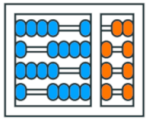


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 - Aula 27 (Complementar)

A biblioteca `numpy` de Python

Algoritmos e Programação de Computadores

Turmas

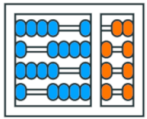
OVXZ

Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Terça-feira, 05 de julho de 2022

21:00h - 23:00h (CB06)



Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Algoritmos e Programação de Computadores

Turmas

OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

Conteúdo

- Numpy
  - O que é **NumPy**?
  - Começando com **NumPy**
  - Criando um *ndarray*
  - *zeros, ones, empty, full*
  - *arange* e *linspace*
  - Operações básicas
  - Multiplicação elemento a elemento e matricial
  - Operações in-place e `random`
  - Upcasting
  - Operações unarias
  - Operações nos eixos da matriz
  - Funçoes de uso geral
  - Indexação, fatiamento(slicing) e iteração
  - *Arrays* multidimensionais
  - Os pontos (...)
  - Iterando no primeiro eixo
  - Mudando a forma de um *array*
  - Empilhando diferentes *arrays*
  - `concatenate`
  - `column_stack, row_stack` (1-D)
  - `column_stack, row_stack` (2-D)
  - `column_stack, row_stack` (3-D)
  - `newaxis`
  - `split` horizontal (por colunas)
  - Copiando?
  - `view` (Cópia falsa)
  - `Slicing` retorna uma cópia falsa
  - `copy`, Cópia profunda
  - Indexando con uma matriz de indices
  - Indices para mais de uma dimensão
  - Buscando maximos usando indexação
  - Indexação com matrices booleanas
- Matplotlib
  - Graficando o conjunto de Mandelbrot
  - Graficando um Histograma

Numpy

Numpy

O que é NumPy?

https://numpy.org/doc/stable/user/whatisnumpy.html

NumPy é um conjunto de ferramentas para **computação científica** em Python.

NumPy é uma biblioteca Python que fornece **um objeto** *array* **multidimensional**:

ndarray,

vários objetos derivados e uma variedade de rotinas para operações rápidas em *arrays*, incluindo matemática, lógica, manipulação de formas, classificação, seleção, E/S, transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais.

https://numpy.org/doc/stable/user/whatisnumpy.html

No núcleo do pacote **NumPy**, está o objeto *ndarray*.

O objeto *ndarray* encapsula matrizes n-dimensionais de tipos de dados homogêneos, com muitas operações implementadas em código compilado para melhor desempenho.

Existem várias diferenças importantes entre as matrizes **NumPy** e os tipos padrão do Python.

https://numpy.org/doc/stable/user/whatisnumpy.html

Os objetos *ndarray* têm um tamanho fixo na criação, ao contrário das listas Python (que podem crescer dinamicamente). Alterar o tamanho de um *ndarray* criará um novo *array* e excluirá o original.

Todos os elementos em um objeto *ndarray* devem ser do mesmo tipo de dados e, portanto, terão o mesmo tamanho na memória. A exceção: pode-se ter *arrays* de objetos (Python, incluindo **NumPy**), permitindo assim *arrays* de elementos de tamanhos diferentes.

https://numpy.org/doc/stable/user/whatisnumpy.html

Os objetos *ndarray* facilitam operações matemáticas avançadas e outros tipos de operações em um grande número de dados. Normalmente, essas operações são executadas com mais eficiência e com menos código do que é possível usar com as sequências internas de Python.

Uma infinidade crescente de pacotes científicos e matemáticos baseados em Python estão usando matrizes **NumPy**; embora eles normalmente suportem entrada de sequência Python, eles convertem essa entrada em matrizes **NumPy** antes do processamento e geralmente produzem objetos *ndarray* ou derivados, melhorando a eficiência nos cálculos.

https://numpy.org/doc/stable/user/whatisnumpy.html

Tamanho e velocidade são muito importantes na computação científica. Por exemplo: Na primeira linha, o caso de multiplicar cada elemento de uma sequência a (1-D) com o elemento correspondente em outra sequência b do mesmo tamanho. Na 2da. linha, matrices 2-D.

C	Python	Numpy

`for (i=0; i<rows; i++) {`
`  c[i] = a[i]*b[i];`
`}`
	`c = []`
	`for i in range(len(a)):`
	`  c.append(a[i]*b[i])`
		`c = a*b`

`for (i=0; i<rows; i++) {`
`  for (j=0; j<columns; j++) {`
`  c[i][j] = a[i][j]*b[i][j];`
`  }`
`}`
	`c=[]`
	`for i in range(len(a)):`
	`  c.append([])`
	`  for j in range(len(a[0])):`
	`  c[i].append(a[i][j]*b[i][j])`
		`c = a*b`

Numpy

Começando com NumPy

https://numpy.org/doc/stable/user/quickstart.html

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

```
[2] b = np.array([(1/2, 2.5, 1e1), (4, 5, 6)])
    b
```

```
    array([[ 0.5,  2.5, 10. ],
           [ 4. ,  5. ,  6. ]])
```

```
[3] [type(b), b.ndim, b.shape, b.size, b.itemsize, b.nbytes]
```

```
    [numpy.ndarray, 2, (2, 3), 6, 8, 48]
```

```
[4] [b.dtype, b.dtype.name]
```

```
    [dtype('float64'), 'float64']
```

```
[5] [b.data,  b.data.ndim, b.data.shape, b.data.obj.size, b.data.itemsize, b.data.nbytes]
```

```
    [<memory at 0x7fbd60451bb0>, 2, (2, 3), 6, 8, 48]
```

https://colab.research.google.com/

Numpy

Criando um *ndarray*

https://numpy.org/doc/stable/user/quickstart.html

```
[6] a = np.array([2, 3, 4])
    [a, a.dtype]
```

```
    [array([2, 3, 4]), dtype('int64')]
```

```
[7] b = np.array([1.2, 3.5, 5.1])
    [b, b.dtype]
```

```
    [array([1.2, 3.5, 5.1]), dtype('float64')]
```

```
[8] c = np.array([[1.1, 2.2], [3, 4]], dtype=complex)
    [c, c.dtype]
```

```
    [array([[1.1+0.j, 2.2+0.j],
            [3. +0.j, 4. +0.j]]), dtype('complex128')]
```

```
[9] d = np.arange(12).reshape(2, 6)
    d
```

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

https://colab.research.google.com/

Numpy

zeros, ones, empty, full

https://numpy.org/doc/stable/user/quickstart.html

```
[10] np.zeros((3, 4))

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

[11] np.ones((2, 3), dtype=np.int16)

     array([[1, 1, 1],
            [1, 1, 1]], dtype=int16)

[12] np.empty((2, 3))

     array([[ 0.5,  2.5, 10. ],
            [ 4. ,  5. ,  6. ]])

[13] X = np.full ((2, 5), 2)
     X

     array([[2, 2, 2, 2, 2],
            [2, 2, 2, 2, 2]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[14] np.zeros_like(X)

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

[15] 5 + np.zeros(X.shape, dtype=int)

     array([[5, 5, 5, 5, 5],
            [5, 5, 5, 5, 5]])

[16] np.arange(10)*10 + 1

     array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91])

[17] np.arange(10)*10 + range(10)

     array([ 0, 11, 22, 33, 44, 55, 66, 77, 88, 99])
```

https://colab.research.google.com/

Numpy

arange e *linspace*

https://numpy.org/doc/stable/user/quickstart.html

```
[18] np.arange( 0, 100, 7)
```

```
array([ 0,  7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98])
```

```
[19] x = np.linspace(1, 2, 10)
     x
```

```
array([1.        , 1.11111111, 1.22222222, 1.33333333, 1.44444444,
       1.55555556, 1.66666667, 1.77777778, 1.88888889, 2.        ])
```

```
[20] f = np.sin(x)
     f
```

```
array([0.84147098, 0.8961922 , 0.93986069, 0.9719379 , 0.99202822,
       0.99988386, 0.99540796, 0.9786557 , 0.94983371, 0.90929743])
```

https://colab.research.google.com/

Numpy

Operações básicas

https://numpy.org/doc/stable/user/quickstart.html

```
[21] a = np.array([20, 30, 40, 50])
     b = np.arange(4)
     b
```

```
     array([0, 1, 2, 3])
```

```
[22] c = a - b
     c
```

```
     array([20, 29, 38, 47])
```

```
[23] b**2 , 10 * np.sin(a), a < 35
```

```
     (array([0, 1, 4, 9]),
      array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854]),
      array([ True,  True, False, False]))
```

https://colab.research.google.com/

Numpy

Multiplicação elemento a elemento e matricial

https://numpy.org/doc/stable/user/quickstart.html

```
[24] A = np.array([[1, 1],
                   [0, 1]])
     B = np.array([[2, 0],
                   [3, 4]])
```

```
[25] A * B     # elementwise product
```

```
     array([[2, 0],
            [0, 4]])
```

```
[26] A @ B
```

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

```
[27] A.dot(B)   # different than B.dot(A)
```

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

https://colab.research.google.com/

Numpy

Operações in-place e `random`

https://numpy.org/doc/stable/user/quickstart.html

```
[28] rg = np.random.default_rng(1)  # create instance of default random number generator

[29] a = rg.random((2, 3)) * 10
     a

     array([[5.11821625, 9.50463696, 1.44159613],
            [9.48649447, 3.11831452, 4.23326449]])

[30] b = np.ones((2, 3), dtype=int)

[31] a *= 3
     a

     array([[15.35464874, 28.51391089,  4.32478838],
            [28.45948341,  9.35494356, 12.69979347]])

[32] a-=b
     a

     array([[14.35464874, 27.51391089,  3.32478838],
            [27.45948341,  8.35494356, 11.69979347]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[33] b += a.astype(int)  # b is not automatically converted to integer type
     b

     array([[15, 28,  4],
            [28,  9, 12]])
```

\

https://colab.research.google.com/

Numpy

Upcasting

https://numpy.org/doc/stable/user/quickstart.html

```
[34] a = np.ones(3, dtype=np.int32)
     a
```

```
     array([1, 1, 1], dtype=int32)
```

```
[35] # from numpy import pi
     b = np.linspace(0, np.pi, 3)
     (b, b.dtype.name)
```

```
     (array([0.        , 1.57079633, 3.14159265]), 'float64')
```

```
[36] c = a + b
     (c, c.dtype.name)
```

```
     (array([1.        , 2.57079633, 4.14159265]), 'float64')
```

```
[37] d = np.exp(c * 1j)
     (d, d.dtype.name)
```

```
     (array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
             -0.54030231-0.84147098j]), 'complex128')
```

https://colab.research.google.com/

Numpy

Operações unarias

https://numpy.org/doc/stable/user/quickstart.html

```
[38] a = rg.random((2, 3))
     a

     array([[0.82770259, 0.40919914, 0.54959369],
            [0.02755911, 0.75351311, 0.53814331]])

[39] a.sum()

     3.1057109529998157

[40] a.min()

     0.027559113243068367

[41] a.max()

     0.8277025938204418
```

https://colab.research.google.com/

Numpy

Operações nos eixos da matriz

https://numpy.org/doc/stable/user/quickstart.html

```
[42] b = np.arange(12).reshape(3, 4)
     b

     array([[ 0,  1,  2,  3],
            [ 4,  5,  6,  7],
            [ 8,  9, 10, 11]])

[43] b.sum(axis=0)     # sum of each column

     array([12, 15, 18, 21])

[44] b.min(axis=1)     # min of each row

     array([0, 4, 8])

[45] b.cumsum(axis=1)  # cumulative sum along each row

     array([[ 0,  1,  3,  6],
            [ 4,  9, 15, 22],
            [ 8, 17, 27, 38]])
```

https://colab.research.google.com/

Numpy

Funçoes de uso geral

https://numpy.org/doc/stable/user/quickstart.html

```
[46] B = np.arange(3)
     B

     array([0, 1, 2])

[47] np.exp(B)

     array([1.        , 2.71828183, 7.3890561 ])

[48] np.sqrt(B)

     array([0.        , 1.        , 1.41421356])

[49] C = np.array([2., -1., 4.])
     C

     array([ 2., -1.,  4.])

[50] np.add(B, C)

     array([2., 0., 6.])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

Outras funções

Ver tambem: all, any, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cov, cross, cumprod, cumsum, diff, dot, floor, inner, invert, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sort, std, sum, trace, transpose, var, vdot, vectorize, where

Numpy

Indexação, fatiamento(slicing) e iteração

https://numpy.org/doc/stable/user/quickstart.html

```
[51] a = np.arange(10)**3
     a

     array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])

[52] a[2]

     8

[53] a[2:5]

     array([ 8, 27, 64])

[54] a[:6:2] = 1000
     a

     array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[55] a[::-1]
```

```
array([ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
```

```
[56] for i in a:
       print(i**(1 / 3.))
```

```
9.999999999999998
1.0
9.999999999999998
3.0
9.999999999999998
4.999999999999999
5.999999999999999
6.999999999999999
7.999999999999999
8.999999999999998
```

https://colab.research.google.com/

Numpy

Arrays multidimensionais

https://numpy.org/doc/stable/user/quickstart.html

```
[57] def f(x, y):
       return 10 * x + y
```

```
[58] b = np.fromfunction(f, (5, 4), dtype=int)
     b
```

```
     array([[ 0,  1,  2,  3],
            [10, 11, 12, 13],
            [20, 21, 22, 23],
            [30, 31, 32, 33],
            [40, 41, 42, 43]])
```

```
[59] b[2, 3]
```

```
     23
```

```
[60] b[0:5, 1]
```

```
     array([ 1, 11, 21, 31, 41])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[61] b[:, 1]

     array([ 1, 11, 21, 31, 41])

[62] b[0:, 1]

     array([ 1, 11, 21, 31, 41])

[63] b[1:3, :]

     array([[10, 11, 12, 13],
            [20, 21, 22, 23]])

[64] b[-1]

     array([40, 41, 42, 43])
```

https://colab.research.google.com/

Numpy

Os pontos (...)

https://numpy.org/doc/stable/user/quickstart.html

```
[65] x = np.arange(32).reshape(2, 2, 2, 2, 2)
     x

     array([[[[[ 0,  1],
               [ 2,  3]],

              [[ 4,  5],
               [ 6,  7]]],


             [[[ 8,  9],
               [10, 11]],

              [[12, 13],
               [14, 15]]]],


            [[[[16, 17],
               [18, 19]],

              [[20, 21],
               [22, 23]]],

             [[[24, 25],
               [26, 27]],

              [[28, 29],
               [30, 31]]]]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[67] x[..., 1]    # is equivalent to x[:, :, :, 1]

     array([[[[ 1,  3],
              [ 5,  7]],

             [[ 9, 11],
              [13, 15]]],


            [[[17, 19],
              [21, 23]],

             [[25, 27],
              [29, 31]]]])
```

```
[68] x[0, ..., 1, :]  # is equivalent to x[0, :, :, 1, :]

     array([[[ 2,  3],
             [ 6,  7]],

            [[10, 11],
             [14, 15]]])
```

https://colab.research.google.com/

Numpy

Iterando no primeiro eixo

https://numpy.org/doc/stable/user/quickstart.html

```
[69] c = np.array([[[  0,  1,  2],  # a 3D array (two stacked 2D arrays)
                    [ 10, 12, 13]],
                   [[100, 101, 102],
                    [110, 112, 113]]])
```

```
[70] c.shape
```

```
(2, 2, 3)
```

```
[71] for row in c:
         print(row)
```

```
[[ 0  1  2]
 [10 12 13]]
[[100 101 102]
 [110 112 113]]
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[72] for element in c.flat:
         print(element)
```

```
0
1
2
10
12
13
100
101
102
110
112
113
```

https://colab.research.google.com/

Numpy

Mudando a forma de um *array*

https://numpy.org/doc/stable/user/quickstart.html

```
[73] rg = np.random.default_rng(1)  # create instance of default random number generator
     a = np.floor(10 * rg.random((3, 4)))
     a
```

```
array([[5., 9., 1., 9.],
       [3., 4., 8., 4.],
       [5., 0., 7., 5.]])
```

```
[74] a.shape
```

```
(3, 4)
```

```
[75] a.ravel()  # returns the array, flattened
```

```
array([5., 9., 1., 9., 3., 4., 8., 4., 5., 0., 7., 5.])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[76] a.reshape(6, 2)
```

```
array([[5., 9.],
       [1., 9.],
       [3., 4.],
       [8., 4.],
       [5., 0.],
       [7., 5.]])
```

```
[77] a.T  # returns the array, transposed
```

```
array([[5., 3., 5.],
       [9., 4., 0.],
       [1., 8., 7.],
       [9., 4., 5.]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[78] a.T.shape
```

```
(4, 3)
```

```
[79] a.shape
```

```
(3, 4)
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[80] a
```

```
array([[5., 9., 1., 9.],
       [3., 4., 8., 4.],
       [5., 0., 7., 5.]])
```

```
[81] a.resize((2, 6))
     a
```

```
array([[5., 9., 1., 9., 3., 4.],
       [8., 4., 5., 0., 7., 5.]])
```

```
[82] a.reshape(3, -1)
```

```
array([[5., 9., 1., 9.],
       [3., 4., 8., 4.],
       [5., 0., 7., 5.]])
```

https://colab.research.google.com/

Numpy

Empilhando diferentes *arrays*

https://numpy.org/doc/stable/user/quickstart.html

```
[83] a = np.floor(10 * rg.random((2, 3)))
     a

     array([[3., 7., 3.],
            [4., 1., 4.]])

[84] b = np.floor(10 * rg.random((2, 3)))
     b

     array([[2., 2., 7.],
            [2., 4., 9.]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[85] np.stack((a,b), axis = 0)  # axis: index of the new axis in the dimensions of the result.

     array([[[3., 7., 3.],
             [4., 1., 4.]],

            [[2., 2., 7.],
             [2., 4., 9.]]])
```

```
[86] np.stack((a,b))[0]

     array([[3., 7., 3.],
            [4., 1., 4.]])
```

```
[87] np.stack((a,b))[1]

     array([[2., 2., 7.],
            [2., 4., 9.]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[88] np.stack((a,b), axis=1)

     array([[[3., 7., 3.],
             [2., 2., 7.]],

            [[4., 1., 4.],
             [2., 4., 9.]]])

[89] np.stack((a,b), axis=1)[:,0]

     array([[3., 7., 3.],
            [4., 1., 4.]])

[90] np.stack((a,b), axis=1)[:,1]

     array([[2., 2., 7.],
            [2., 4., 9.]])
```

https://colab.research.google.com/

Numpy

concatenate

https://numpy.org/doc/stable/user/quickstart.html

```
[91] a = np.array([[1, 2], [3, 4]])
     b = np.array([[5, 6], [7, 8]])
```

```
[92] np.concatenate((a, b), axis=0)   # np.vstack((a, b))
```

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

```
[93] np.concatenate((a, b), axis=1)  # np.hstack((a, b))
```

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

```
[94] np.concatenate((a, b), axis=None)
```

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

https://colab.research.google.com/

Numpy

column_stack, row_stack (1-D)

https://numpy.org/doc/stable/user/quickstart.html

```
[95] a = np.array([4., 2.])
     b = np.array([3., 8.])

[96] np.column_stack((a, b))

     array([[4., 3.],
            [2., 8.]])

[97] np.row_stack((a, b))

     array([[4., 2.],
            [3., 8.]])
```

https://colab.research.google.com/

Numpy

column_stack, row_stack (2-D)

https://numpy.org/doc/stable/user/quickstart.html

```
[98] a = np.array([[1, 2],[3, 4]])
```

```
[99] b = np.array([[5, 6],[7, 8]])
```

```
[100] Y = np.column_stack((a, b))  # np.hstack((a,b))
      Y
```

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

```
[101] np.row_stack((a, b))         # np.vstack((a,b))
```

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

https://colab.research.google.com/

Numpy

column_stack, row_stack (3-D)

https://numpy.org/doc/stable/user/quickstart.html

```
[102] a = np.floor(10 * rg.random((2, 2, 2)))
      a

      array([[[9., 7.],
              [5., 2.]],

             [[1., 9.],
              [5., 1.]]])
```

```
[103] b = np.floor(10 * rg.random((2, 2, 2)))
      b

      array([[[6., 7.],
              [6., 9.]],

             [[0., 5.],
              [4., 0.]]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[104] np.column_stack((a, b))    # np.hstack((a,b))
```

```
array([[[9., 7.],
        [5., 2.],
        [6., 7.],
        [6., 9.]],

       [[1., 9.],
        [5., 1.],
        [0., 5.],
        [4., 0.]]])
```

```
[105] np.row_stack((a, b))     # np.vstack((a,b))
```

```
array([[[9., 7.],
        [5., 2.]],

       [[1., 9.],
        [5., 1.]],

       [[6., 7.],
        [6., 9.]],

       [[0., 5.],
        [4., 0.]]])
```

https://colab.research.google.com/

Numpy

newaxis

https://numpy.org/doc/stable/user/quickstart.html

```
[106] from numpy import newaxis
      a = np.array([4., 2.])
      b = np.array([3., 8.])
```

```
[107] a[:, newaxis]
```

```
      array([[4.],
             [2.]])
```

```
[108] a[:, newaxis, newaxis]
```

```
      array([[[4.]],

             [[2.]]])
```

```
[109] a[newaxis, :]
```

```
      array([[4., 2.]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[110] a[newaxis, :, newaxis, newaxis ]
      array([[[[4.]],
              [[2.]]]])

[111] a[newaxis, :, newaxis, newaxis ][0,:,0,0]
      array([4., 2.])

[112] a[newaxis, :, newaxis, newaxis ][:,0,0,0]
      array([4.])

[113] newaxis is None
      True
```

https://colab.research.google.com/

Numpy

split horizontal (por colunas)

https://numpy.org/doc/stable/user/quickstart.html

```
[114] rg = np.random.default_rng(1)
      a = rg.integers(low=15, high=30, size=(3, 10))
      a

      array([[22, 22, 26, 29, 15, 17, 27, 29, 18, 19],
             [28, 21, 19, 27, 18, 21, 24, 23, 16, 15],
             [27, 26, 27, 23, 27, 19, 21, 26, 16, 19]])

[115] (x,y) = np.hsplit(a, 2)

[116] x

      array([[22, 22, 26, 29, 15],
             [28, 21, 19, 27, 18],
             [27, 26, 27, 23, 27]])

[117] y

      array([[17, 27, 29, 18, 19],
             [21, 24, 23, 16, 15],
             [19, 21, 26, 16, 19]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[118] np.hsplit(a, [2,3] )  #  a[:2],  a[2:3],  a[3:]
```

```
    [array([[22, 22],
            [28, 21],
            [27, 26]]), array([[26],
            [19],
            [27]]), array([[29, 15, 17, 27, 29, 18, 19],
            [27, 18, 21, 24, 23, 16, 15],
            [23, 27, 19, 21, 26, 16, 19]])]
```

https://colab.research.google.com/

Numpy

Copiando?

https://numpy.org/doc/stable/user/quickstart.html

```
[119] a = rg.integers(low=20, high=50, size=(2, 5))

      array([[23, 33, 49, 24, 31],
             [32, 47, 26, 35, 27]])

[120] b = a

[121] b is a

      True

[122] def f(x):
        print(id(x))

[123] id(a)

      140520359268048

[124] f(a)

      140520359268048
```

https://colab.research.google.com/

Numpy
view (Cópia falsa)

https://numpy.org/doc/stable/user/quickstart.html

```
[125] a = np.arange(5,40,3)
      a

      array([ 5,  8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38])

[126] c = a.view()

[127] c.base is a

      True

[128] c.flags.owndata

      False
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[129] c = c.reshape((2, 6))
      c

      array([[ 5,  8, 11, 14, 17, 20],
             [23, 26, 29, 32, 35, 38]])

[130] a

      array([ 5,  8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38])

[131] c[1, 3] = 1000
      c

      array([[   5,    8,   11,   14,   17,   20],
             [  23,   26,   29, 1000,   35,   38]])

[132] a

      array([   5,    8,   11,   14,   17,   20,   23,   26,   29, 1000,   35,
              38])
```

https://colab.research.google.com/

Numpy

Slicing retorna uma cópia falsa

https://numpy.org/doc/stable/user/quickstart.html

```
[133] s = c[:, 1:3]
      s

      array([[ 8, 11],
             [26, 29]])

[134] s[:] = 5555
      s

      array([[5555, 5555],
             [5555, 5555]])

[135] a

      array([   5, 5555, 5555,   14,   17,   20,   23, 5555, 5555, 1000,   35,
              38])

[136] c

      array([[   5, 5555, 5555,   14,   17,   20],
             [  23, 5555, 5555, 1000,   35,   38]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[137] a = list(range(5))
      b = a[2:4]
      b[:] = [5000,6000]
      a,b
```

```
([0, 1, 2, 3, 4], [5000, 6000])
```

```
[138] a = np.arange(5)
      b = a[2:4]
      b[:] = [5000,6000]
      a,b
```

```
(array([   0,    1, 5000, 6000,    4]), array([5000, 6000]))
```

https://colab.research.google.com/

Numpy

`copy`, Cópia profunda

https://numpy.org/doc/stable/user/quickstart.html

```
[139] a = np.arange(5)
      b = a[2:4].copy()    # a new array object is created
      b[:] = [5000,6000]
      a,b

      (array([0, 1, 2, 3, 4]), array([5000, 6000]))

[140] del a  # the memory of ``a`` can be released.
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

Functions and Methods Overview

Array Creation:	arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, r_, zeros, zeros_like
Conversions:	ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat
Manipulations:	array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack
Questions:	all, any, nonzero, where
Ordering:	argmax, argmin, argsort, max, min, ptp, searchsorted, sort
Operations:	choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum
Basic Statistics:	cov, mean, std, var
Basic Linear Algebra:	cross, dot, outer, linalg.svd, vdot

Numpy

Indexando con uma matriz de indices

https://numpy.org/doc/stable/user/quickstart.html

```
[141] a = np.arange(10)**2
      a

      array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])

[142] idx = np.array([1, 1, 3, 8, 5])
      idx

      array([1, 1, 3, 8, 5])

[143] b = a[idx]
      b

      array([ 1,  1,  9, 64, 25])

[144] b.base is a

      False
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[145] jdx = np.array([[3, 4], [9, 7]])
      jdx

      array([[3, 4],
             [9, 7]])

[146] b = a[jdx]
      b

      array([[ 9, 16],
             [81, 49]])

[147] b.base is a

      False

[148] b[:] = 1000
      a

      array([   0,   1,   4,   9,  16,  25,  36,  49,  64,  81])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[149] palette = np.array([[0, 0, 0],         # black
                          [255, 0, 0],       # red
                          [0, 255, 0],       # green
                          [0, 0, 255],       # blue
                          [255, 255, 255]])  # white
```

```
[150] image = np.array([[0, 1, 2, 0],  # each value corresponds to a color in the palette
                        [0, 3, 4, 0]])
```

```
[151] palette[image]  # the (2, 4, 3) color image
```

```
array([[[  0,   0,   0],
        [255,   0,   0],
        [  0, 255,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0, 255],
        [255, 255, 255],
        [  0,   0,   0]]])
```

https://colab.research.google.com/

Numpy

Indices para mais de uma dimensão

https://numpy.org/doc/stable/user/quickstart.html

```
[152] a = np.arange(12).reshape(3, 4)
      a

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

```
[153] i = np.array([[0, 1],  # indices for the first dim of `a`
                    [1, 2]])
```

```
[154] j = np.array([[2, 1],  # indices for the second dim
                    [3, 3]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[155] a[i, j]  # i and j must have equal shape
```

```
array([[ 2,  5],
       [ 7, 11]])
```

```
[156] a[i, 2]
```

```
array([[ 2,  6],
       [ 6, 10]])
```

```
[157] b = a[i, :]
      b
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 4,  5,  6,  7],
        [ 8,  9, 10, 11]]])
```

```
[158] b[:,:,2]  # b[...,2]
```

```
array([[ 2,  6],
       [ 6, 10]])
```

https://colab.research.google.com/

Numpy

Buscando maximos usando indexação

https://numpy.org/doc/stable/user/quickstart.html

```
[159] time = np.linspace(-20, 145, 5)
      time

      array([-20.  ,  21.25,  62.5 , 103.75, 145.  ])
```

```
[160] data = np.array([np.sin(time), np.cos(time), np.exp(-1/time)])
      data

      array([[-0.91294525,  0.67513565, -0.32579556, -0.07736505,  0.46774516],
             [ 0.40808206, -0.7376936 ,  0.94544024, -0.99700283,  0.88386337],
             [ 1.0512711 ,  0.95403128,  0.98412732,  0.99040775,  0.99312717]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[161] ind = data.argmax(axis=1)
      ind

      array([1, 2, 0])

[162] time_max = time[ind]
      time_max

      array([ 21.25,  62.5 , -20.  ])

[163] data_max = data[ range(data.shape[0]) , ind ]
      data_max

      array([0.67513565, 0.94544024, 1.0512711 ])

[164] data.max(axis=1)

      array([0.67513565, 0.94544024, 1.0512711 ])

[165] np.all(data_max == data.max(axis=1))

      True
```

https://colab.research.google.com/

Numpy

Indexação com matrices booleanas

https://numpy.org/doc/stable/user/quickstart.html

```
[166] a = rg.integers(low=20, high=50, size=(2, 8))
      a

      array([[20, 42, 21, 28, 34, 34, 23, 49],
             [42, 48, 22, 41, 28, 36, 47, 28]])

[167] b = a > 30
      b

      array([[False,  True, False, False,  True,  True, False,  True],
             [ True,  True, False,  True, False,  True,  True, False]])

[168] a[b]

      array([42, 34, 34, 49, 42, 48, 41, 36, 47])

[169] a[b] = 0
      a

      array([[20,  0, 21, 28,  0,  0, 23,  0],
             [ 0,  0, 22,  0, 28,  0,  0, 28]])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
a = np.arange(12).reshape(3, 4)
```

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

```
[171] b1 = np.array([False, True, True])
      b2 = np.array([True, False, True, False])
```

```
[172] a[b1, :]     # a[b1]
```

```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
[173] a[:, b2]
```

```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
```

```
[174] a[b1, b2]
```

```
array([ 4, 10])
```

https://colab.research.google.com/

Matplotlib

Matplotlib

Graficando o conjunto de Mandelbrot

https://numpy.org/doc/stable/user/quickstart.html

```
[175] import numpy as np
      import matplotlib.pyplot as plt
```

```
[176] def mandelbrot(h, w, maxit=20, r=2):
          """Returns an image of the Mandelbrot fractal of size (h,w)."""
          x = np.linspace(-2.5, 1.5, 4*h+1)
          y = np.linspace(-1.5, 1.5, 3*w+1)
          A, B = np.meshgrid(x, y)
          C = A + B*1j
          z = np.zeros_like(C)
          divtime = maxit + np.zeros(z.shape, dtype=int)

          for i in range(maxit):
              z = z**2 + C
              diverge = abs(z) > r                    # who is diverging
              div_now = diverge & (divtime == maxit)  # who is diverging now
              divtime[div_now] = i                    # note when
              z[diverge] = r                          # avoid diverging too much

          return divtime
```

```
[177] plt.clf()
```

https://colab.research.google.com/

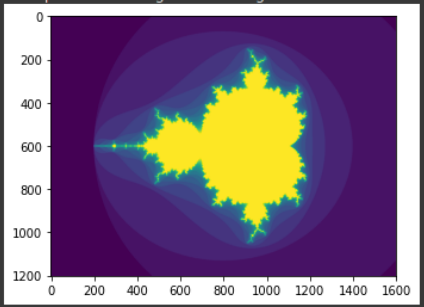
https://numpy.org/doc/stable/user/quickstart.html

```
[177] plt.clf()
```

```
<Figure size 432x288 with 0 Axes>
```

```
[178] plt.imshow(mandelbrot(400, 400))
```

```
<matplotlib.image.AxesImage at 0x7fd5d2065750>
```



https://colab.research.google.com/

Matplotlib

Graficando um Histograma

https://numpy.org/doc/stable/user/quickstart.html

```
[179] import numpy as np
      import matplotlib.pyplot as plt
      rg = np.random.default_rng(1)
```

```
[180] mu, sigma = 2, 0.5
      v = rg.normal(mu, sigma, 10000)
      v
```

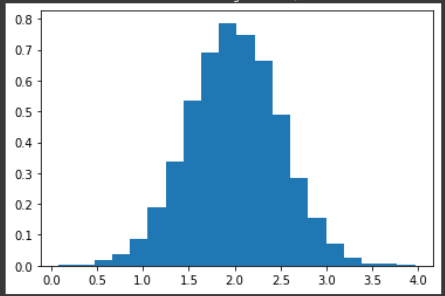
```
      array([2.1727921 , 2.41080907, 2.16521854, ..., 2.15062613, 1.61436007,
             2.09274213])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[181] plt.hist(v, bins=20, density=True)

(array([0.00257412, 0.00411859, 0.02007815, 0.03758218, 0.08546084,
        0.18945537, 0.33875443, 0.53644699, 0.69140912, 0.78716646,
        0.74649533, 0.66257896, 0.4901128 , 0.28572752, 0.15650661,
        0.07156059, 0.02677087, 0.00823719, 0.00566307, 0.00154447]),
 array([0.08106893, 0.27530993, 0.46955093, 0.66379193, 0.85803293,
        1.05227393, 1.24651493, 1.44075593, 1.63499693, 1.82923793,
        2.02347893, 2.21771993, 2.41196094, 2.60620194, 2.80044294,
        2.99468394, 3.18892494, 3.38316594, 3.57740694, 3.77164794,
        3.96588894]),
 <a list of 20 Patch objects>)
```



https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[182] (n, bins) = np.histogram(v, bins=20, density=True)
      n
```

```
array([0.00257412, 0.00411859, 0.02007815, 0.03758218, 0.08546084,
       0.18945537, 0.33875443, 0.53644699, 0.69140912, 0.78716646,
       0.74649533, 0.66257896, 0.4901128 , 0.28572752, 0.15650661,
       0.07156059, 0.02677087, 0.00823719, 0.00566307, 0.00154447])
```

```
[183] bins
```

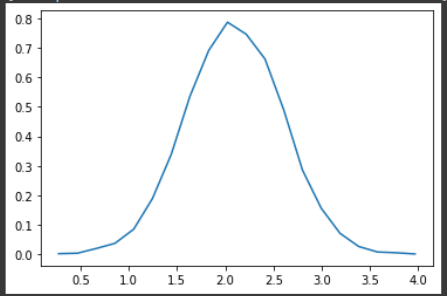
```
array([0.08106893, 0.27530993, 0.46955093, 0.66379193, 0.85803293,
       1.05227393, 1.24651493, 1.44075593, 1.63499693, 1.82923793,
       2.02347893, 2.21771993, 2.41196094, 2.60620194, 2.80044294,
       2.99468394, 3.18892494, 3.38316594, 3.57740694, 3.77164794,
       3.96588894])
```

https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[184] plt.plot((bins[1:]), n)
```

```
[<matplotlib.lines.Line2D at 0x7fd5d1a69f90>]
```

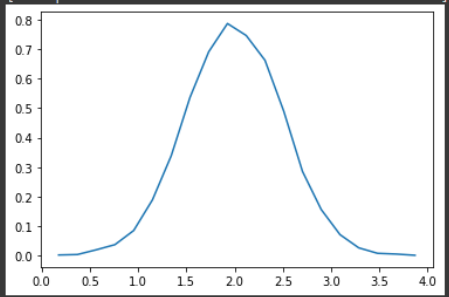


https://colab.research.google.com/

https://numpy.org/doc/stable/user/quickstart.html

```
[185] plt.plot(.5 * (bins[1:] + bins[:-1]), n)
```

```
[<matplotlib.lines.Line2D at 0x7fd5d19e3ed0>]
```



https://colab.research.google.com/

Perguntas ....

Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/

  Aula Introdutória [slides] [vídeo]
  Primeira Aula de Laboratório [slides] [vídeo]
  Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [slides] [vídeo]
  Comandos Condicionais [slides] [vídeo]
  Comandos de Repetição [slides] [vídeo]
  Listas e Tuplas [slides] [vídeo]
  Strings [slides] [vídeo]
  Dicionários [slides] [vídeo]
  Funções [slides] [vídeo]
  Objetos Multidimensionais [slides] [vídeo]
  Algoritmos de Ordenação [slides] [vídeo]
  Algoritmos de Busca [slides] [vídeo]
  Recursão [slides] [vídeo]
  Algoritmos de Ordenação Recursivos [slides] [vídeo]
  Arquivos [slides] [vídeo]
  Expressões Regulares [slides] [vídeo]
  Execução de Testes no Google Cloud Shell [slides] [vídeo]
  Numpy [slides] [vídeo]
  Pandas [slides] [vídeo]

- Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/

  Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/
  Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/

- Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/
- Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/
- Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/
- w3schools - Python Tutorial https://www.w3schools.com/python/
- Outros, citados nos Slides.