



# *Introdução ao Numerical Python (Numpy)*

CONSTRUÇÃO E ANÁLISE  
DE ALGORITMOS

Prof. Elias Paulino Medeiros

Instituto Federal do Ceará (IFCE)

# Introdução

Scalar

Vector

Matrix

Tensor

1

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

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

$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

# Principais Características do NumPy

É um pacote necessário para pesquisa científica em python;

O objeto principal do NumPy é o array multidimensional;

Armazenam elemento de um mesmo tipo;

Indexado por uma tupla de inteiros não negativos;

....

# Principais Características do NumPy

Têm tamanho fixo na criação;

Facilitam operações matemáticas avançadas e outros tipos de operações em um grande número de dados;

O array de NumPy é chamada ***ndarray***.

# Como instalar?

O NumPy pode ser instalado com o **conda**, com **pip**, ou com um gerenciador de pacotes no **MacOS e Linux**.

```
conda install numpy
```

```
pip install numpy
```

# Criação do array

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
```

```
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
```

# Criação do array

```
>>> import numpy as np
>>> a = np.array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> a = np.array(1, 2, 3, 4)    # WRONG
```

```
Traceback (most recent call last):
```

```
...
```

```
TypeError: array() takes from 1 to 2 positional arguments but 4 were given
```

```
>>> a = np.array([1, 2, 3, 4]) # RIGHT
```

# Definindo o tipo de dado

```
>>> c = np.array([[1, 2], [3, 4]], dtype=complex)
>>> c
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```



# Funções zeros e ones

```
>>> np.zeros((3, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> np.ones((2, 3, 4), dtype=np.int16)
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],
       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
>>> np.empty((2, 3))
array([[3.73603959e-262, 6.02658058e-154, 6.55490914e-260], # may vary
       [5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])
```

# Operações básicas

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.arange(4)
>>> b
array([0, 1, 2, 3])
>>> c = a - b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10 * np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a < 35
array([ True,  True, False, False])
```

# Operações básicas

```
>>> A = np.array([[1, 1],
...               [0, 1]])
>>> B = np.array([[2, 0],
...               [3, 4]])
>>> A * B      # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B      # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)   # another matrix product
array([[5, 4],
       [3, 4]])
```

# Operações básicas

```
>>> rg = np.random.default_rng(1) # create instance of default random number generator
>>> a = rg.random((2, 3))
>>> a
array([[0.82770259, 0.40919914, 0.54959369],
       [0.02755911, 0.75351311, 0.53814331]])
>>> a.sum()
3.1057109529998157
>>> a.min()
0.027559113243068367
>>> a.max()
0.8277025938204418
```

# Operações básicas

```
>>> b = np.arange(12).reshape(3, 4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)      # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)      # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)   # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

# Indexando e Fatiando arrays

```
>>> a = np.arange(10)**3
>>> a
array([ 0,  1,  8, 27, 64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> # equivalent to a[0:6:2] = 1000;
>>> # from start to position 6, exclusive, set every 2nd element to 1000
>>> a[:6:2] = 1000
>>> a
array([1000,  1, 1000,  27, 1000, 125, 216, 343, 512, 729])
>>> a[::-1] # reversed a
array([ 729, 512, 343, 216, 125, 1000,  27, 1000,  1, 1000])
```

# Mudando a forma de um array

```
>>> a.ravel() # returns the array, flattened  
array([3., 7., 3., 4., 1., 4., 2., 2., 7., 2., 4., 9.])  
>>> a.reshape(6, 2) # returns the array with a modified shape  
array([[3., 7.],  
       [3., 4.],  
       [1., 4.],  
       [2., 2.],  
       [7., 2.],  
       [4., 9.]])
```

```
>>> a.T # returns the array, transposed  
array([[3., 1., 7.],  
       [7., 4., 2.],  
       [3., 2., 4.],  
       [4., 2., 9.]])  
>>> a.T.shape  
(4, 3)  
>>> a.shape  
(3, 4)
```



# Empilhando arrays

```
>>> a = np.floor(10 * rg.random((2, 2)))
>>> a
array([[9., 7.],
       [5., 2.]])
>>> b = np.floor(10 * rg.random((2, 2)))
>>> b
array([[1., 9.],
       [5., 1.]])
>>> np.vstack((a, b))
array([[9., 7.],
       [5., 2.],
       [1., 9.],
       [5., 1.]])
>>> np.hstack((a, b))
array([[9., 7., 1., 9.],
       [5., 2., 5., 1.]])
```



# Filtrando arrays

```
>>> a = np.arange(12).reshape(3, 4)
>>> b = a > 4
>>> b # `b` is a boolean with `a`'s shape
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b] # 1d array with the selected elements
array([ 5,  6,  7,  8,  9, 10, 11])
>>> a[b] = 0 # ALL elements of `a` higher than 4 become 0
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
```

# Fatiando arrays

```
>>> a = np.arange(12).reshape(3, 4)
>>> b1 = np.array([False, True, True])
>>> b2 = np.array([True, False, True, False])
>>>
>>> a[b1, :]
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> a[b1]
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>> a[:, b2]
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
>>>
>>> a[b1, b2]
array([ 4, 10])
```

# Lendo arquivos de dados



arquivo\_de\_dados.txt

```
>> data = np.genfromtxt('arquivo_de_dados.txt', delimiter=',')
```

**delimiter** é o caractere que delimita os campos de dados do arquivos dos dados. Geralmente é usado virgula (','), espaço em braço (' ') ou ponto e virgula(';').

```
array([[0, 1, 1, 1],  
       [0, 1, 3, 1],  
       [3, 4, 1, 3],  
       [0, 1, 2, 0],  
       [2, 1, 0, 0]])
```

	0	1	2	3
0	0	1	1	1
1	0	1	3	1
2	3	4	1	3
3	0	1	2	0
4	2	1	0	0

# Manipulando Numpy arrays

```
>> data[:,0][data[:,0] == 0] = -1 #(substituindo valor de uma coluna)
```

```
array([[ -1,  1,  1,  1],  
       [ -1,  1,  3,  1],  
       [  3,  4,  1,  3],  
       [ -1,  1,  2,  0],  
       [  2,  1,  0,  0]])
```

	0	1	2	3
0	-1	1	1	1
1	-1	1	3	1
2	3	4	1	3
3	-1	1	2	0
4	2	1	0	0

```
>> nodes, counts = np.unique(data[:,1], return_counts=True) #(contando a quant. de diferentes valores em uma coluna)
```

```
>> nodes
```

```
array([1, 4])
```

```
>> counts
```

```
array([4, 1], dtype=int64)
```

# Manipulando Numpy arrays

```
>> data = data.astype('float') #(mudando o tipo de dados do array)
```

```
>> data
```

```
array([[ -1.,  1.,  1.,  1.],  
       [ -1.,  1.,  3.,  1.],  
       [  3.,  4.,  1.,  3.],  
       [ -1.,  1.,  2.,  0.],  
       [  2.,  1.,  0.,  0.]])
```

```
>> data[1,1] = np.nan #(inserindo um valor nan no array)
```

```
>> data
```

```
array([[ -1.,  1.,  1.,  1.],  
       [ -1., nan,  3.,  1.],  
       [  3.,  4.,  1.,  3.],  
       [ -1.,  1.,  2.,  0.],  
       [  2.,  1.,  0.,  0.]])
```

```
>> np.isnan(data)
```

```
True
```

```
>> data[~np.isnan(data).any(axis=1)] #(removendo a linha com valor nan)
```