Introdução à NumPy

Prof. Vicente Helano

UFCA | Centro de Ciências e Tecnologia

Importando a NumPy

Importando a NumPy

In [1]:
```
import numpy as np # chamando a numpy de np
```

Tipos de dados

Seus tipos primitivos são:

Tipos de dados

Seus tipos primitivos são:

`bool` `int32` `int64` `uint32` `uint64` `float32` `float64` `complex64` `complex128`

Tipos de dados

mas o mais útil é o:

`numpy.array`

Constantes

A NumPy possui diversas constantes matemáticas pré-definidas.

Constantes

A NumPy possui diversas constantes matemáticas pré-definidas.

```
In [2]: np.pi,np.e

Out[2]: (3.141592653589793, 2.718281828459045)
```

Funções matemáticas

Também há diversas funções prontas:

Funções matemáticas

Também há diversas funções prontas:

In [3]:
```
teta = np.pi/2

print("%25.20e" % np.sin(teta))
```

```
1.00000000000000000000e+00
```

Lidando com arranjos

A `numpy.array` é um arranjo numérico **homogêneo**. Apesar de ser um tipo *mutável*, o tamanho de um arranjo NumPy é **imutável** e, por isso, não pode ser vazio.

Construção de arranjos

A partir de listas de listas:

Construção de arranjos

A partir de listas de listas:

In [4]:
```
# Matrizes: [ [linha 0], [linha 1], ..., [linha n-1] ]
A = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])
A
```

Out[4]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Construção de arranjos

A partir de outro `numpy.array`:

Construção de arranjos

A partir de outro `numpy.array`:

In [5]:
```
B = np.array(A) # cópia de A
B,A
```

Out[5]:
```
(array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]),
 array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]))
```

Construção de arranjos

Usando funções *especiais*:

Construção de arranjos

Usando funções *especiais*:

In [6]:
```
np.arange(10) # a sintaxe é semelhante à da `range`
```

Out[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Construção de arranjos

Usando funções *especiais*:

Construção de arranjos

Usando funções *especiais*:

In [7]:
```
O = np.zeros((4,5),dtype=float) # o par (4, 5) indica os números de linhas e colu
I = np.eye(4) # identidade
U = np.ones((4,5)) # 1's
O,I,U
```

Out[7]:
```
(array([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]),
 array([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]]),
 array([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]))
```

Construção de arranjos

Usando funções *especiais*:

In [8]:
```
np.linspace(-1, 1, 10)
```

Out[8]:
```
array([-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
        0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ])
```

Alterando as dimensões de um arranjo

In [9]:
```
A = np.array([[1,3.5],[-1,0],[1,3.]])
A
```

Out[9]:
```
array([[ 1. ,  3.5],
       [-1. ,  0. ],
       [ 1. ,  3. ]])
```

Alterando as dimensões de um arranjo

In [9]:
```
A = np.array([[1,3.5],[-1,0],[1,3.]])
A
```

Out[9]:
```
array([[ 1. ,  3.5],
       [-1. ,  0. ],
       [ 1. ,  3. ]])
```

In [10]:
```
A.size, A.shape
```

Out[10]:
```
(6, (3, 2))
```

Alterando as dimensões de um arranjo

In [11]:
```
A = np.linspace(1.0, 4.0, 16)
A
```

Out[11]:
```
array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. , 3.2, 3.4,
       3.6, 3.8, 4. ])
```

Alterando as dimensões de um arranjo

In [12]:
```
A.reshape(2,8)
```

Out[12]:
```
array([[1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4],
       [2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. ]])
```

Alterando as dimensões de um arranjo

In [13]:
```
A.flatten()
```

Out[13]:
```
array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. , 3.2, 3.4,
       3.6, 3.8, 4. ])
```

Acessando elementos de um arranjo

In [14]:
```
A = np.array([[8, 2, 3],[9, 7, 2],[5, 5, 7],[4, 5, 9]])
A
```

Out[14]:
```
array([[8, 2, 3],
       [9, 7, 2],
       [5, 5, 7],
       [4, 5, 9]])
```

Acessando elementos de um arranjo

In [14]:
```
A = np.array([[8, 2, 3],[9, 7, 2],[5, 5, 7],[4, 5, 9]])
A
```

Out[14]:
```
array([[8, 2, 3],
       [9, 7, 2],
       [5, 5, 7],
       [4, 5, 9]])
```

In [15]:
```
A[3,2]
```

Out[15]:
```
9
```

Operações com arranjos

In [16]:
```
A = np.array([[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
print(A)
print(B)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
[[7 1 3 0]
 [6 0 2 6]
 [2 2 1 9]
 [8 6 7 1]]
```

Operações com arranjos

In [16]:
```
A = np.array([[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
print(A)
print(B)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
[[7 1 3 0]
 [6 0 2 6]
 [2 2 1 9]
 [8 6 7 1]]
```

In [17]:
```
B+1 == A+B # soma de matrizes
```

Out[17]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Operações com arranjos

In [18]:
```
A = np.array([[1, 1, 1, 1],[2, 2, 2, 2],[3, 3, 3, 3],[4, 4, 4, 4]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
```

Operações com arranjos

In [18]:
```
A = np.array([[1, 1, 1, 1],[2, 2, 2, 2],[3, 3, 3, 3],[4, 4, 4, 4]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
```

In [19]:
```
A*B # produto coeficiente a coeficiente cij = aij * bij
```

Out[19]:
```
array([[ 7,  1,  3,  0],
       [12,  0,  4, 12],
       [ 6,  6,  3, 27],
       [32, 24, 28,  4]])
```

Operações com arranjos

Produto matricial $\mathbf{AB}$:

In [20]:
```
A = np.array([[1, 1, 1, 1],[2, 2, 2, 2],[3, 3, 3, 3],[4, 4, 4, 4]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
```

Operações com arranjos

Produto matricial $\mathbf{AB}$:

In [20]:
```
A = np.array([[1, 1, 1, 1],[2, 2, 2, 2],[3, 3, 3, 3],[4, 4, 4, 4]])
B = np.array([[7, 1, 3, 0],[6, 0, 2, 6],[2, 2, 1, 9],[8, 6, 7, 1]])
```

In [21]:
```
np.dot(A,B) # Resultado = AB
```

Out[21]:
```
array([[23,  9, 13, 16],
       [46, 18, 26, 32],
       [69, 27, 39, 48],
       [92, 36, 52, 64]])
```

Operações com arranjos

Se dois arranjos $\mathbf{v}$ e $\mathbf{w}$ forem unidimensionais (vetor linha), o resultado será o produto escalar entre eles:

In [22]:
```
v = np.array([1, 1, 1, 1])
w = np.array([7, 1, 3, 0])
```

Operações com arranjos

Se dois arranjos $\mathbf{v}$ e $\mathbf{w}$ forem unidimensionais (vetor linha), o resultado será o produto escalar entre eles:

In [22]:
```
v = np.array([1, 1, 1, 1])
w = np.array([7, 1, 3, 0])
```

In [23]:
```
np.dot(v,w) # Resultado = <u,v>
```

Out[23]: 11

Exercício

(a) Aplique o que você aprendeu sobre arranjos da numpy para completar a função `matpot` a seguir. Ela deve **retornar** a k-ésima potência de uma matriz $\mathbf{A}$, $n \times n$. Você deve realizar o produto de matrizes utilizando a função `np.dot`.

Exercício

In [24]:

```
def matpot(A,k):
    """
    Calcula a k-ésima potência da matriz A.

       Argumentos:
         A (numpy.array): matriz quadrada armazenada usando o `array` da numpy.
         k (int): valor da potência desejada, k >= 0.

       Retorno:
         (numpy.array): o valor de A^k
    """
    # você deve iniciar a implementação desta função a partir daqui.
```

Exercício

(b) Aplique sua implementação da `matpot` com $k = 30$ e a matriz:

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

In [25]:
```
# Escreva o código de verificação aqui embaixo.
```

Exercício

(b) Aplique sua implementação da `matpot` com $k = 30$ e a matriz:

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

In [25]:
```
# Escreva o código de verificação aqui embaixo.
```

Exercício

(b) Aplique sua implementação da `matpot` com $k = 30$ e a matriz:

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

In [25]:
```
# Escreva o código de verificação aqui embaixo.
```

Exercício

(c) Você percebeu algum fato interessante? Discuta com seus colegas.

Saiba mais

- Ao longo do curso, aprenderemos muito mais sobre a NumPy. Se você ficou curioso e quer saber mais sobre ela, sugiro que acesse: https://www.numpy.org/

- As matrizes $\mathbf{B}$ e $\mathbf{C}$ acima recebem o nome de *idempotente* e *nilpotente*, respectivamente. Assista ao vídeo abaixo sobre algumas das matrizes especiais que veremos ao longo de nosso curso:

In [27]:
```
from IPython.display import HTML, IFrame
IFrame(src="https://www.youtube.com/embed/Nhw0kIJwEvs", width=320, height=240)
```

Vicente Helano
UFCA | Centro de Ciências e Tecnologia