Основы **NumPy**

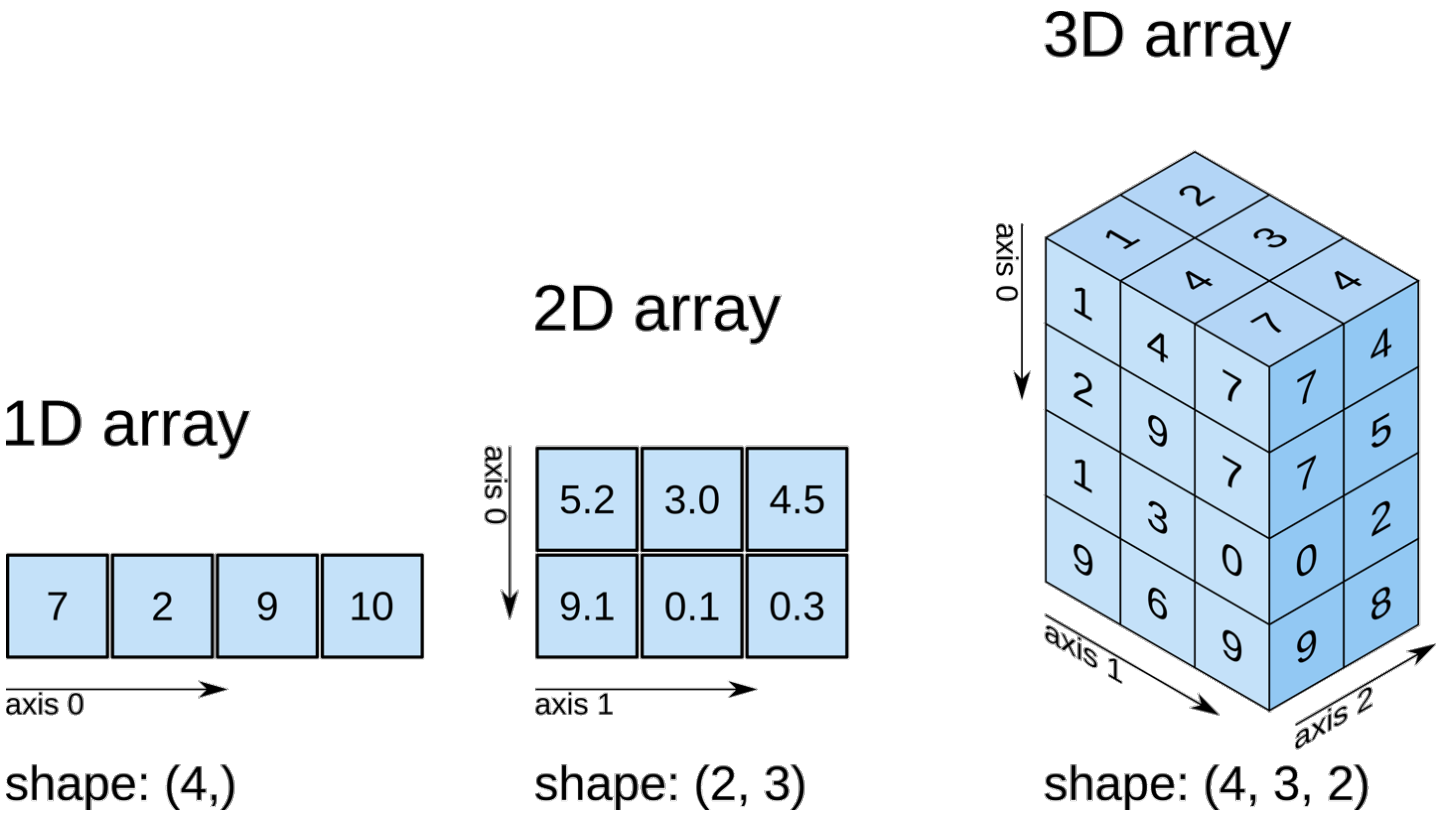
Основа NymPy - это объект **ndarray,** который используется для представления **N-мерного массива;** это быстрый и гибкий контейнер для хранения большим наборов данных в **Python.** Массивы позволяют выполнять математические перации над целыми блоками данных**.** По сути это обычный **list** с большим набором возможностей**.**

У любого **numpy-массива (array)** есть атрибут **shape -** кортеж**,** описывающий размер по каждому измерению**,** и атрибут **dtype -** объект**,** описывающий тип данных в массив**; ndim -** количество измерений массива **(**одномерный**,** двухмерный и т.д**.). tolist() -** конвертирует **array** в обычный **python list**

Array vs list:

- **Array** используют меньше памяти**,** чем списки
- **Array** обладают значительно большей функциональностью
- **Array** требуют**,** чтобы данные были однородными**;** списки нет

1D (одномерные массивы, списки, векторы), 2D (двумерные массивы, матрицы) и 3D (куб)-arrays:



Четырехмерный**,** пятимерный и более массив **—** это **тензор.**

Работа с **NumPy**

In [1]:

```
import numpy as np
```

In [2]:

```
data = [1, 2, 3, 4, 5]

arr = np.array(data) # создание из list

print(arr)
print(arr.shape)
print(arr.dtype)
print(arr.ndim)
```

```
[1 2 3 4 5]
(5,)
int64
1
```

Типы данных **NumPy:**

- **np.int64 // Signed 64-bit integer types**
- **np.float32 // Standard double-precision floating point**
- **np.complex // Complex numbers represented by 128 floats>**
- **np.bool // Boolean type storing TRUE and FALSE values**
- **np.object // Python object type**
- **np.string_ // Fixed-length string type>**
- **np.unicode_ // Fixed-length unicode type**

In [3]:

```
arr2 = np.array([1, 2, 3, 4, 5])

print(arr2)
print(arr2.shape)
print(arr2.dtype)
print(arr2.ndim)
```

```
[1 2 3 4 5]
(5,)
int64
1
```

In [4]:

```
arr3 = np.array([1, 2, 3, 4, 5], dtype=np.float) # задаём явно тип данных

print(arr3)
print(arr3.shape)
print(arr3.dtype)
print(arr3.ndim)
print(len(arr3))
print(arr3.size)
```

```
[1. 2. 3. 4. 5.]
(5,)
float64
1
5
5
```

In [5]:

```
arr3 = arr3.astype(np.int64) # приводим все элементы к типу int64

print(arr3)
print(arr3.dtype)
```

```
[1 2 3 4 5]
int64
```

In [6]:

```
arr4 = np.arange(0, 20, 1.5)
```

```
arr4
```

Out[6]:

```
array([ 0. ,  1.5,  3. ,  4.5,  6. ,  7.5,  9. , 10.5, 12. , 13.5, 15. ,
       16.5, 18. , 19.5])
```

In [7]:

```
arr5 = np.linspace(0, 2, 5) # 5 чисел от 0 до 2

arr5
```

Out[7]:

```
array([0. , 0.5, 1. , 1.5, 2. ])
```

In [8]:

```
arr5 = np.linspace(0, 2, 50)

arr5
```

Out[8]:

```
array([0.        , 0.04081633, 0.08163265, 0.12244898, 0.16326531,
       0.20408163, 0.24489796, 0.28571429, 0.32653061, 0.36734694,
       0.40816327, 0.44897959, 0.48979592, 0.53061224, 0.57142857,
       0.6122449 , 0.65306122, 0.69387755, 0.73469388, 0.7755102 ,
       0.81632653, 0.85714286, 0.89795918, 0.93877551, 0.97959184,
       1.02040816, 1.06122449, 1.10204082, 1.14285714, 1.18367347,
       1.2244898 , 1.26530612, 1.30612245, 1.34693878, 1.3877551 ,
       1.42857143, 1.46938776, 1.51020408, 1.55102041, 1.59183673,
       1.63265306, 1.67346939, 1.71428571, 1.75510204, 1.79591837,
       1.83673469, 1.87755102, 1.91836735, 1.95918367, 2.        ])
```

In [9]:

```
random_arr = np.random.random((5,))

random_arr
```

Out[9]:

```
array([0.71266268, 0.85998461, 0.39411119, 0.88311842, 0.99725676])
```

In [10]:

```
random_arr2 = np.random.random_sample((5,))
random_arr2
```

Out[10]:

```
array([0.52246906, 0.87445036, 0.83718441, 0.19932268, 0.98272956])
```

Диапазон случайных чисел от **[a,b]: (b - a) * np.random() + a,** где **b > a**

In [11]:

```
# 5 случайных значений от -5 до 10
random_arr3 = (10 - -5) * np.random.random_sample((5,)) - 5
random_arr3
```

Out[11]:

```
array([ 6.94092513,  6.81737375,  7.51712893, -4.14482843,  2.04890578])
```

Операции

In [12]:

```
arr = np.array([1, 2, 3, 4, 5])

arr = np.sqrt(arr)
print(arr)

arr = np.sin(arr)
print(arr)

arr = np.cos(arr)
print(arr)

arr = np.log(arr)
print(arr)

arr = np.exp(arr)
print(arr)
```

```
[1.         1.41421356 1.73205081 2.         2.23606798]
[0.84147098 0.98776595 0.98702664 0.90929743 0.78674913]
[0.66636675 0.55055622 0.55117323 0.61430028 0.70615086]
[-0.40591509 -0.59682621 -0.59570612 -0.48727141 -0.34792639]
[0.66636675 0.55055622 0.55117323 0.61430028 0.70615086]
```

In [13]:

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

print(a)
print(b)
```

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

In [14]:

```
c = a + b # np.add(a, b)

print(c)
```

```
[ 7  9 11 13 15]
```

In [15]:

```
a * b # np.multiply(a,b)
```

Out[15]:

```
array([ 6, 14, 24, 36, 50])
```

In [16]:

```
a - b # np.subtract
```

Out[16]:

```
array([-5, -5, -5, -5, -5])
```

In [17]:

```
a / b #np.divide
```

Out[17]:

```
array([0.16666667, 0.28571429, 0.375     , 0.44444444, 0.5       ])
```

In [18]:

```
arr = np.array([1, 2, 3, 4, 5])
arr = arr * 2  # все элементы умножить на 2
print(arr)
```

[2 4 6 8 10]

In [19]:

```
arr = arr ** 2  # возводим все элементы в квадрат
arr
```

Out[19]:

```
array([  4,  16,  36,  64, 100])
```

In [20]:

```
simple_list = [1, 2, 3, 4, 5]
simple_list = simple_list * 2
print(simple_list)

simple_list = simple_list ** 2
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-0b0f93be7e0b> in <module>
      3 print(simple_list)
      4
----> 5 simple_list = simple_list ** 2

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Функции агрегаты

In [21]:

```
arr = np.random.randint(-5, 10, 10)
arr
```

Out[21]:

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

In [22]:

```
print(arr.max())
print(arr.min())
print(arr.mean())
print(arr.sum())
print(arr.std())
print(np.median(arr))
```

```
6
-3
0.5
5
3.1064449134018135
0.0
```

In [23]:

```
arr < 2
```

Out[23]:

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

Манипуляции с массивами

In [24]:

```
np.insert(arr, 2, -20) # вставляем 20 в позицию 2
```

Out[24]:

```
array([  6,   1, -20,   2,  -3,   2,  -3,  -3,  -1,  -1,   5])
```

In [25]:

```
np.delete(arr, 2) # удаляем элемент во 2 позиции, с матрицами можно укзывать строки или столбцы
```

Out[25]:

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

In [26]:

```
np.sort(arr)
```

Out[26]:

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

In [27]:

```
arr2 = np.array([0, 0, 0])
arr = np.concatenate((arr, arr2))
arr
```

Out[27]:

```
array([ 6,  1,  2, -3,  2, -3, -3, -1, -1,  5,  0,  0,  0])
```

In [28]:

```
np.array_split(arr, 3) #hsplit, vsplit
```

Out[28]:

```
[array([ 6,  1,  2, -3,  2]), array([-3, -3, -1, -1]), array([5, 0, 0, 0])]
```

Индексы и одномерные массивы

In [29]:

```
arr = np.array([1, -2, 3, -4, 5])
arr[0] = 0
print(arr[2])
print(arr[0:2])
print(arr[::-1])
print(arr[arr < 2])
print(arr[(arr < 2) & (arr > 0)])
print(arr[(arr > 4) | (arr < 0)])
arr[1:4] = 0
print(arr)
```

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

Матрицы и многомерные массивы

In [30]:

```
matrix = np.array([(1,2,3),(4,5,6)], dtype=np.float64)
matrix
```

Out[30]:

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

In [31]:

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

Out[31]:

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

In [32]:

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

Out[32]:

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

In [33]:

```
a_3d_array = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
a_3d_array
```

Out[33]:

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

       [[5, 6],
        [7, 8]]])
```

In [34]:

```
matrix = np.array([(1,2,3),(4,5,6), (7, 8, 9)])
matrix
print(matrix.shape)
print(matrix.ndim)
print(matrix.size)
```

```
(3, 3)
2
9
```

In [35]:

```
matrix.reshape(1,9)
```

Out[35]:

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

In [36]:

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

Out[36]:

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

In [37]:

```
matrix.reshape(2,6)
```

Out[37]:

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

In [38]:

```
matrix2 = np.random.random((2,2))
matrix2
```

Out[38]:

```
array([[0.03505188, 0.48018074],
       [0.32415882, 0.81960652]])
```

In [39]:

```
print(matrix)
matrix = np.resize(matrix, (2,2))
print()
print(matrix)
```

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

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

In [40]:

```
new_matrix = np.arange(16).reshape(2,8)
new_matrix
```

Out[40]:

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

Создание специальных матриц

In [41]:

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

Out[41]:

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

In [42]:

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

Out[42]:

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

In [43]:

```
np.eye(5)
```

Out[43]:

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
```

```
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [44]:

```
np.full((3,3), 9)
```

Out[44]:

```
array([[9, 9, 9],
       [9, 9, 9],
       [9, 9, 9]])
```

In [45]:

```
np.empty((3,2))
```

Out[45]:

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

Операции над матрицами

In [46]:

```
matrix1 = np.array([(1, 2), (3, 4)])
print(matrix1)
print()
matrix2 = np.array([(5, 6), (7, 8)])
print(matrix2)
```

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

[[5 6]
 [7 8]]
```

In [47]:

```
matrix1 + matrix2 # np.add
```

Out[47]:

```
array([[ 6,  8],
       [10, 12]])
```

In [48]:

```
matrix1 - matrix2
```

Out[48]:

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

In [49]:

```
matrix1 * matrix2
```

Out[49]:

```
array([[ 5, 12],
       [21, 32]])
```

In [50]:

```
matrix1 / matrix2
```

Out[50]:

```
array([[0.2       , 0.33333333],
```

```
array([[0.2        , 0.33333333],
       [0.42857143, 0.5       ]])
```

In [51]:

```
matrix1.dot(matrix2) # скалаярное произведение
```

Out[51]:

```
array([[19, 22],
       [43, 50]])
```

Axis - ось**. 0 -** ось строк**, 1 -** ось столбцов

https://i.stack.imgur.com/gj5ue.jpg

In [52]:

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

Out[52]:

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

In [53]:

```
np.delete(matrix,1,axis=0) # удаляет 1 строку матрицы
```

Out[53]:

```
array([[1, 2, 3],
       [7, 8, 9]])
```

In [54]:

```
np.delete(matrix,1,axis=1) # удаляет 1 столбец матрицы
```

Out[54]:

```
array([[1, 3],
       [4, 6],
       [7, 9]])
```

In [55]:

```
np.sin(matrix)
```

Out[55]:

```
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ],
       [ 0.6569866 ,  0.98935825,  0.41211849]])
```

In [56]:

```
np.log(matrix)
```

Out[56]:

```
array([[0.        , 0.69314718, 1.09861229],
       [1.38629436, 1.60943791, 1.79175947],
       [1.94591015, 2.07944154, 2.19722458]])
```

При применении определенных функций **Numpy,** таких как **np.mean (),max()** и т.д. мы можем указать**,** по какой оси мы хотим вычислять значения**.**

Для **axis = 0** это означает**,** что мы применяем функцию к каждому «столбцу» или ко всем значениям**,** которые встречаются по вертикали**.**

Для **axis = 1** это означает**,** что мы применяем функцию к каждой «строке» или ко всем значениям по

горизонтали.

In [57]:

```
matrix.sum()
```

Out[57]:

```
45
```

In [58]:

```
matrix
```

Out[58]:

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

In [59]:

```
print(matrix.max())
print(matrix.max(axis=0))  # ищем наибольший элемент по каждой строке
```

```
9
[7 8 9]
```

In [60]:

```
np.mean(matrix, axis=0) # среднее значение идёт по строкам (1 + 4 + 7)/3 = 4
```

Out[60]:

```
array([4., 5., 6.])
```

In [61]:

```
np.mean(matrix, axis=1) # среднее значение идёт по столбцам (1 + 2 + 3)/3 = 2
```

Out[61]:

```
array([2., 5., 8.])
```

In [62]:

```
matrix**2
```

Out[62]:

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

In [63]:

```
arr1 = np.array([[1, 2, 3],[4, 5, 6]])
print(arr1)
print()
arr2 = np.array([[7, 8, 9],[10, 11, 12]])
print(arr2)
print()
arr = np.concatenate((arr1,arr2),axis=1) # добавляем arr2 к arr1 как столбцы
arr
```

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

[[ 7  8  9]
 [10 11 12]]
```

Out[63]:

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

In [64]:

```
arr = np.concatenate((arr1,arr2),axis=0) # добавляем arr2 к arr1 как строки
arr
```

Out[64]:

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

In [65]:

```
np.hstack((arr1,arr2))
```

Out[65]:

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

In [66]:

```
arr = np.vstack((arr1,arr2))
arr
```

Out[66]:

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

In [67]:

```
np.split(arr,2)
```

Out[67]:

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

In [68]:

```
np.split(arr,2, axis=0)
```

Out[68]:

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

In [69]:

```
np.split(arr,3, axis=1)
```

Out[69]:

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

In [70]:

```
np.append(arr, np.array([1, 2, 3]))
```

Out[70]:

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

Индексация

In [71]:

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

Out[71]:

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

In [72]:

```
matrix[1,2] # 2 элемент 1 строки
```

Out[72]:

```
6
```

In [73]:

```
matrix[2] # 2 строка
```

Out[73]:

```
array([7, 8, 9])
```

In [74]:

```
matrix[:,2] # 2 столбец
```

Out[74]:

```
array([3, 6, 9])
```

In [75]:

```
matrix[1:3, 0:2] # 0 и 1 столбец, 1  и 2 строки
```

Out[75]:

```
array([[4, 5],
       [7, 8]])
```

In [76]:

```
matrix[matrix > 2]
```

Out[76]:

```
array([3, 4, 5, 6, 7, 8, 9])
```

Специальные операции

In [77]:

```
matrix
```

Out[77]:

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

```
       [7, 8, 9]])
```

In [78]:

```
matrix.T  # транспонированная
```

Out[78]:

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

In [79]:

```
matrix.flatten() # в массив
```

Out[79]:

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

In [80]:

```
matrix[0] = 9
```

In [81]:

```
np.linalg.inv(matrix) # обратная матрица
```

Out[81]:

```
array([[ 3.75299969e+14,  1.12589991e+15, -1.12589991e+15],
       [-7.50599938e+14, -2.25179981e+15,  2.25179981e+15],
       [ 3.75299969e+14,  1.12589991e+15, -1.12589991e+15]])
```

In [82]:

```
np.trace(matrix) # след матрицы
```

Out[82]:

```
23
```

In [83]:

```
np.linalg.det(matrix) # определитель
```

Out[83]:

```
-7.99360577730114e-15
```

In [84]:

```
np.linalg.matrix_rank(matrix) # ранк
```

Out[84]:

```
2
```

In [85]:

```
eigenvalues, eigenvectors = np.linalg.eig(matrix) # собственные числа и векторы
print(eigenvalues)
print()
print(eigenvectors)
```

```
[ 2.19043260e+01  1.09567398e+00 -1.64282867e-15]

[[ 0.69279812  0.83297925  0.40824829]
 [ 0.38120547 -0.5046154  -0.81649658]
 [ 0.61213818 -0.22695566  0.40824829]]
```

Доп возможности

In [86]:

```
np.info(np.eye)
```

```
 eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

Return a 2-D array with ones on the diagonal and zeros elsewhere.

Parameters
----------
N : int
  Number of rows in the output.
M : int, optional
  Number of columns in the output. If None, defaults to `N`.
k : int, optional
  Index of the diagonal: 0 (the default) refers to the main diagonal,
  a positive value refers to an upper diagonal, and a negative value
  to a lower diagonal.
dtype : data-type, optional
  Data-type of the returned array.
order : {'C', 'F'}, optional
    Whether the output should be stored in row-major (C-style) or
    column-major (Fortran-style) order in memory.

    .. versionadded:: 1.14.0

Returns
-------
I : ndarray of shape (N,M)
  An array where all elements are equal to zero, except for the `k`-th
  diagonal, whose values are equal to one.

See Also
--------
identity : (almost) equivalent function
diag : diagonal 2-D array from a 1-D array specified by the user.

Examples
--------
>>> np.eye(2, dtype=int)
array([[1, 0],
       [0, 1]])
>>> np.eye(3, k=1)
array([[0.,  1.,  0.],
       [0.,  0.,  1.],
       [0.,  0.,  0.]])
```

In [ ]:

```
np.loadtxt('file.txt') # загрузка из текстовог офайла
np.genfromtxt('file.csv',delimiter=',') # загрузка из csv офайла
np.savetxt('file.txt',arr,delimiter=' ') # запись в текстовый файл
np.savetxt('file.csv',arr,delimiter=',') # запись в csv файл
```