

class6

December 11, 2021

1 Introdução a Python

1.1 Aula 6

2 Sumário

- Instalação e setup
- Módulo NumPy
- Exercícios

3 Instalação e setup



4 Módulo NumPy

Para importar o módulo

```
[1]: import numpy as np
```

Para verificar a respetiva versão

```
[2]: print(np.__version__)
```

1.20.1

4.1 Criação de arrays

```
[3]: array0d = np.array(1) #poderia ser passado qualquer outro objeto semelhante a  
    → um array
```

```
print(array0d)
```

1

```
[4]: array1d = np.array([1,2,3])
```

```
print(array1d)
```

[1 2 3]

```
[5]: array2d = np.array([[1,2,3], [4,5,6]])
```

```
print(array2d)
```

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

```
[6]: array3d = np.array([[[1,2,3], [4,5,6], [7,8,9]]])
```

```
print(array3d)
```

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

```
[7]: array5d = np.array([1,2,3], ndmin=5)
```

```
print(array5d.ndim)
```

5

4.2 Indexação de arrays

```
[8]: pairs = np.array([2,4,6])
```

```
pairs[0]
```

```
[8]: 2
```

```
[9]: even_odd = np.array([[2,4,6], [1,3,5]])
```

```
even_odd[1, 1]
```

[9]: 3

```
[10]: powers_of_3 = np.array([1,3,9,27])  
      powers_of_3[-2]
```

[10]: 9

```
[11]: multiples_of_2_3 = np.array([[2,4,6],[3,6,9]])  
      multiples_of_2_3[1, -3]
```

[11]: 3

```
[12]: primes = np.array([2, 3, 5, 7])  
      primes[1:3]
```

[12]: array([3, 5])

```
[13]: divisors_of_6_8 = np.array([[1, 2, 3, 6], [1, 2, 4, 8]])  
      divisors_of_6_8[1, 1:4]
```

[13]: array([2, 4, 8])

```
[14]: powers_of_2 = np.array([1,2,4,8])  
      powers_of_2[-3:-1]
```

[14]: array([2, 4])

```
[15]: negative_positive = np.array([[-1,-2,-3,-4], [1,2,3,4]])  
      negative_positive[0, -4:-2]
```

[15]: array([-1, -2])

```
[16]: multiples_of_5 = np.array([5,10,15,20])  
      multiples_of_5[1:]
```

[16]: array([10, 15, 20])

```
[17]: int_float = np.array([[1,2,3,4],[1.1,2.2,3.3,4.4]])  
      int_float[1, 2:]
```

```
[17]: array([3.3, 4.4])
```

```
[18]: perfect_squares = np.array([1,4,9,16,25])  
perfect_squares[:2]
```

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

```
[19]: end_in_4_8 = np.array([[4,14,24,34],[8,18,28,38]])  
end_in_4_8[0, :3]
```

```
[19]: array([ 4, 14, 24])
```

```
[20]: decimal_places_1_2 = np.array([[1.1, 1.2, 1.3], [1.11, 1.22, 1.33]])  
decimal_places_1_2[0:2, 2]
```

```
[20]: array([1.3 , 1.33])
```

```
[21]: start_in_2_6 = np.array([[2,20,21,22], [6,60,61,62]])  
start_in_2_6[0:2, 1:3]
```

```
[21]: array([[20, 21],  
          [60, 61]])
```

4.3 Tipos de dados

S: string

i: integer

f: float

c: complex

b: boolean

```
[22]: array_of_ints = np.array([1,2,3,4])  
array_of_ints.dtype
```

```
[22]: dtype('int64')
```

```
[23]: array_of_strings = np.array([2,4,6,8], dtype = "S") #também poderíamos definir  
      ↪ o tamanho
```

```
array_of_strings.dtype
```

```
[23]: dtype('S1')
```

```
[24]: string_to_int = np.array(["1", "2", "3"])

new_string_to_int = string_to_int.astype("i") #também poderíamos colocar int

new_string_to_int.dtype
```

```
[24]: dtype('int32')
```

4.4 Copy vs View

```
[25]: repeated = np.array([2,2,2,2])

copy_repeated = repeated.copy()

repeated[0] = -2

print(repeated)
print(copy_repeated)
```

```
[-2  2  2  2]
[2  2  2  2]
```

```
[26]: numbers_1_to_5 = np.array([1,2,3,4,5])

view_numbers_1_to_5 = numbers_1_to_5.view()

numbers_1_to_5[0] = 0

print(numbers_1_to_5)
print(view_numbers_1_to_5)
```

```
[0 2 3 4 5]
[0 2 3 4 5]
```

```
[27]: ones = np.array([1,1,1,1,1])

copy_ones = ones.copy()
view_ones = ones.view()

print(copy_ones.base)
print(view_ones.base)
```

```
None
[1 1 1 1 1]
```

4.5 Forma e remodelação de arrays

```
[28]: ascending_descending = np.array([[1,2,3,4], [4,3,2,1]])

ascending_descending.shape
```

```
[28]: (2, 4)
```

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

matrix_4_3 = huge_array.reshape(4,3)

matrix_4_3
```

```
[29]: array([[ 1,  2,  3],
             [ 4,  5,  6],
             [ 7,  8,  9],
             [10, 11, 12]])
```

```
[30]: dim1 = np.array([11,12,13,14,15,16,17,18,19,20,21,22])

dim3 = dim1.reshape(2,3,2)

dim3
```

```
[30]: array([[[11, 12],
             [13, 14],
             [15, 16]],

            [[17, 18],
             [19, 20],
             [21, 22]]])
```

```
[31]: calculate_last_dimension = np.array([1,2,3,4,5,6,7,8])

new_calculate_last_dimension = calculate_last_dimension.reshape(2,2,-1)

new_calculate_last_dimension
```

```
[31]: array([[1, 2],
            [3, 4]],

           [[5, 6],
```

```
[7, 8]])
```

```
[32]: separate_arrays = np.array([[3,6,7], [1,3,7]])  
  
      joint_arrays = separate_arrays.reshape(-1)  
  
      joint_arrays
```

```
[32]: array([3, 6, 7, 1, 3, 7])
```

4.6 Iteração de arrays

```
[33]: high_dimensionality_array = np.array([[[[1,2],[3,4]], [[5,6], [7,8]]]])  
  
      for x in np.nditer(high_dimensionality_array):  
          print(x)
```

```
1  
2  
3  
4  
5  
6  
7  
8
```

```
[34]: array_of_chars = np.array([["a", "b", "c"], ["d", "e", "f"]])  
  
      for idx, x in np.ndenumerate(array_of_chars):  
          print(idx, x)
```

```
(0, 0) a  
(0, 1) b  
(0, 2) c  
(1, 0) d  
(1, 1) e  
(1, 2) f
```

4.7 Junção de arrays

```
[35]: first_array = np.array([1,2,3])  
  
      second_array = np.array([4,5,6])  
  
      resulting_array = np.concatenate((first_array, second_array))
```

```
resulting_array
```

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

```
[36]: array1 = np.array([[1,2], [3,4]])  
  
array2 = np.array([[5,6], [7,8]])  
  
final_array = np.concatenate((array1, array2), axis = 1) #axis toma o valor 0  
↳ por omissão  
  
final_array
```

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

```
[37]: top = np.array([6,7,8])  
bottom = np.array([9,10,11])  
  
top_and_bottom = np.stack((top, bottom))  
  
top_and_bottom
```

```
[37]: array([[ 6,  7,  8],  
            [ 9, 10, 11]])
```

```
[38]: top = np.array([6,7,8])  
bottom = np.array([9,10,11])  
  
top_and_bottom = np.vstack((top, bottom))  
  
top_and_bottom
```

```
[38]: array([[ 6,  7,  8],  
            [ 9, 10, 11]])
```

```
[39]: first = np.array([10,11,12])  
  
second = np.array([13,14,15])  
  
first_and_second = np.stack((first, second), axis=1)  
  
first_and_second
```

```
[39]: array([[10, 13],  
            [11, 14],
```



```
[12, 15]])
```

```
[40]: first = np.array([10,11,12])  
  
      second = np.array([13,14,15])  
  
      first_and_second = np.dstack((first, second))  
  
      first_and_second
```

```
[40]: array([[10, 13],  
            [11, 14],  
            [12, 15]])
```

```
[41]: begin = np.array([10,9,8])  
  
      end = np.array([7,6,5])  
  
      begin_and_end = np.hstack((begin, end))  
  
      begin_and_end
```

```
[41]: array([10,  9,  8,  7,  6,  5])
```

4.8 Separação de arrays

```
[42]: array_with_6_elems = np.array([1,2,3,4,5,6])  
  
      new_array_with_6_elems = np.array_split(array_with_6_elems,3)  
  
      new_array_with_6_elems
```

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

```
[43]: array_with_12_elems = np.array([[1,2], [3,4], [5,6], [7,8], [9,10], [11,12]])  
  
      new_array_with_12_elems = np.array_split(array_with_12_elems,3)  
  
      new_array_with_12_elems
```

```
[43]: [array([[1, 2],  
            [3, 4]]),  
      array([[5, 6],  
            [7, 8]]),  
      array([[ 9, 10],  
            [11, 12]])]
```

```
[44]: array_with_5_elems = np.array([1,2,3,4,5])
```

```
new_array_with_5_elems = np.array_split(array_with_5_elems, 4)
```

```
new_array_with_5_elems
```

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

```
[45]: array_with_10_elems = np.array([[1,2], [3,4], [5,6], [7,8], [9,10]])
```

```
new_array_with_10_elems = np.array_split(array_with_10_elems,4)
```

```
new_array_with_10_elems
```

```
[45]: [array([[1, 2],  
          [3, 4]]),  
       array([[5, 6]]),  
       array([[7, 8]]),  
       array([[ 9, 10]])]
```

```
[46]: lists_of_3_elems = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12], [13,14,15],  
    ↪ [16,17,18]])
```

```
new_3_arrays = np.array_split(lists_of_3_elems, 3, axis=1)
```

```
new_3_arrays
```

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

Nota: As funções `hsplit()`, `vsplit()` e `dsplit()` fazem o oposto das funções `hstack()`, `vstack()` e

dstack(), respetivamente.

4.9 Pesquisa em arrays

```
[47]: array_with_repeated_elems = np.array([1,2,3,3,4,5,3])

positions = np.where(array_with_repeated_elems == 3) #para além da condição
↪ pode receber mais dois
#argumentos: x (que vai ser executado caso a condição seja verdadeira) e y (que
↪ vai ser executado
#caso a condição seja falsa)

positions
```

```
[47]: (array([2, 3, 6]),)
```

```
[48]: initial_array = np.array([1,2,3,4])

position = np.searchsorted(initial_array, 2) #é suposto este método ser usado
↪ em arrays ordenados

position
```

```
[48]: 1
```

```
[49]: initial_array = np.array([1,2,3,4])

index = np.searchsorted(initial_array, 2, side = "right")

index
```

```
[49]: 2
```

```
[50]: starting_array = np.array([2,4,6,8])

indexes = np.searchsorted(starting_array, [1,3,5])

indexes
```

```
[50]: array([0, 1, 2])
```

4.10 Ordenação de arrays

```
[51]: unordered_array = np.array([9,1,6,3,2])  
  
np.sort(unordered_array) #também funciona para outros tipos de dados
```

```
[51]: array([1, 2, 3, 6, 9])
```

```
[52]: unordered_array_2d = np.array([[6,1,5], [5,9,1]])  
  
np.sort(unordered_array_2d)
```

```
[52]: array([[1, 5, 6],  
           [1, 5, 9]])
```

4.11 Filtragem de arrays

```
[53]: array_from_10_to_15 = np.array([10,11,12,13,14,15])  
  
booleans = [True, False, True, True, False, True]  
  
filtered_array = array_from_10_to_15[booleans]  
  
filtered_array
```

```
[53]: array([10, 12, 13, 15])
```

```
[54]: array_from_50_to_60 = np.array([50,51,52,53,54,55,56,57,58,59,60])  
  
higher_than_55 = [True if x > 55 else False for x in array_from_50_to_60]  
  
array_from_50_to_60[higher_than_55]
```

```
[54]: array([56, 57, 58, 59, 60])
```

```
[55]: array_from_50_to_60 = np.array([50,51,52,53,54,55,56,57,58,59,60])  
  
array_from_50_to_60[array_from_50_to_60 > 55]
```

```
[55]: array([56, 57, 58, 59, 60])
```

4.12 Outras funções relevantes

- `arange()`

```
[56]: array_with_arange = np.arange(5, 10)

array_with_arange
```

```
[56]: array([5, 6, 7, 8, 9])
```

- isnan()

```
[57]: array_with_nan_values = np.array([1, 2, np.nan, 4])

np.isnan(array_with_nan_values)
```

```
[57]: array([False, False,  True, False])
```

- mean()

```
[58]: heights = np.array([157,164,175,159,180])

np.mean(heights) #também podemos especificar o eixo
```

```
[58]: 167.0
```

- diag()

```
[59]: matrix_4_4 = np.arange(16).reshape(4,4)

np.diag(matrix_4_4)
```

```
[59]: array([ 0,  5, 10, 15])
```

- cumsum()

```
[60]: dozens = np.array([10,20,30])

np.cumsum(dozens)
```

```
[60]: array([10, 30, 60])
```

- prod()

```
[61]: hundreds = np.array([100,200,300])

np.prod(hundreds)
```

```
[61]: 6000000
```

5 Exercícios

- 1) Escreva uma função que recebe uma matriz 5x5 e retorna a média da segunda coluna.
- 2) Escreva uma função que recebe uma matriz 3x3 e retorna apenas os seus elementos pares.
- 3) Escreva uma função que recebe um número indeterminado de arrays como argumento e retorna aquele que resulta na maior média.
- 4) Escreva uma função que substitui todos os valores de um array (passado como argumento) que sejam inferiores a um número (também passado como argumento) por -100.
- 5) Escreva uma função que recebe um array e um valor como argumento, separa esse array segundo esse argumento e devolve o número de elementos em cada partição.
- 6) Escreva uma função que recebe dois arrays como argumento, podendo ser ambos views ou ambos cópias. Caso se verifique a primeira opção retorna a junção horizontal dos mesmos e caso se verifique a segunda opção retorna a junção vertical dos mesmos.
- 7) Escreva uma função que recebe dois arrays como argumento e calcula a soma de todos os elementos com todos os elementos.
- 8) Escreva uma função que devolve uma matriz 3x3 com valores desde 2 até 10.
- 9) Escreva uma função que recebe um array. Para os arrays de inteiros retorna a sua soma cumulativa e para os arrays de floats retorna o seu produto.
- 10) Escreva uma função que recebe uma matriz 4x4 e retorna a terceira coluna.
- 11) Escreva uma função que recebe um array com valores em falta como argumento e retorna as posições dos mesmos.
- 12) Escreva uma função que recebe um array como argumento cujos valores correspondem a graus centígrados e retorna a sua conversão para graus Fahrenheit.
- 13) Escreva uma função que retorna o índice da n.^a repetição de um item num array (ambos passados como argumento).
- 14) Escreva uma função que cria uma matriz 5x5 com os valores inteiros de 1 a 5 na diagonal.
- 15) Escreva uma função que retorna as posições dos elementos em x onde o seu valor é superior ao seu valor correspondente em y, sendo que x e y são arrays passados como argumento.