

Programação em Python

<https://advancedinstitute.ai>



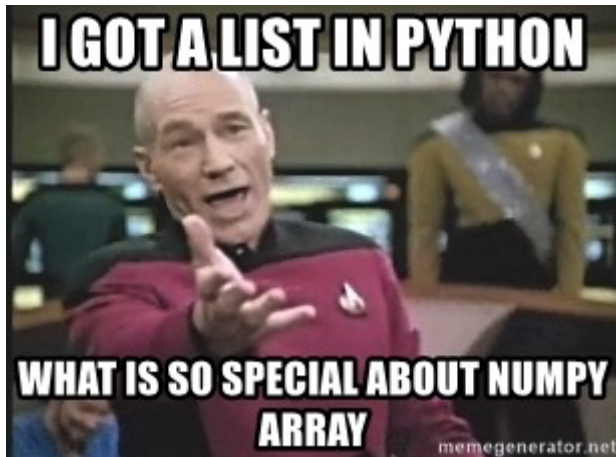
Programação Python

Operações Matriciais com o ***Numpy***

Referências e Fontes das Imagens

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

The Elephant in the Room



Introdução

- ❑ Originalmente como parte do *SciPy* e depois foi destacado como uma **biblioteca fundamental**
- ❑ **Código é mais limpo** do que o código Python “direto” que tenta realizar a mesma tarefa
- ❑ **Menos *loops* necessários**, porque as operações **funcionam diretamente em *arrays* e matrizes**;

Introdução Cont.

- ❑ **Funções matemáticas** tornam a vida mais fácil;
- ❑ Algoritmos subjacentes foram projetados com **alto desempenho** em mente;
- ❑ Os arrays do NumPy são **armazenados de forma mais eficiente**
- ❑ Grandes partes do NumPy são **escritas em C**. Isso torna o NumPy **mais rápido do que o código Python puro**

Armazenamento Otimizado em Memória

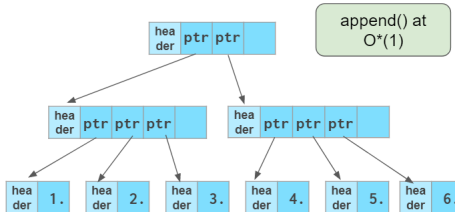
python list

1.	2.	3.
4.	5.	6.

vs

numpy array

1.	2.	3.
4.	5.	6.



append() at $O^*(1)$

append() at $O(N)$

header	1.	2.	3.	4.	5.	6.
--------	----	----	----	----	----	----

Operações diretas

- ❑ Soma de vetores (utilizando listas):

```
1  for i in range(len(a)):  
2      a[i] = i ** 2  
3      b[i] = i ** 3  
4      c.append(a[i] + b[i])
```

- ❑ Utilizando *Numpy Arrays*

```
1  c = a + b
```


Arrays N-dimensionais

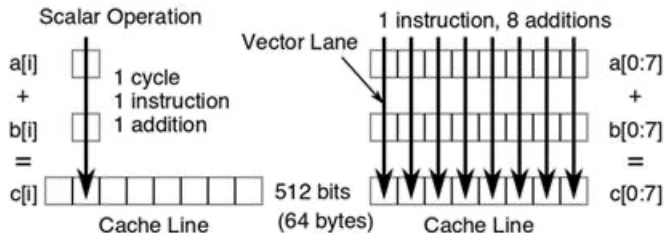
- Arrays de propósito geral e homogêneos;
 - Mais fácil determinar o tamanho de armazenamento necessário;
- Conseguem **vetorizar** operações;
- O número de dimensões e itens em um array é definido por sua forma (atributo **shape**)
 - Tupla de N inteiros não negativos que especificam os tamanhos de cada dimensão.
- Tipos especiais associados aos elementos (**dtype**):
 - `numpy.intc` (`numpy.int32`), `numpy.int_` (`numpy.int64`), `numpy.single` (`numpy.float32`), `numpy.str_`, `numpy.bool_`

Arrays N-dimensionais Cont.

- **ndarrays** podem compartilhar dados, de modo que as alterações feitas em um podem ser visíveis em outro;
 - Um ndarray pode ser uma *view* para outro, i.e., mostrando somente linhas e colunas selecionadas
- Segmento unidimensional contíguo de memória de computador;
 - Esquema de indexação que mapeia N inteiros para a localização de um item no bloco
 - **Stride** (*offset*)
- Múltiplas formas de inicialização

Vetorização de operações

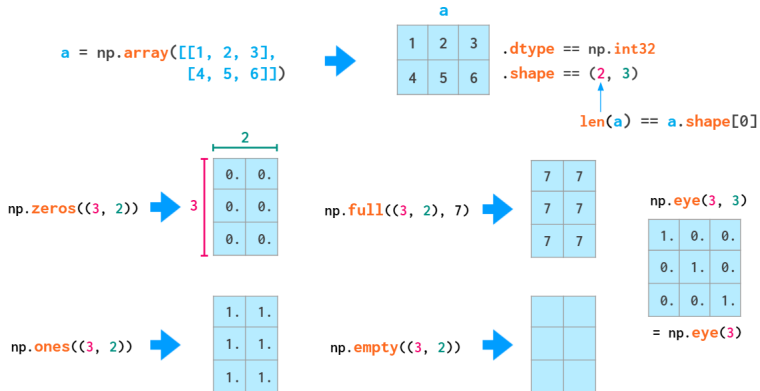
- Paradigma *Single Instruction Multiple Data* - SIMD
 - Forma de **paralelismo de dados**



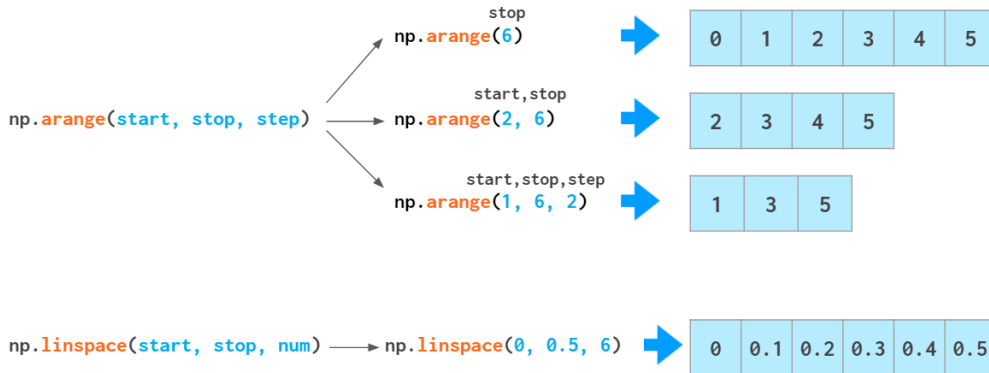
Criação de ndarrays

- ❑ `numpy.array` - Converte outro objeto (list,tupla, etc) para `ndarray`
- ❑ `numpy.empty` - constrói um `ndarray` sem inicializar seus elementos;
- ❑ `numpy.eye` - constrói um `ndarray` com valores `1` na sua diagonal (identidade)
- ❑ `numpy.ones`- cria matriz de tamanho `n` apenas com valores `1`
- ❑ `numpy.zeros` - cria `ndarray` de tamanho `n` apenas com valores `0`
- ❑ `numpy.full` - cria `ndarray` preenchido com valor específico
- ❑ Criação a partir de um protótipo: `numpy.ones_like`, `numpy.zeros_like`, `numpy.empty_like`, `numpy.full_like`

Criação de ndarrays



Inicialização monotônica de ndarrays



Inicialização aleatória de ndarrays

```
rng = np.random.default_rng()
```

```
rng.integers(0, 10, 3)
```

uniform, $x \in [0, 10)$



4	3	7
---	---	---



```
rng.integers(0, 10, 3, endpoint=True)
```

uniform, $x \in [0, 10]$

```
rng.random(3)
```

uniform, $x \in [0, 1)$



0.7	0.3	0.8
-----	-----	-----

```
rng.standard_normal(3)
```

normal, $\mu=0$, $\sigma=1$



0.4	-1.1	0.8
-----	------	-----

```
rng.uniform(1, 10, 3)
```

uniform, $x \in [1, 10)$



5.1	2.7	7.2
-----	-----	-----

```
rng.normal(5, 2, 3)
```

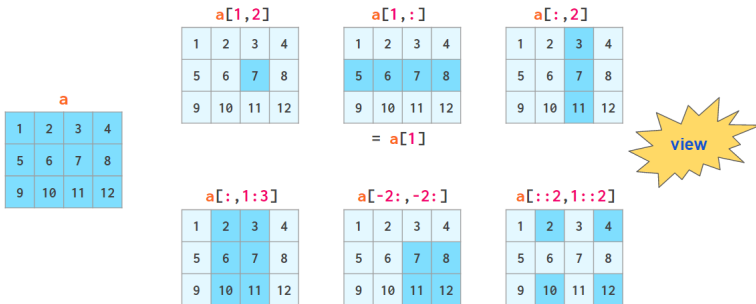
normal, $\mu=5$, $\sigma=2$



4.5	3.2	6.7
-----	-----	-----

Slicing

- Para o caso de *arrays* NumPy unidimensionais funciona exatamente como o *slicing* de listas Python
- Sintaxe similar para mais de uma dimensão



Indexação utilizando booleanos

a	1	2	3	4	5	6	7	6	5	4	3	2	1
a > 5	False	False	False	False	False	True	True	True	False	False	False	False	False

`np.any(a > 5)`

True

`a[a > 5]`

6 7 6

`np.all(a > 5)`

False

`a[a > 5] = 0`

a	1	2	3	4	5	0	0	0	5	4	3	2	1
----------	---	---	---	---	---	---	---	---	---	---	---	---	---

`a[(a >= 3) & (a <= 5)] = 0`

a	1	2	0	0	0	6	7	6	0	0	0	2	1
----------	---	---	---	---	---	---	---	---	---	---	---	---	---

& and
| or
^ xor
~ not

Propriedades do ndarray

- ❑ **T**: matriz transposta
- ❑ **dtype**: tipo de dados armazenado na matriz
- ❑ **size**: quantidade de elementos na matriz
- ❑ **shape**: dimensões da matriz
- ❑ **ndim**: número de dimensões da matriz
- ❑ **nbytes**: número de *bytes* utilizados para armazenar a matriz em memória
- ❑ **imag**: a parte imaginária dos elementos (complexos) da matriz
- ❑ **real**: a parte real dos elementos (complexos) armazenados na matriz

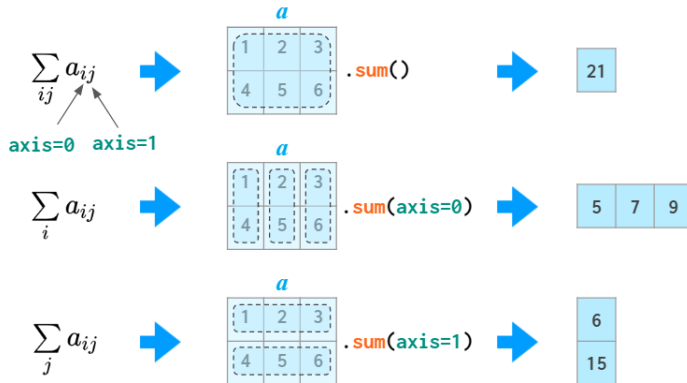
Algumas operações interessantes

- ❑ `max()`: retorna o maior elemento da matriz
- ❑ `min()`: retorna o menor elemento da matriz
- ❑ `sum()`: retorna a soma dos elementos da matriz
- ❑ `mean()`: retorna a média dos elementos da matriz
- ❑ `var()`: retorna a variância dos elementos da matriz
- ❑ `std`: retorna o desvio padrão dos elementos na matriz
- ❑ `prod()`: retorna o produto dos elementos da matriz
- ❑ `nonzero()`: retorna os elementos da matriz cujos valores são diferentes de zero

Eixos

- *axis* são definidos para matrizes com mais de uma dimensão
- O número do índice em questão: O primeiro índice é eixo = 0, o segundo é eixo = 1 e assim por diante;
- Uma matriz bidimensional possui dois eixos correspondentes:
 - O primeiro correndo verticalmente para baixo nas linhas (eixo 0)
 - O segundo correndo horizontalmente nas colunas (eixo 1)

Eixos



Alterando o formato de um ndarray

- ❑ Operação `reshape()`: criação de novo `ndarray` com novo shape.
 - Para versão *inplace*, utilizar `resize()`

```
1 >>> a = np.arange(1,10)
2 array([1, 2, 3, 4, 5, 6, 7, 8, 9])
3 >>> a.shape
4 (9,)
5 >>> a.reshape(3,3)
6 array([[1, 2, 3],
7         [4, 5, 6],
8         [7, 8, 9]])
```

Dúvidas?