01 - NumPy

Docupedia Export

Author:Lima Queila (CtP/ETS)
Date:21-Jan-2026 13:53

Table of Contents

1 Bibliotecas Python

Uma Biblioteca é uma coleção de módulos de script acessíveis a um programa Python para simplificar o processo de programação e remover a necessidade de reescrever os comandos mais usados.
Eles podem ser usados chamando-os / importando-os no início de um script.

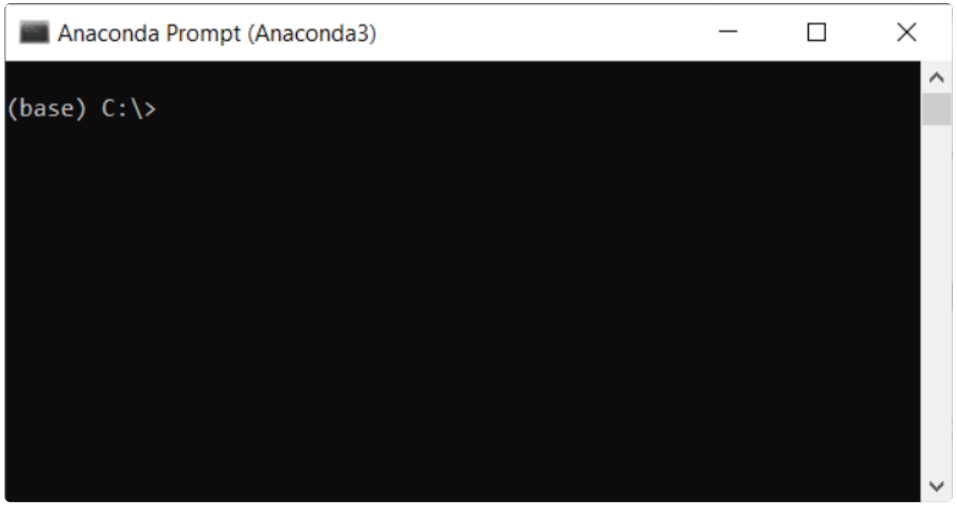
Exemplo de biblioteca já utilizada:

- Time
- Random

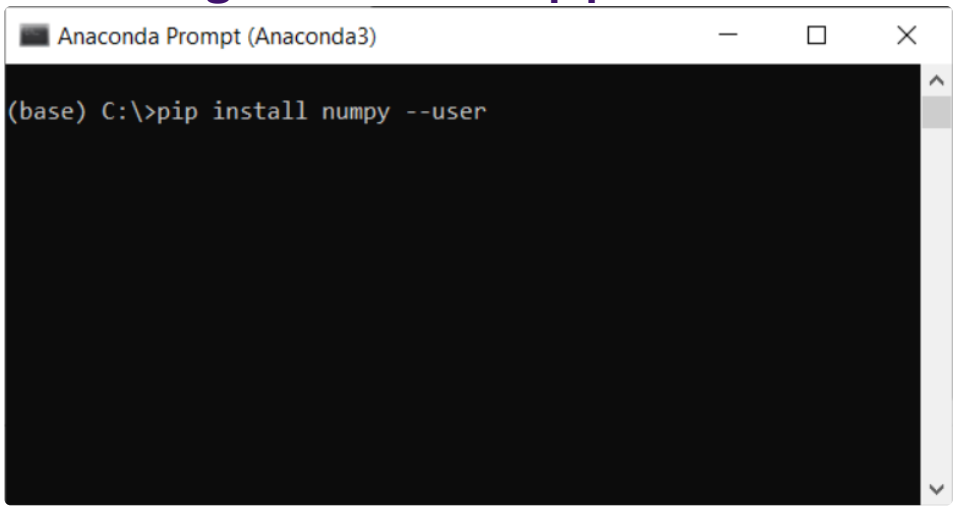
Existem bibliotecas padrões que já vem instaladas junto ao Python e bibliotecas que precisam ser instaladas.

1.1 Para instalar, siga os seguintes passos:

1.1.1 1º Abra o Anaconda Prompt



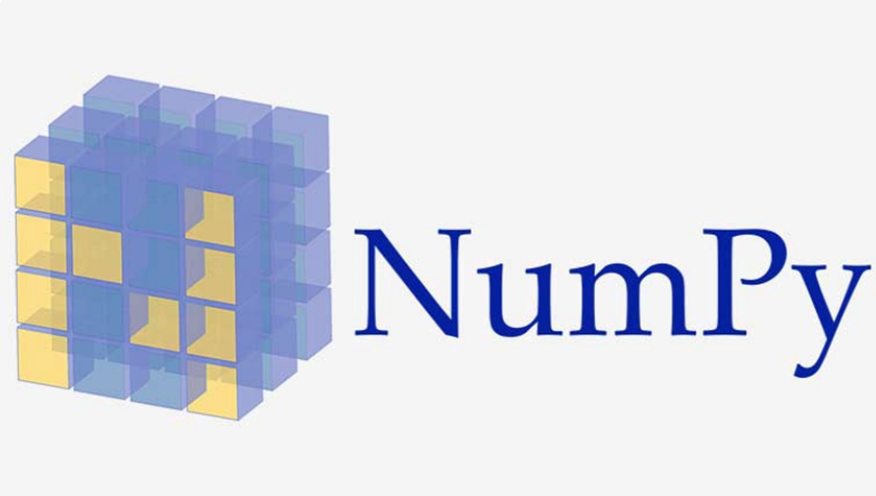
1.1.2 2º Digite o comando "pip install" + o nome da biblioteca + --user



1.1.3 3º Depois de instalada, é só utilizar dentro de nosso programa!!!

1.2 Então vamos começar a estudar as bibliotecas!

2 TREINAMENTO DE PYTHON (NUMPY)



O **NumPy** é uma poderosa biblioteca Python que é usada principalmente para realizar cálculos em Arrays Multidimensionais.
O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos.

2.1 Objetivos do NumPy

- **Modelos de Machine Learning:** Ao escrever algoritmos de Machine Learning, supõe-se que se realize vários cálculos numéricos em Array. Por exemplo, multiplicação de Arrays, transposição, adição, etc. O NumPy fornece uma excelente biblioteca para cálculos fáceis (em termos de escrita de código) e rápidos (em termos de velocidade).

- **Processamento de Imagem e Computação Gráfica:** Imagens no computador são representadas como Arrays Multidimensionais de números. NumPy torna-se a escolha mais natural para o mesmo. O NumPy, na verdade, fornece algumas excelentes funções de biblioteca para rápida manipulação de imagens. Alguns exemplos são o espelhamento de uma imagem, a rotação de uma imagem por um determinado ângulo etc.

- **Tarefas matemáticas:** NumPy é bastante útil para executar várias tarefas matemáticas como integração numérica, diferenciação, interpolação, extrapolação e muitas outras. O NumPy possui também funções incorporadas para álgebra linear e geração de números aleatórios.

```
import numpy as np # importamos a biblioteca, damos o apelido "np" para facilitar a escrita no decorrer do programa
```

```
lista = [1,2,3]
print(lista)
print(type(lista))
listaP=["Ve", 1]
```

```
[1, 2, 3]
<class 'list'>
```

```
array = np.array(listaP)
print(array)
print(type(array))
```

```
['Ve' '1']
<class 'numpy.ndarray'>
```

```
matriz = [[1,2,3], [4,5,6], [7,8,9]] # lista de listas
print(matriz)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
mat_numpy = np.array(matriz) # matriz do numpy
print(mat_numpy)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

2.2 Tamanho de um array

```
print(len(lista))
print(len(matriz)) # numero de linhas
print(len(matriz[0])) # numero de colunas
```

3
3
3

Usando listas, nós não temos a garantia de que todas as linhas terão o mesmo número de elementos. Logo, usar listas para simular uma matriz não é a melhor opção.

```
print(array.shape)
print(mat_numpy.shape)
```

(2,)
(3, 3)

2.3 Número de dimensões

```
print(array.ndim) # vetor tem 1 dimensão
print(mat_numpy.ndim) # matriz tem 2 dimensões
```

1
2

2.4 Redimensionando arrays

```
my_array = np.array([1,2,3,4,5,6,7,8])
print(my_array)
print('\n',my_array.reshape((2,4)))
np.reshape(my_array, (2,4))
```

[1 2 3 4 5 6 7 8]

[[1 2 3 4]
[5 6 7 8]]

2.5 Desafio

Crie um array numPy com números de 0 a 9.

Resultado

```
arrayy=np.array([0,1,2,3,4,5,6,7,8,9])
print(arrayy)
```

[0 1 2 3 4 5 6 7 8 9]

Nós podemos usar np.arange para criar um array com valores uniformemente distribuídos dentro de um intervalo especificado.
Com o seguinte formato: np.arange(start, stop, step, dtype=None).

```
array=np.arange(20,30,2)
print(array)
```

[20 22 24 26 28]

```
array=np.arange(10).reshape((2,5))
print(array)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[0 1 2 3 4 5 6 7 8 9]
```

2.6 Desafio

Agora tente criar um array de 0 a 100, pulando de 10 em 10.
DICA: Utilizando o np.arange

Resultado

```
my_array = np.arange(0,110,10)
print(my_array)
```

```
[  0  10  20  30  40  50  60  70  80  90 100]
```

2.6.1 Criando arrays de valores únicos.

```
np.zeros((2,2)) # cria uma matriz só com zeros
```

```
array([[0., 0.],
       [0., 0.]])
```

```
np.ones((5,2)) # cria uma matriz só com valores um
```

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

```
np.full((5,2),6) # cria uma matriz com o valor passado
```

```
array([[6, 6],
       [6, 6],
       [6, 6],
       [6, 6],
       [6, 6]])
```

2.7 Desafio

Crie uma matriz booleana de 10x10 com somente "True".

Resultado

```
arrayBool=np.full((10,10), True)
print(arrayBool)
```

```
[[ True  True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True]
```

```
[ True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True  True  True  True  True]]
```

2.7.1 Valores igualmente espaçados

```
print(np.linspace(1,6,9)) #inicio - final / passo-1
np.linspace(2,3,5)
```

```
[1.    1.625 2.25  2.875 3.5   4.125 4.75  5.375 6.   ]
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
```

2.7.2 Valores aleatórios

```
np.random.rand(2,2) # os parâmetros determinam a dimensão da matriz
```

```
array([[0.57414859, 0.35237768],
       [0.98436401, 0.69488699]])
```

```
np.random.randint(0,10,10) # 10 números inteiros aleatórios entre 0 e 10
```

```
array([7, 7, 9, 4, 2, 0, 8, 7, 9, 6])
```

2.8 Dados estatísticos do array

```
array= np.random.randint(0,50,10)
```

```
print(array)
```

[34 26 4 32 11 42 27 9 21 16]

```
array.max() # maior valor do array
```

47

```
array.min() # menor valor do array
```

2

```
array.argmax() # índice do maior valor
```

9

```
array.argmin() # índice do menor valor
```

5

```
array.mean() # média dos valores do array
```

35.6

```
array.std() # desvio padrão
```

12.379014500355025

```
array.var() # variância
```

153.23999999999998

A mediana precisa ordenar os valores do array, por isso é uma função do numpy, não um método do array.

```
np.median(array) # mediana
```

39.0

2.9 Indexação de arrays

```
print(array)
```

[34 26 4 32 11 42 27 9 21 16]

Podemos fazer a indexação de arrays igual é feito com listas.

```
array[0]
```

34

```
array[:5] # do começo até o quinto item
```

array([34, 26, 4, 32, 11])

```
array[0:10:2] # do primeiro ao último de 2 em 2
```

array([34, 4, 11, 27, 21])

2.9.1 Com matrizes é um índice para a linha e um para a coluna

```
print(mat_numpy)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
mat_numpy[2,2]
```

9

```
mat_numpy[:2,:2] # as duas primeiras linhas, as duas primeiras colunas
```

array([[1, 2],
       [4, 5]])

2.9.2 Podemos indexar por condições:

```
mat_numpy[mat_numpy > 3]
```

array([4, 5, 6, 7, 8, 9])

2.10 Desafio

Identifique todos os números ímpares de um array 3x3 e os substitua por 0.

Resultado

```
my_array = np.array([1,9,4,6,3,6,3,8,2]).reshape((3,3))
print(my_array)

impar = (my_array%2 != 0)
my_array[impar] = 0
```

```
print(my_array)
```

```
[[1 9 4]
 [6 3 6]
 [3 8 2]]
[[0 0 4]
 [6 0 6]
 [0 8 2]]
```

2.11 Operações com arrays

```
a = np.random.randint(0,50,5)
b = np.random.randint(0,50,5)
print(a)
print(b)
```

```
[ 9 19 12 47 48]
[47 24 29 11 49]
```

```
print(a + 10) # faz a soma com todos os elementos do array
print(np.sum(a))
```

```
[19 29 22 57 58]
135
```

```
print(a + b)
print(np.add(a,b))
```

```
[56 43 41 58 97]
[56 43 41 58 97]
```

```
print(a - b)
print(np.subtract(a,b))
```

```
[-38  -5 -17  36  -1]
[-38  -5 -17  36  -1]
```

```
print(a * b) # multiplicação escalar
print(np.multiply(a,b))
```

```
[ 423  456  348  517 2352]
[ 423  456  348  517 2352]
```

```
a @ b # produto interno
```

```
4096
```

Observe que * é a multiplicação elemento a elemento, e não uma multiplicação matricial.
Usamos a função ".dot" para calcular o produto interno de vetores, multiplicar um vetor por uma matriz e multiplicar matrizes.

```
x=np.array([[1,2],[3,4]])
x2=np.array([[1,2],[3,4]])
y=np.array([5,6])
print(x)
print(y)
print(y.dot(x))
print(np.dot(y,x))
print()
print(np.dot(x,x2))
```

```
[[1 2]
 [3 4]]
```

```
[5 6]

[23 34]
[23 34]

[[ 7 10]
 [15 22]]
```

2.12 Desafio

Crie um array de 2 linhas e 3 colunas e faça as seguintes operações:

- Faça um reshape de 3 linhas e 2 colunas;
- Some a primeira linha por 5;
- Multiplique a segunda linha por 3;
- E divida a segunda coluna por 2.

Resultado

```
my_array = np.array([[2,7,3],[9,7,12]])
print(my_array)
my_array = my_array.reshape((3,2))
print('\n',my_array)
my_array[1,0] += 5
print('\n',my_array)
my_array[1] *= 3
print('\n',my_array)
my_array[:,1] = my_array[:,1] / 2
print('\n',my_array)
```

```
[[ 2  7  3]
 [ 9  7 12]]

 [[ 2  7]
```

```
[ 3  9]
[ 7 12]]

[[ 7 12]
[ 3  9]
[ 7 12]]

[[ 7 12]
[ 9 27]
[ 7 12]]

[[ 7  6]
[ 9 13]
[ 7  6]]
```

2.13 Operações de conjuntos

```
np.union1d(a,b) # união (junta os valores sem repetir os iguais)
```

```
array([ 9, 11, 12, 19, 24, 29, 47, 48, 49])
```

```
np.intersect1d(a,b) # intersecção (valores em comum)
```

```
array([47])
```

```
c=np.array([1,2,2,2,6,3,3])
np.unique(c) # valores únicos do array
```

array([1, 2, 3, 6])

```
np.setdiff1d(a,b) # diferença entre conjuntos (valores em A que não aparecem em B)
```

array([9, 12, 19, 48])

2.14 Problema de referência

Imagine que você quer criar um array a partir de um array já existente:

```
array1 = np.arange(1,11)
array2 = array1[:5]

print(array1)
print(array2)
```

[1 2 3 4 5 6 7 8 9 10]
[1 2 3 4 5]

Quando fazemos uma alteração no array2, o array1 também sofre essa alteração, pois os dois estão referenciados.

```
array2 *= 2

print(array1)
print(array2)
```

[2 4 6 8 10 6 7 8 9 10]
[2 4 6 8 10]

Como fazer para um ser independente do outro??
Criamos como uma **CÓPIA**!!!

```
array1 = np.arange(1,11)
array2 = array1[:5].copy()

print(array1)
print(array2)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[1 2 3 4 5]
```

Agora posso alterar array2, mantendo array1 inalterado.

```
array2 *= 2

print(array1)
print(array2)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[ 2  4  6  8 10]
```

```
x= array2.flatten()
print(x)
array2[0]=3
print(array2)
print(x)
```

```
[ 2  4  6  8 10]
[ 3  4  6  8 10]
[ 2  4  6  8 10]
```

```
y= array2.ravel() #Retorna uma view e não uma cópia
print(y)
array2[0]=4
print(array2)
```

```
print(y)
```

```
[ 4  4  6  8 10]
[ 4  4  6  8 10]
[ 4  4  6  8 10]
```

```
x=np.array([[1,2],[3,4]])
print(x.T)
```

```
[[1 3]
 [2 4]]
```

Broadcasting - é um mecanismo que permite que o numpy funcione com matrizes de formas diferentes ao executar operações aritméticas isto é, quando temos uma matriz menor e uma matriz maior, e queremos usar a matriz menor várias vezes para executar alguma operação na matriz maior. Suponha que queremos adicionar um vetor constante a cada linha de uma matriz.

```
x=np.array([[1,2,3],[4,5,6]])
v=np.array([1,0,1])
y=np.empty_like(x)

for i in range(2):
    y[i,:]=x[i,:]+v
print(y)
```

```
[[2 2 4]
 [5 5 7]]
```

```
a=np.floor(10*np.random.random((3,4))) #numeros aleatórios maior que 0.0xxx
print(a)
```

```
[[4. 8. 4. 6.]
 [6. 5. 7. 8.]
 [3. 6. 8. 2.]]
```

```
t=True
f=False
print(np.where(a>2,t,f))
```

```
[[ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True False]]
```

```
s=np.array([0,1,2,3,4,5,6,7,8,9])
print(s[2:5])
print(s[:4])
print(s[6:])
print(s[:])
```

```
[2 3 4]
[0 1 2 3]
[6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

```
s=np.array([[0,1,2,3,4],
            [5,6,7,8,9],
            [10,11,12,13,14]])
print(s[:3,2:])
print(s[2:,:])
print(s[:,4:])
print(s[::2,::3])
```

```
[[ 2  3  4]
 [ 7  8  9]
 [12 13 14]]
[[10 11 12 13 14]]
```

```
[[ 4]
 [ 9]
 [14]]
[[ 0  3]
 [10 13]]
```

3 Desafio

3.1 Tabuleiro da Diagonal

Crie uma função que recebe um número inteiro `n` e retorna um array `n x n` do `NumPy`, com as seguintes regras:

- Os elementos da diagonal principal devem valer `1`
- Os elementos da diagonal secundária devem valer `2`
- Os demais elementos devem valer `0`
- Quando um elemento pertence às duas diagonais (isso só ocorre se `n` for ímpar), ele deve valer `3`