Advanced Institute for Artificial Intelligence

Programação em Python

https://advancedinstitute.ai

Programação Python

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

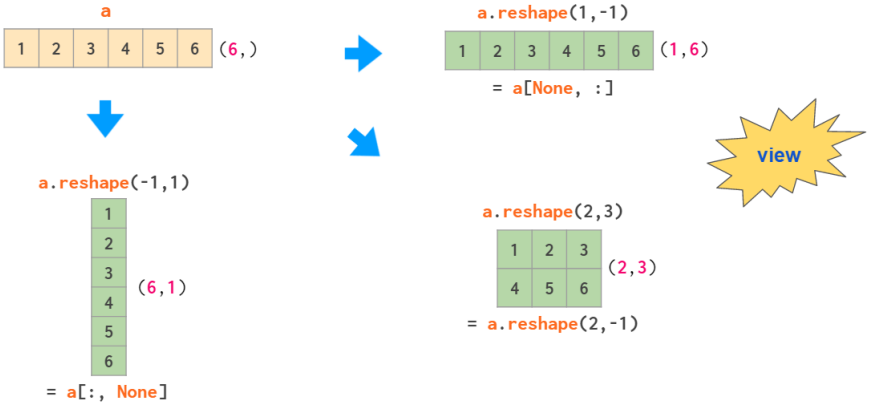
Referências e Fontes das Imagens

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

Vetores-coluna vs Vetores-linha

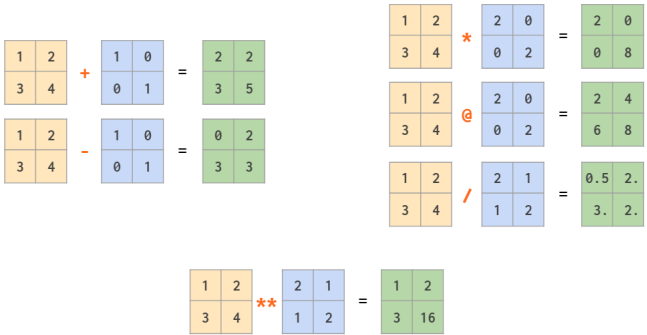
- No contexto 2D, os vetores de linha e coluna são tratados de forma diferente
- Normalmente o NumPy armazena matrizes 1D sempre que possível
- Por padrão, as matrizes 1D são tratadas como vetores de linha em operações 2D *shape* `(n,)` ou `(1,n)`
- Para construir vetores coluna: `np.reshape(-1,1)`,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

Vetores-coluna vs Vetores-linha



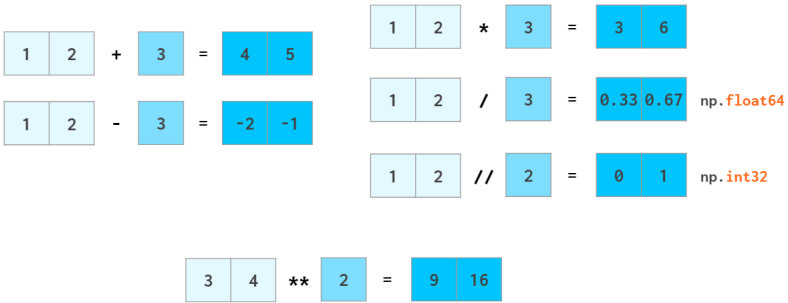
Aritmética de Matrizes

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



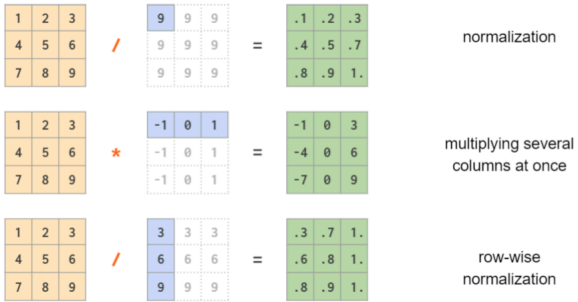
Broadcasting

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



Broadcasting Cont.

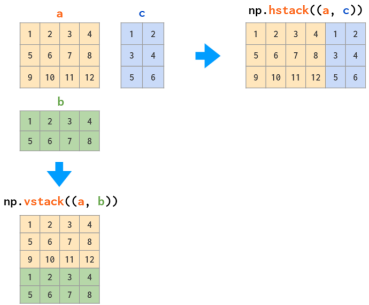
- *Broadcasting* de Matrizes
  - NumPy permite operações mistas entre um vetor e uma matriz, e até mesmo entre dois vetores:



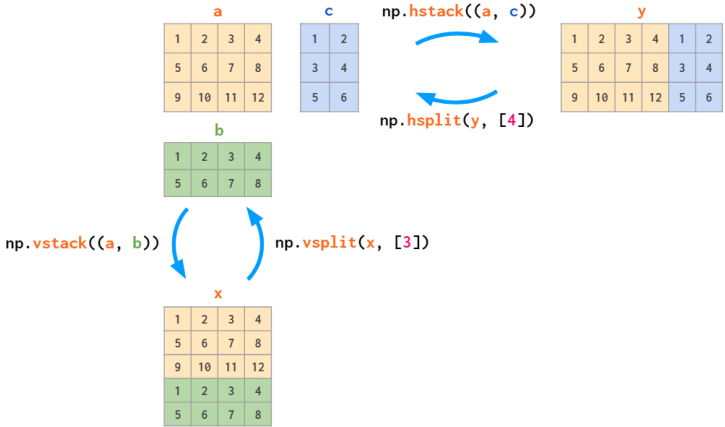
Broadcasting Cont.

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

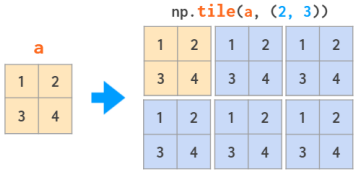
- Unificação na **horizontal** ou na **vertical**
- Cuidado com o shape em vetores coluna

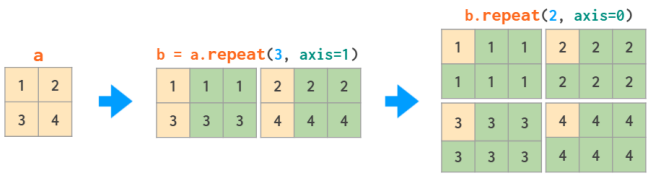


Separação de Matrizes - `hsplit()` e `vsplit()`

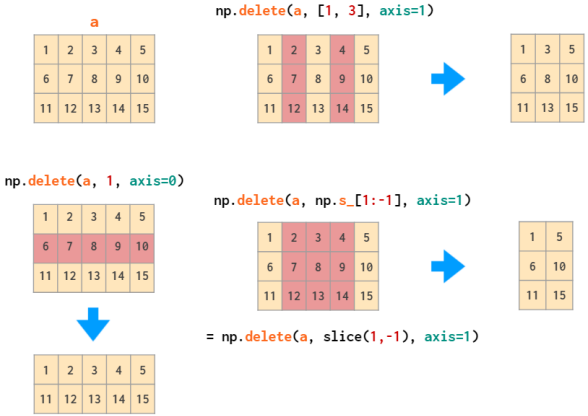


Replicação de Matrizes - `tile()`

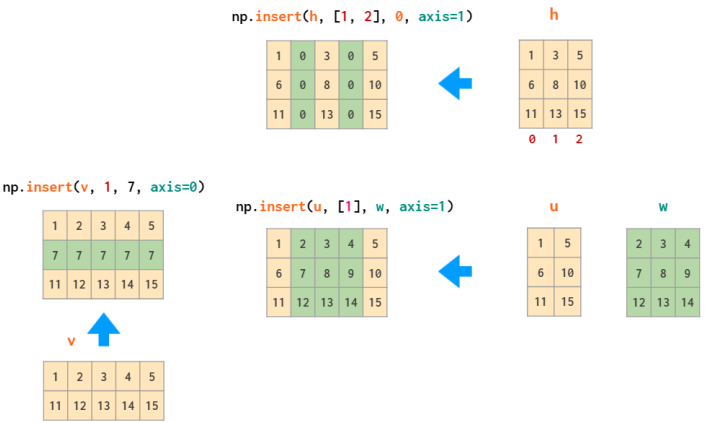




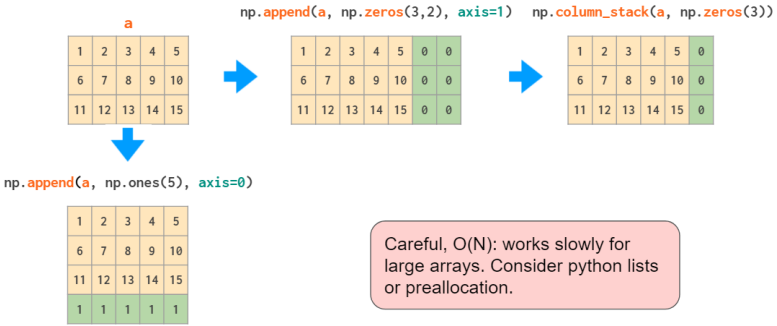
Remoção de linhas e colunas - `delete()`



Inserção de linhas e colunas - `insert()`

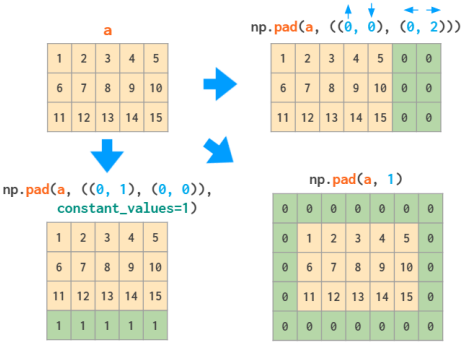


Inserção de linhas e colunas - `append()`



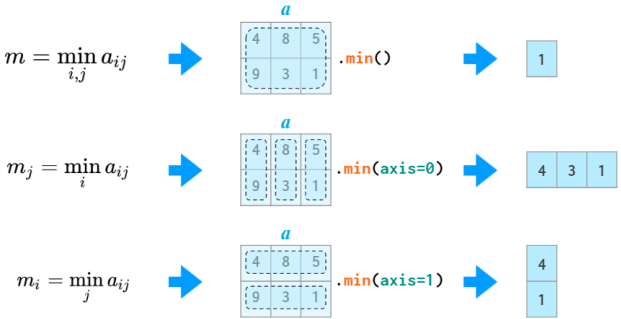
Careful, O(N): works slowly for large arrays. Consider python lists or preallocation.

Inserção de linhas e colunas - `pad()`



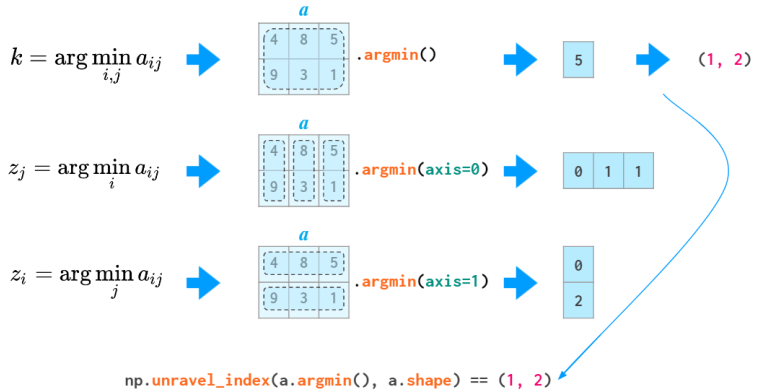
Estatística em Matrizes

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



Estatística em Matrizes

- Retornando as posições:



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

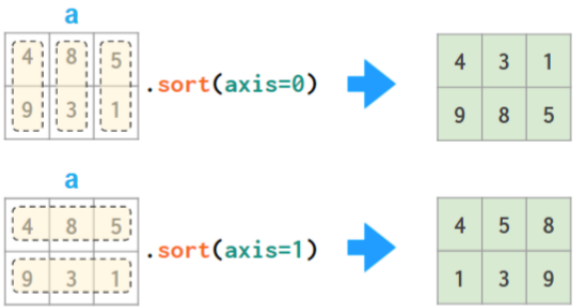
$b = \exists\, i, j \mid a_{ij} > 5$ → `np.any(` *a* [1 2 3 / 4 5 6] `> 5)` → `True`

$b_j = \exists\, i \mid a_{ij} > 5$ → `np.any(` *a* [1 2 3 / 4 5 6] `> 5, axis=0)` → `False` `False` `True`

$b_i = \exists\, j \mid a_{ij} > 5$ → `np.any(` *a* [1 2 3 / 4 5 6] `> 5, axis=1)` → `False` / `True`

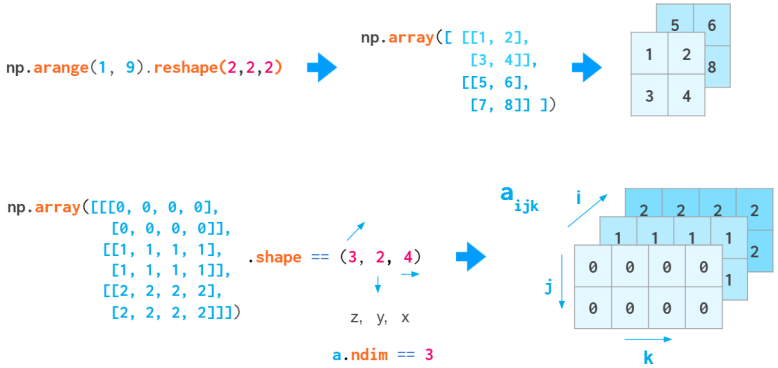
Ordenação de Matrizes

- 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)**

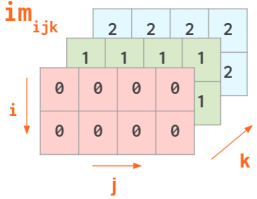


Indexação em 3D

- 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)**

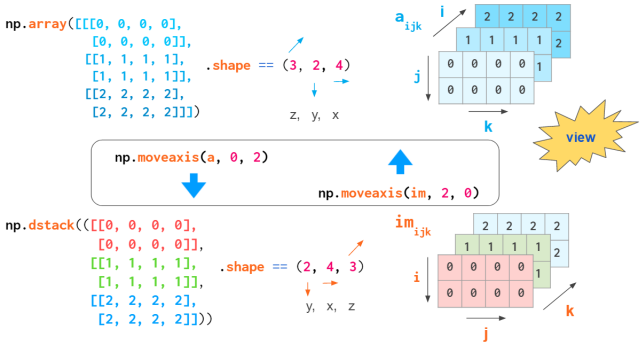
```
np.array([ [[0,1,2], [0,1,2], [0,1,2], [0,1,2]],
           [[0,1,2], [0,1,2], [0,1,2], [0,1,2]] ])   .shape == (2, 4, 3)
                                                          y, x, z
```

```
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?