@Odemir Depieri Jr

Um guia sobre o Numpy Python

```
[49] print('Ola Mundão!')

     Ola Mundão!
```

O que é o NumPy

Numpy

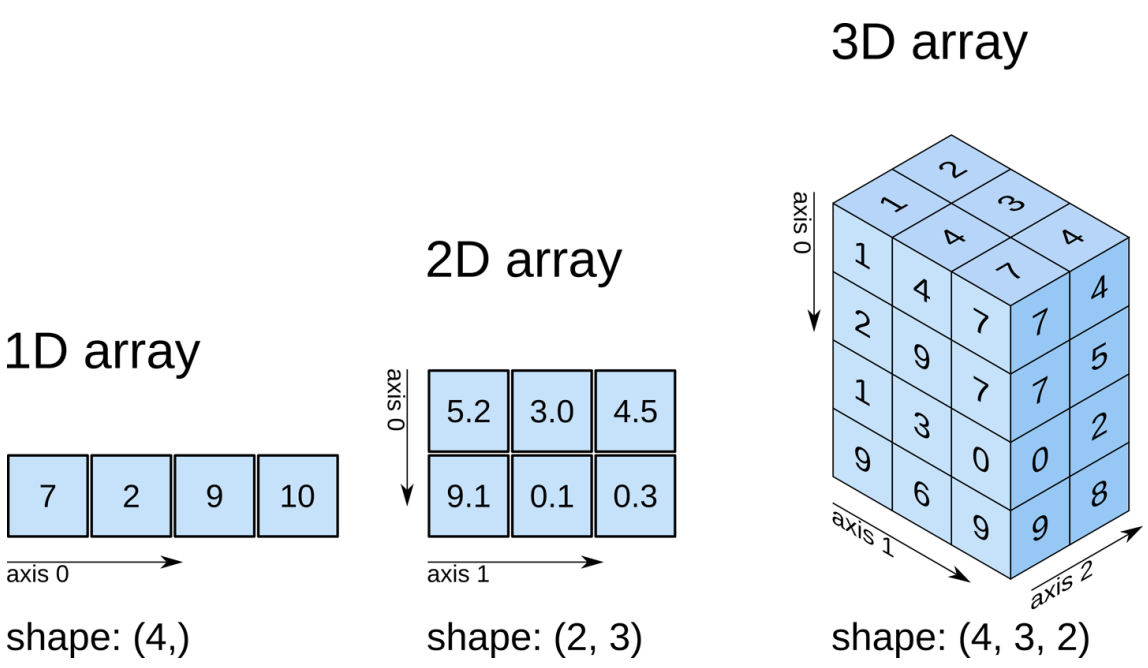
NumPy é uma biblioteca do Python usada para trabalhar com **matrizes**. Ele também tem funções para trabalhar no domínio da **álgebra linear**.

Essa biblioteca visa fornecer um objeto de matriz que **é até 50x mais rápido do que as listas Python tradicionais**.

O objeto array em NumPy é chamado ndarray, ele fornece várias funções de suporte que tornam o trabalho ndarraymuito fácil.

Os arrays são **usados com muita frequência em ciência de dados**, onde velocidade e recursos são muito importantes.

Ilustração das Matrizes geradas pelo Numpy



Testar a Potência do Numpy

Vamos estressar a performance do Numpy !!!!

```
[43] # Biblioteca para recursos matemáticos e matrizes
     import numpy as np
```

Vamos gerar números aleatórios com o Numpy.

```
[103]  # Gerando 1 milhão de registros
       Estrensando_Numpy_01 = np.arange(1000000)
      # Gerando 10 milhões de registros
       Estrensando_Numpy_02 = np.arange(10000000)
       # Gerando 50 milhões de registros
       Estrensando_Numpy_03 = np.arange(50000000)
       # Gerando 100 milhões de registros
       Estrensando_Numpy_04 = np.arange(100000000)
      # Verificando o tamanho dos Array
       print( 'Tamanho do Array:', len( Estrensando_Numpy_01 ) )
       print( 'Tamanho do Array:', len( Estrensando_Numpy_02 ) )
       print( 'Tamanho do Array:', len( Estrensando_Numpy_03 ) )
       print( 'Tamanho do Array:', len( Estrensando_Numpy_04 ) )
```

```
Tamanho do Array: 1000000
Tamanho do Array: 10000000
Tamanho do Array: 50000000
Tamanho do Array: 100000000
```

Vamos percorrer esses Arrays com milhões de registros e verificar em quanto tempo demorou essa tarefa.

```
# 1º Teste --- 1 Milhão de registros
%time for i in Estrensando_Numpy_01: pass
```

```
CPU times: user 123 ms, sys: 913 µs, total: 123 ms
Wall time: 127 ms
```

Rápido neh !

```
[99] # 2º Teste --- 10 Milhões de registros
     %time for i in Estrensando_Numpy_02: pass
```

```
CPU times: user 1.17 s, sys: 841 µs, total: 1.17 s
Wall time: 1.18 s
```

```
[100] # 3º Teste --- 50 Milhões de registros
      %time for i in Estrensando_Numpy_03: pass
```

```
CPU times: user 5.97 s, sys: 7.98 ms, total: 5.98 s
Wall time: 5.99 s
```

```
[101] # 4º Teste --- 100 Milhões de registros
      %time for i in Estrensando_Numpy_04: pass
```

```
CPU times: user 11.5 s, sys: 17.4 ms, total: 11.5 s
Wall time: 11.6 s
```

Mama mia !!! 😱

Mão na Massa

```
[43] # Biblioteca para recursos matemáticos e matrizes
     import numpy as np
```

```
[44] # Verificando a versão da Biblioteca
     print(np.__version__)
```

```
1.19.5
```

```
[45] # Criando uma matriz com 1 dimensão
     Array = np.array([10,20,30,40,50])
     Array
```

```
array([10, 20, 30, 40, 50])
```

```
[46] # Verificando o tipo do objetov
     print( type(Array) )
```

```
<class 'numpy.ndarray'>
```

```
[47] # Criando uma tupla usando matriz com 1 dimensão
     # Lembrando que não é possivel alterar uma tupla
     Tulpla_Array = np.array((10,20,30,40,50))
     Tulpla_Array
```

```
array([10, 20, 30, 40, 50])
```

```
[48] # Criar uma matriz com 0 zero dimensão
     Matriz_Zero = np.array(11)
     Matriz_Zero
```

```
array(11)
```

```
[49] # Criar uma matriz com 2 Dimensões
     Matriz_Duas_Dimensoes = np.array([
         [10, 9, 8, 7, 6],
         [5, 4, 3, 2, 1, 0]
     ])

     print( Matriz_Duas_Dimensoes )
```

```
[list([10, 9, 8, 7, 6]) list([5, 4, 3, 2, 1, 0])]
```

```
[50] # Criar uma matriz com 3 Dimensões
     Matriz_Tres_Dimensoes = np.array([
         [15, 14, 13, 12, 11],
         [10, 9, 8, 7, 6],
         [5, 4, 3, 2, 1, 0]
     ])

     print( Matriz_Tres_Dimensoes )
```

```
[list([15, 14, 13, 12, 11]) list([10, 9, 8, 7, 6])
 list([5, 4, 3, 2, 1, 0])]
```

```
[51] # Verificar a quantidade de dimensão em uma matriz

     print('Qtas dimesões no "Matriz_Zero":', Matriz_Zero.ndim, '\n' )
     print('Qtas dimesões no "Array":', Array.ndim, '\n' )
     print('Qtas dimesões no "Matriz_Duas_Dimensoes":',
           Matriz_Duas_Dimensoes.ndim, '\n' )
     print('Qtas dimesões no "Matriz_Tres_Dimensoes":',
           Matriz_Tres_Dimensoes.ndim, '\n' )
```

```
Qtas dimesões no "Matriz_Zero": 0

Qtas dimesões no "Array": 1

Qtas dimesões no "Matriz_Duas_Dimensoes": 1

Qtas dimesões no "Matriz_Tres_Dimensoes": 1
```

```
[52] # Podemos criar uma matriz e definir a quantidade de dimensões

     Definindo_Numeros_Dimensoes = np.array(
         [999, 998, 997, 996, 995 ],
         ndmin=10
     )

     print ( Definindo_Numeros_Dimensoes, '\n' )
     print ('Dimensões:', Definindo_Numeros_Dimensoes.ndim )
```

```
[[[[[[[[[[999 998 997 996 995]]]]]]]]]]

Dimensões: 10
```

```
[53] # Podemos acessar um valor atras da posição
     # Muito similar a uma lista do Python

     Array = np.array([10,20,30,40,50])

     print('Acessando posição 1:', Array[0] )
     print('Acessando ultima posição:', Array[-1] )
     print('Acessando penultimo posição:', Array[-2:-1] )
     print('Acessando um range posição:', Array[0:3] )
```

```
Acessando posição 1: 10
Acessando ultima posição: 50
Acessando penultimo posição: [40]
Acessando um range posição: [10 20 30]
```

```
[54] # Podemos fazer operações matematicas nas matrizes
     Soma = Array[0] + Array[1]
     Subtração = Array[0] - Array[1]
     Divisão = Array[0] / Array[1]
     Multiplicação = Array[0] * Array[1]
     Resto = Array[0] % Array[1]
     Expoente = Array[0] ** Array[1]

     # Podemos fazer condições
     Condição_Igual = Array[0] == Array[1]
     Condição_Maior = Array[0] > Array[1]
     Condição_Maior_Igual = Array[0] >= Array[1]
     Condição_Menor = Array[0] < Array[1]
     Condição_Menor_Igual = Array[0] <= Array[1]
     Diferente = Array[0] != Array[1]

     print('Operações Matematicas Resultados \o/')
     print('Soma:', Soma )
     print('Subtração:', Subtração )
     print('Divisão:', Divisão )
     print('Multiplicação:', Multiplicação )
     print('Resto:', Resto )
     print('Expoente:', Expoente )

     print('\n', 'Condições Resultados \o/')
     print('Condição_Igual:', Condição_Igual )
     print('Condição_Maior_Igual:', Condição_Maior_Igual )
     print('Condição_Menor:', Condição_Menor )
     print('Condição_Menor_Igual:', Condição_Menor_Igual )
     print('Diferente:', Diferente )
```

```
Operações Matematicas Resultados \o/
Soma: 30
Subtração: -10
Divisão: 0.5
Multiplicação: 200
Resto: 10
Expoente: 7766279631452241920

 Condições Resultados \o/
Condição_Igual: False
Condição_Maior_Igual: False
Condição_Menor: True
Condição_Menor_Igual: True
Diferente: True
```

```
[55] # Acessando uma matriz com 2 dimensões dimensões
     Matriz_Dois_Dimensoes = np.array([
         [15, 14, 13, 12, 11],
         [10, 9, 8, 7, 6]
     ])

     print('Acessando a 2º Dimensão, Posição numero 2')
     print(Matriz_Dois_Dimensoes[1, 1] )
```

```
Acessando a 2º Dimensão, Posição numero 2
9
```

```
[56] # Tipos de formatos de campo

     '''
     i - inteiro
     b - boleano
     u - inteiro sem sinal
     f - flutuar
     c - flutuação complexa
     m - timedelta
     M - data hora
     O - objeto
     S - fragmento
     U - string Unicode
     V - pedaço fixo de memória para outro tipo (vazio)
     '''

     # Verificando os formatos
     Inteiro = np.array( [10,20,30,40,50] )
     Textos = np.array( ['Uva', 'Pera', 'Limão'] )
     Fragmento = np.array( [10,20,30,40,50], dtype='S' )
     Flutuante = np.array( [10,20,30,40,50], dtype='f' )

     print( Inteiro.dtype )
     print( Textos.dtype )
     print( Fragmento.dtype )
     print( Flutuante.dtype )
```

```
int64
<U5
|S2
float32
```

```
[57] # Podemos converter o formato da matriz para outro

     Inteiro = np.array( [10,20,30,40,50] )
     Converter_Quebrado = Inteiro.astype('f')
     Converter_Booleano = Inteiro.astype('bool')

     print( Inteiro.dtype )
     print( Converter_Quebrado.dtype )
     print( Converter_Booleano.dtype )
```

```
int64
float32
bool
```

```
[58] # Podemos verificar o tamanho da Matriz
     Matriz_Tres_Dimensoes.shape
```

```
(3,)
```

```
[59] # Podemos remodelar uma Matriz

     Matriz_Uma_Dimensão = np.array([1, 2, 3, 4, 5, 6, 7,
                                     8, 9, 10, 11, 12])

     # Comando reshape faz isso de forma bem simples
     # 1ºArgumento é a Dimensão e o seguinte Posições
     Remodelando_01 = Matriz_Uma_Dimensão.reshape(4,3)
     Remodelando_02 = Matriz_Uma_Dimensão.reshape(2,6)
     print( '1º Matriz:', '\n', Remodelando_01, '\n' )
     print( '2º Matriz:', '\n', Remodelando_02 )
```

```
1º Matriz:
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

2º Matriz:
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
```

[60]
```
# Percorrendo uma matriz de 1 dimensão
Matriz_Uma_Dimensão = np.array([1, 2, 3, 4, 5, 6, 7,
                                8, 9, 10, 11, 12])

for Percorrendo in Matriz_Uma_Dimensão:
  print( Percorrendo )
```

```
1
2
3
4
5
6
7
8
9
10
11
12
```

[61]
```
# Percorrendo uma matriz de 2 dimensão
Matriz_Duas_Dimensoes = np.array([
    [10, 9, 8, 7, 6],
    [5, 4, 3, 2, 1, 0]
])

for Percorrendo in Matriz_Duas_Dimensoes:
  for Posicao in Percorrendo:
    print( Posicao )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: Vis
  after removing the cwd from sys.path.
10
9
8
7
6
5
4
3
2
1
0
```

```
[62] # Percorrendo uma matriz e enumerar a posição da matriz
     Matriz_Uma_Dimensão = np.array([1, 2, 3, 4, 5, 6, 7,
                                     8, 9, 10, 11, 12])

     for Posicao, Percorrendo in np.ndenumerate(Matriz_Uma_Dimensão):
       print( Posicao, Percorrendo )
```

```
(0,) 1
(1,) 2
(2,) 3
(3,) 4
(4,) 5
(5,) 6
(6,) 7
(7,) 8
(8,) 9
(9,) 10
(10,) 11
(11,) 12
```

```
[63] # Vamos juntar 2 matrizes em uma unica
     # Juntar matrizez usando o comando 'concatenate'
     Array_01 = np.array([1, 2, 3])
     Array_02 = np.array([4, 5, 6])

     # Exemplo 01
     Jutando_01 = np.concatenate((Array_02, Array_01))
     # Exemplo 02
     Jutando_02 = np.concatenate((Array_01, Array_02, Array_01))
     # Exemplo 03
     Jutando_03 = np.concatenate((Array_01 * 2,
                                  Array_02 * Array_01,
                                  Array_01 - Array_02))

     print( Jutando_01 , '\n')
     print( Jutando_02, '\n' )
     print( Jutando_03 )
```

```
[4 5 6 1 2 3]

[1 2 3 4 5 6 1 2 3]

[ 2  4  6  4 10 18 -3 -3 -3]
```

```
[64] # Quebrando/Fatiando uma matriz
     # Podemos escolhar em quantos pedassos podemos quebrar uma matriz

     Array = np.array([1, 2, 3, 4, 5, 6])
     Quebrando_Array_em3 = np.array_split( Array, 3)
     Quebrando_Array_em4 = np.array_split( Array, 4)
     Quebrando_Array_em2 = np.array_split( Array, 2)

     print( Quebrando_Array_em3, '\n' )
     print( Quebrando_Array_em4, '\n' )
     print( Quebrando_Array_em2 )
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]

[array([1, 2]), array([3, 4]), array([5]), array([6])]

[array([1, 2, 3]), array([4, 5, 6])]
```

```
[65] # Podemos usar outro metodos para quebrar 'hsplit'
     # Esse metodo divide uma matriz em várias submatrizes horizontalmente
     Matriz = np.array([[1, 2, 3],
                        [4, 5, 6],
                        [7, 8, 9],
                        [10, 11, 12],
                        [13, 14, 15],
                        [16, 17, 18]])

     Quebrando = np.hsplit(Matriz, 3)

     print( Quebrando )
```

```
[array([[ 1],
        [ 4],
        [ 7],
        [10],
        [13],
        [16]]), array([[ 2],
        [ 5],
        [ 8],
        [11],
        [14],
        [17]]), array([[ 3],
        [ 6],
        [ 9],
        [12],
        [15],
        [18]])]
```

```
[66] # Procurando valores em uma Matriz
     Array = np.array([1, 2, 3, 4, 5, 6])

     # Exemplo 1
     Procurando_01 = np.where( Array == 2 )
     # Exemplo 2
     Procurando_02 = np.where( Array > 2 )
     # Exemplo 3
     Procurando_03 = np.where( Array == True )

     print( Procurando_01 , '\n' )
     print( Procurando_02 , '\n' )
     print( Procurando_03 )
```

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

(array([2, 3, 4, 5]),)

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

```
[67] # Ordenando uma matriz em ordem

     Array = np.array([ 4, 10, 6, 11, 1, 2])
     Matriz_Duas_Dimensoes = np.array([
         [10, 9, 8, 7, 6],
         [5, 4, 3, 2, 1, 0]
     ])

     Ordenando_01 = np.sort( Array )
     Ordenando_02 = np.sort( Matriz_Duas_Dimensoes )

     print( Ordenando_01, '\n' )
     print( Ordenando_02 )
```

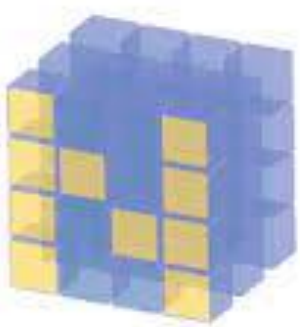
```
[ 1  2  4  6 10 11]

[list([5, 4, 3, 2, 1, 0]) list([10, 9, 8, 7, 6])]
```

Final

Esse guia sobre o numpy e suas principais funções mais usadas no meu conceito.

Link da documentação, caso queira mais detalhes.
https://numpy.org/doc/



NumPy

Odemir Depieri Jr

Software Engineer Sr
Tech Lead
Specialization AI