

# NumPy

Paulo Almeida  
André Grégio



**JUSTIÇA 4.0: INOVAÇÃO E EFETIVIDADE NA REALIZAÇÃO DA JUSTIÇA PARA TODOS**  
PROJETO DE EXECUÇÃO NACIONAL BRA/20/015

# 1. NumPy



# NumPy

NumPy

**Num**erical **Py**thon

Biblioteca para Python para lidar com vetores e matrizes  
“grandes” e de múltiplas dimensões

Mitiga muitas das ineficiências computacionais do Python  
quando lidando com esse tipo de dado

# NumPy

## Algumas vantagens do NumPy

- ▶ Estrutura de dados eficiente para armazenar e operar em **arrays** (vetores/matrizes)
- ▶ Funções matemáticas prontas para operar em arrays
- ▶ Funções para carga e descarga de dados de/para o disco
- ▶ Funções de álgebra, geração de números aleatórios, transformadas, ...
- ▶ ...

# Uso

Depois de instalar o NumPy, basta importar

Dica: instale via pip, veja o tutorial

```
import numpy as np
```

# Uso

Depois de instalar o NumPy, basta importar

Dica: instale via pip, veja o tutorial

Importe a biblioteca numpy

```
import numpy as np
```

Os itens no escopo da biblioteca  
numpy serão chamados por np  
(criamos um *alias*)

# Função Array

A função `array`(sequencia de itens) aceita qualquer sequência de itens, como uma lista, para criar um array NumPy

# Função Array

A função array (sequencia de itens) aceita qualquer sequência de itens, como uma lista, para criar um array NumPy

Um array NumPy em princípio é similar a uma lista

Exemplo: os itens possuem um índice, partindo do zero

Os arrays NumPy são chamados de **ndarrays**



# Exemplo

```
import numpy as np
```

```
#criando a partir de uma lista
```

```
lista = [1,2,3]
```

```
array1 = np.array(lista)
```

```
print("Elemento em zero: ", array1[0])
```

```
for item in array1:
```

```
    print(item)
```

```
#criando diretamente
```

```
array2 = np.array([4,5,6])
```

```
for item in array2:
```

```
    print(item)
```

# Matrizes

O conceito de matrizes no NumPy é similar ao aprendido para listas tradicionais

Exemplo:

	Coluna 0	Coluna 1	Coluna 2
Linha 0	4	5	6
Linha 1	7	8	9

# Acesso

É possível iterar nos itens da matriz, ou acessar diretamente via o operador []

Exemplo:

```
import numpy as np
```

```
#criando diretamente
```

```
array = np.array([[4,5,6],[7,8,9]])
```

```
for linha in array:
```

```
    for coluna in linha:
```

```
        print(coluna, end=' ')
```

```
    print()
```

```
print("Item na linha 1, coluna 2: ", array[1][2])
```

# Propriedades

Arrays NumPy possuem algumas **propriedades** que podem ser acessadas

Exemplos:

- dtype: qual o tipo dos dados do array

- shape: qual o formato (dimensões) do array

- size: tamanho total do array

- ndim: número de dimensões

# Exemplo

```
import numpy as np
```

```
vetor = np.array([1.7,2,3,4])  
matriz = np.array([[4,5,6],[7,8,9]])
```

```
print(vetor.ndim, vetor.shape, vetor.dtype, vetor.size)  
print(matriz.ndim, matriz.shape, matriz.dtype, matriz.size)
```

## 2. Inicialização de Arrays



## Vazio e uns

A função `zeros(shape, dtype=float)` preenche um vetor com zeros

O vetor tem dimensões `shape`

A função `ones(shape, dtype=float)` faz o mesmo, mas preenche o vetor com uns

## Vazio e uns

Por padrão os valores serão floats. O tipo dos valores pode ser modificado pelo parâmetro dtype

Veja uma listagem de tipos válidos aqui

[numpy.org/devdocs/user/basics.types.html](https://numpy.org/devdocs/user/basics.types.html)



# Exemplo

```
import numpy as np
```

```
vetor_zeros = np.zeros(4)
```

```
matriz_zeros = np.zeros((3,2))
```

```
matriz_uns = np.ones((5,5), dtype=np.intc)
```

```
print(vetor_zeros)
```

```
print(matriz_zeros)
```

```
print(matriz_uns)
```

# full

`full(shape, fill_value, dtype=None)` cria um vetor preenchido com `fill_value`

```
import numpy as np
```

```
matriz_seis = np.full((5,5), 6, dtype=np.intc)
```

```
print(matriz_seis)
```

# empty

A função `empty(shape, dtype=float)` criar um array de dimensões `shape` sem inicializar os valores

Não é possível saber os valores atribuídos a cada posição do vetor

## Lixo de memória

Custa mais barato chamar `empty` do que `zero` ou `ones`, por exemplo

Útil quando precisamos criar um array que vamos preencher posteriormente

# arange

A função `arange(start, stop, step)` retorna um vetor gerado no intervalo `[start, stop)`

Inicia em `start` e termina em `stop` (sem incluir o `stop`)  
Cada valor entre `start` e `stop` tem uma distância `step`

# linspace

`linspace(start, stop, num)` retorna um vetor gerado no intervalo `[start, stop]`

O vetor vai possuir `num` valores igualmente espaçados

# Exemplo

```
import numpy as np
```

```
vetor1 = np.arange(10, 20, 2)
```

```
print(vetor1)
```

```
vetor2 = np.linspace(10, 20, 5)
```

```
print(vetor2)
```

# Aleatório

É possível preencher um array com valores aleatórios chamando `np.random.rand(d0, d1, ..., dn)`

`d0, d1, ...` especifica o tamanho das dimensões do array

Os números são gerados a partir de uma distribuição uniforme

Os números gerados estarão no intervalo  $[0, 1)$

```
import numpy as np
```

```
matrizRand = np.random.rand(2, 3)  
print(matrizRand)
```

```
vetRand = np.random.rand(5)  
print(vetRand)
```

# Aleatório com inteiros

`random.randint(low, high=None, size=None)` é similar a `rand`, mas preenche o array com valores inteiros

```
import numpy as np
```

```
dados = np.random.randint(0,100,(5,6))
```

```
print(dados)
```



## Outras distribuições

Existem funções prontas para outras distribuições, como Poisson: `np.random.poisson(lam=1.0, size=None)`

`lam` é o valor de  $\lambda$

`size` é uma tupla indicando as dimensões do Array

```
import numpy as np
```

```
poisson = np.random.poisson(lam=5.0, size=(2,2))  
print(poisson)
```

### 3. Alterando os Arrays



## Outras distribuições

Como em listas, é possível acessar e modificar os itens diretamente através do operador []

É possível também modificar todos os elementos do array (ou de uma única linha de uma matriz por exemplo) combinando o operador [] com os operadores:

- $+=$  X some o valor atual do elemento com X
- $-=$  X decremente o valor atual do elemento com X
- $*=$  X multiplique o valor atual do elemento com X
- $/=$  X divida o valor atual do elemento com X

# Exemplo

```
import numpy as np
```

```
matriz = np.ones((4,4), dtype=int)
```

```
matriz[0][0] = 33
```

```
matriz += 14
```

```
matriz[1] *= 2
```

```
print(matriz)
```

# Exemplo

```
import numpy as np
```

```
matriz = np.ones((4,4), dtype=int)
```

```
matriz[0][0] = 33
```

```
matriz += 14
```

```
matriz[1] *= 2
```

Elemento na linha 0, coluna 0,  
recebe 33

```
print(matriz)
```

# Exemplo

```
import numpy as np
```

```
matriz = np.ones((4,4), dtype=int)
```

```
matriz[0][0] = 33
```

```
matriz += 14
```

```
matriz[1] *= 2
```

Some 14 em todos elementos

```
print(matriz)
```

# Exemplo

```
import numpy as np
```

```
matriz = np.ones((4,4), dtype=int)
```

```
matriz[0][0] = 33
```

```
matriz += 14
```

```
matriz[1] *= 2
```

```
print(matriz)
```

Multiplique a linha 1 por 2

# reshape

É possível modificar as dimensões de um determinado array chamando `reshape(newshape)`

A função vai retornar uma variável contendo uma visão com as dimensões rearranjadas

As dimensões são uma tupla passada para `newshape`



# Exemplo

```
import numpy as np

dados = np.linspace(2, 32, 16, dtype=np.intc)
print(dados.shape)
print(dados)

alterado = dados.reshape((8,2))
print(alterado.shape)
print(alterado)
```

# Deduzir dimensões

A função reshape aceita que seja atribuído -1 a uma das dimensões.

Nesse caso, a própria função vai inferir automaticamente o valor dessa dimensão

```
import numpy as np
```

```
dados = np.linspace(2, 32, 16, dtype=np.intc)  
alterado = dados.reshape((-1,2))
```

```
print(alterado)
```

# Cuidado

A função reshape retorna uma variável que contém uma nova visão, mas que compartilha os dados

Modificar os dados de uma visão, automaticamente todas as visões que usam esses dados

```
import numpy as np
```

```
dados = np.linspace(2, 32, 16, dtype=np.intc)  
alterado = dados.reshape((-1,2))
```

```
dados[0] = -5  
print(dados)  
print(alterado)
```

# 4.

## Funções matemáticas



# Funções matemáticas

Existem diversas funções matemáticas prontas para operar em arrays NumPy

Veja uma lista aqui

[numpy.org/doc/stable/reference/routines.math.html](https://numpy.org/doc/stable/reference/routines.math.html)

Vamos ver alguns exemplos

# sum

A função `sum()` retorna a soma dos elementos em um array

```
import numpy as np

dados = np.ones((4,3))
print(dados)
soma = dados.sum()
print(soma)
```

# sum

Opcionalmente, pode-se definir a dimensão na qual a soma será feita `sum(dim)`

Exemplo

Dimensão 1

Dimensão 0

1	1	1
1	1	1
1	1	1
1	1	1

# Exemplo

Saída do programa:

[4. 4. 4.]

[3. 3. 3. 3.]

```
import numpy as np
```

```
dados = np.ones((4,3))
```

```
soma = dados.sum(0)
```

```
print(soma)
```

```
soma = dados.sum(1)
```

```
print(soma)
```

Dimensão 0

Dimensão 1

A diagram showing a 4x3 grid of cells, each containing the number 1. A vertical purple arrow on the left points downwards and is labeled 'Dimensão 0'. A horizontal purple arrow on the top points to the right and is labeled 'Dimensão 1'.

1	1	1
1	1	1
1	1	1
1	1	1



# sum

É possível também executar a soma para uma linha específica

Exemplo:

```
import numpy as np
```

```
dados = np.ones((4,3))
```

```
linha0 = dados[0].sum()
```

```
print(linha0)
```

# diff

`np.diff(array)` faz o cálculo `array[n] - array[n-1]`, ou seja, faz o cálculo da diferença entre os elementos com `lag=1`

É possível especificar outros lags

```
import numpy as np
```

```
dados = np.random.randint(0,10,(5,6))
```

```
print(dados)
```

```
print(np.diff(dados))
```

## cumsum

`np.cumsum(array)` faz a soma cumulativa dos elementos do array

```
import numpy as np
```

```
dados = np.ones((3,3))
```

```
print(dados)
```

```
print(np.cumsum(dados))
```

5.

# Lendo e Escrevendo CSVs com NumPy



# Leitura de CSVs

Existem várias formas para se carregar o CSV para o NumPy

Exemplo

Usando o que foi aprendido em aulas passadas, iterar no arquivo csv para carregar um *ndarray* manualmente

## genfromtxt

O NumPy possui a função `np.genfromtxt(caminhoArquivo, delimiter= delim, skip_header = skip, usecols = cols, dtype=tipos)`

Faz a carga automática de dados do csv

`delim`: delimitador usado no arquivo

`skip`: número de linhas de header a serem ignoradas

`cols`: tupla com as colunas que devem ser carregadas

`tipos`: tupla com os tipos das colunas

# Exemplo

```
import numpy as np
```

```
data = np.genfromtxt('ArquivoCNJ.csv', delimiter=';',  
skip_header = 1, usecols = (13,24), dtype=(np.intc, np.intc))
```

```
print(data.shape)
```

```
print(data.sum(0))
```

# Salvando

Para salvar um ndarray em um CSV, uma possibilidade é o uso da função `np.savetxt(nomeArquivo, array, fmt=specf, delimiter=delim)`

- `specf`: especificador de formato dos dados.

  - Por padrão é float

  - Especificadores similares a linguagem C

  - [numpy.org/devdocs/reference/generated/numpy.savetxt.html](http://numpy.org/devdocs/reference/generated/numpy.savetxt.html)

- `delimiter`: delimitador do CSV



# Exemplo

```
import numpy as np
```

```
dados = np.ones((3,4), dtype=np.intc)  
np.savetxt("dados.csv", dados, delimiter=";", fmt="%d")
```

# Parâmetros

Tanto `genfromtxt` quanto `savetxt` possuem diversos parâmetros extras que podem te ajudar em tarefas específicas.

Veja na documentação

Por exemplo, se você deseja adicionar um *header*, utilize o parâmetro de mesmo nome

Dica: Os *headers* e *footers* são inseridos por padrão como comentários

Para alterar isso, coloque `comments=""`

# Exemplo

```
import numpy as np
```

```
dados = np.ones((3,4), dtype=np.intc)
header = "Coluna0;Coluna1;Coluna2;Coluna3"
np.savetxt("dados.csv", dados, delimiter=";",
fmt="%d", header=header, comments="")
```

6.

# Teste seus conhecimentos



## Teste seus conhecimentos

1. Replique tudo que foi ensinado durante as aulas no seu computador para fixar os conhecimentos.
2. Considere o arquivo csv disponibilizado. O arquivo representa uma matriz onde cada linha possui 10 colunas, mas não sabemos quantas linhas exatamente a matriz possui (no arquivo os itens estão em uma linha só). Utilize seus conhecimentos de NumPy para carregar o CSV, e processar a matriz mostrando a soma dos itens de cada linha, e a soma total dos itens da matriz.



# Obrigado!

Bons Estudos!!!