Numpy

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.18.4'
```

Типы данных

In [3]:

```
# bool_, int_, intc, intp, int8, int16, int32, int64, uint8,
# uint16, uint32, uint64, float_, float16, float32, float64, complex_,
# complex64, complex128
```

Массивы

In [4]:

```
# создание массива из списка Python
a = np.array([1,2,3,4,5])
print(a)
```

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

In [5]:

```
# dtype - тип массива
a = np.array([1,2,3,4,5], dtype=np.float32)
print(a)
```

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

In [6]:

```
# многомерный массив
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
```

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

In [7]:

```
# массив целых чисел длины 10, заполненный нулями
z = np.zeros(10, dtype=int)
print(z)
```

```
[0 0 0 0 0 0 0 0 0 0]
```

In [8]:

```
# массив единиц с типом float размером 3x5
o = np.ones((3,5), dtype=float)
print(o)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

In [9]:

```
# массив размером 3x5, заполненный значением 2.71
f = np.full((3,5), 2.71)
print(f)
```

```
[[2.71 2.71 2.71 2.71 2.71]
 [2.71 2.71 2.71 2.71 2.71]
 [2.71 2.71 2.71 2.71 2.71]]
```

In [10]:

```
# массив значений от 0 до 20 с шагом 2
r = np.arange(0, 20, 2)
print(r)
```

```
[ 0  2  4  6  8 10 12 14 16 18]
```

In [11]:

```
# массив из 5 значений равномерно располагающихся между 0 и 2
l = np.linspace(0, 2, 5)
print(l)
```

```
[0.  0.5 1.  1.5 2. ]
```

In [12]:

```
# массив размером 3x3 равномерно распределенных случайных величин
# от 0 до 1
r = np.random.random((3,3))
print(r)
```

```
[[0.06853093 0.4234799  0.85072438]
 [0.37414171 0.82066428 0.27979438]
 [0.82374183 0.26240387 0.55227524]]
```

In [13]:

```
# массив размером 3x3 нормально распределенных случайных величин
# от 0 до 1, с медианой 0 и стандартным отклонением 1
n = np.random.normal(0, 1, (3,3))
print(n)
```

```
[[-0.37889358  1.16831645  0.33492513]
 [ 0.2895384  -0.76578003  0.2848451 ]
 [-0.26656892 -0.23287554 -1.73487397]]
```

In [14]:

```
# массив размером 3x3 случайных целых чисел от 0 до 10
r = np.random.randint(0, 10, (3,3))
print(r)
```

```
[[1 2 6]
 [1 3 5]
 [2 2 0]]
```

In [15]:

```
# единичная матрица 3x3
e = np.eye(3)
print(e)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [16]:

```
# неинициализированный массив произвольных целочисленных значений
e = np.empty(3)
print(e)
```

```
[1. 1. 1.]
```

Атрибуты массива

In [17]:

```
x = np.random.randint(10, size=(2,3,4))
```

In [18]:

```
# размерность
print(x.ndim)
```

```
3
```

In [19]:

```
# размер каждого измерения
print(x.shape)
```

```
(2, 3, 4)
```

In [20]:

```
# общий размер массива
print(x.size)
```

```
24
```

In [21]:

```
# тип данных
print(x.dtype)
```

```
int32
```

In [22]:

```
# размер каждого элемента в байтах
print(x.itemsize)
```

```
4
```

In [23]:

```
# полный размер массива в байтах
print(x.nbytes)
```

```
96
```

Индексация массива

In [24]:

```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a[1,1])
# также стандартно
print(a[1][1])
```

```
5
5
```

In [25]:

```
print(a[0, ::-1])
```

```
[3 2 1]
```

Копирование подмассивов

In [26]:

```
# при изменении подмассива изменяется сам головной массив
# при изменении подмассива с необходимостью оставить неизменным
# головной массив нужен метод копирования
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
a2 = a[:2, :2].copy()
a2[0,0] = 100
print(a)
print(a2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[100   2]
 [  4   5]]
```

Методы массива

In [27]:

```
# изменение формы
# размер исходного массива должен соответствовать размеру измененного
a = np.arange(1,10)
print(a)
print(a.reshape((3,3)))
```

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

In [28]:

```
# объединение
m = np.array([[1, 2, 3], [4, 5, 6]])
# по первой оси
print(np.concatenate([m, m]))
# по второй оси
print(np.concatenate([m, m], axis=1))
```

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

In [29]:

```
# объединение по вертикали
x = np.array([1, 2, 3])
y = np.array([[4, 5, 6], [7, 8, 9]])
print(np.vstack([x, y]))
```

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

In [30]:

```
# объединение по горизонтали
x = np.array([[1], [2]])
y = np.array([[4, 5, 6], [7, 8, 9]])
print(np.hstack([x, y]))
```

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

In [31]:

```
# разбиение
# необходимо передать индексы разбиения
# (n точек разбиения - n+1 подмассив)
x = np.arange(1,10)
x1,x2,x3 = np.split(x, [3, 6])
print(x1,x2,x3)
```

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

In [32]:

```
# разбиение по вертикали
x = np.arange(9).reshape((3,3))
x1,x2 = np.vsplit(x,[2])
print(x1,x2)
```

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

In [33]:

```
# разбиение по горизонтали
x = np.arange(9).reshape((3,3))
x1,x2 = np.hsplit(x,[2])
print(x1,x2)
```

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

Математические функции над массивами

In [34]:

```
# арифметические функции
x = np.arange(1, 6)
print(x)
print(-x)    # np.negative(x)
print(x+3)   # np.add(x, 3)
print(x-3)   # np.subtract(x, 3)
print(x*3)   # np.multiply(x, 3)
print(x/3)   # np.divide(x, 3)
print(x//3)  # np.floor_divide(x, 3)
print(x%3)   # np.mod(x, 3)
print(x**3)  # np.power(x, 3)
```

```
[1 2 3 4 5]
[-1 -2 -3 -4 -5]
[4 5 6 7 8]
[-2 -1  0  1  2]
[ 3  6  9 12 15]
[0.33333333 0.66666667 1.         1.33333333 1.66666667]
[0 0 1 1 1]
[1 2 0 1 2]
[  1   8  27  64 125]
```

In [35]:

```
# абсолютное значение
print(np.abs(-x))
```

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

In [36]:

```
# тригонометрические функции
print(np.sin(x))
print(np.cos(x))
print(np.tan(x))
print(np.arcsin(x//3))
print(np.arccos(x//3))
print(np.arctan(x//3))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
[ 1.55740772 -2.18503986 -0.14254654  1.15782128 -3.38051501]
[0.         0.         1.57079633 1.57079633 1.57079633]
[1.57079633 1.57079633 0.         0.         0.        ]
[0.         0.         0.78539816 0.78539816 0.78539816]
```

In [37]:

```
# показательные функции и логарифмы
print(np.exp(x))      # e**x
print(np.exp2(x))     # 2**x
print(np.power(3, x)) # 3**x
print(np.log(x))      # натуральный логарифм
print(np.log2(x))     # логарифм по основанию 2
print(np.log10(x))    # логарифм по основанию 10
```

```
[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
[ 2.  4.  8. 16. 32.]
[  3   9  27  81 243]
[0.         0.69314718 1.09861229 1.38629436 1.60943791]
[0.         1.         1.5849625  2.         2.32192809]
[0.         0.30103    0.47712125 0.60205999 0.69897   ]
```

In [38]:

```
# функции агрегирования
print(np.sum(x))             # аналог x.sum()
print(np.max(x))             # аналог x.max()
print(np.min(x))             # аналог x.min()
print(np.prod(x))            # произведение элементов
print(np.mean(x))            # среднее значение элементов
print(np.std(x))             # стандартное отклонение
print(np.var(x))             # дисперсия
print(np.argmax(x))          # индекс максимального значения
print(np.argmin(x))          # индекс минимального значения
print(np.median(x))          # медиана
print(np.percentile(x, 25))  # квантили элементов
print(np.any(x))             # существуют ли элементы со значением true
print(np.all(x))             # все ли элементы имеют значение true
```

```
15
5
1
120
3.0
1.4142135623730951
2.0
4
0
3.0
2.0
True
True
```

In [39]:

```
m = np.random.random((3, 4))
print(m)
```

```
[[0.94096894 0.87893607 0.90290652 0.23193166]
 [0.89835856 0.89752619 0.56413736 0.19059505]
 [0.21053988 0.47055481 0.10207379 0.21755797]]
```

In [40]:

```
print(m.sum())
```

```
6.506086804737606
```

In [41]:

```
# минимальное значение каждого из столбцов
print(m.min(axis=0))
```

```
[0.21053988 0.47055481 0.10207379 0.19059505]
```

In [42]:

```
# максимальное значение в каждой из строк
print(m.max(axis=1))
```

```
[0.94096894 0.89835856 0.47055481]
```

Транслирование (broadcasting)

Транслирование представляет собой набор правил по применению бинарных универсальных функций (сложение, вычитание, умножение и т. д.) к массивам различного размера.

In [43]:

```
a = np.array([0, 1, 2])
```

In [44]:

```
# скалярное значение
print(a+5)
```

```
[5 6 7]
```

In [45]:

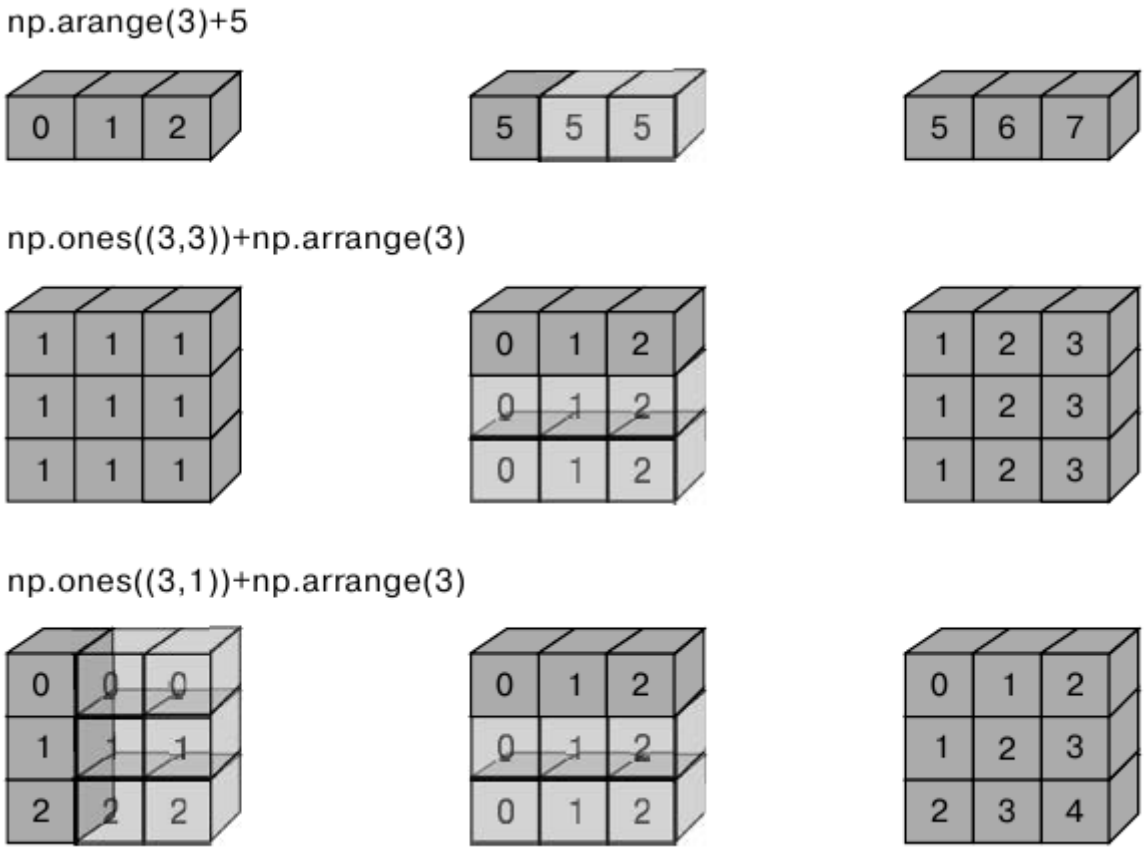
```
# одномерный и двумерный массив
m = np.ones((3, 3))
print(m)
print(m+a)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
```

In [46]:

```
# транслирование обоих массивов
a = np.arange(3)
b = a.reshape((3, 1))
print(a)
print(b)
print(a+b)
```

```
[0 1 2]
[[0]
 [1]
 [2]]
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```



- **Правило 1**: если размерность двух массивов отличается, форма массива с меньшей размерностью дополняется единицами с ведущей (левой) стороны.
- **Правило 2**: если форма двух массивов не совпадает в каком-то измерении, массив с формой, равной 1 в данном измерении, растягивается вплоть до соответствия форме другого массива.
- **Правило 3**: если в каком-либо измерении размеры массивов различаются и ни один не равен 1, генерируется ошибка.

Операторы сравнения

Результат этих операторов сравнения всегда представляет собой массив с булевым типом данных.

In [47]:

```
x = np.array([1,2,3,4,5])
print(x<3)
```

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

In [48]:

```
print(x!=3)
```

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

In [49]:

```
print(x*2 == x+2)
```

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

Булевые массивы

In [50]:

```
# подсчет количества элементов
print(np.count_nonzero(x<=3))
```

```
3
```

In [51]:

```
print(np.sum(x<=3))
```

```
3
```

In [52]:

```
print(np.any(x>3))
```

```
True
```

In [53]:

```
print(np.all(x>3))
```

```
False
```

При создании булева выражения с заданным массивом следует использовать операторы **&** и **|**, а не операции **and** или **or**.

In [54]:

```
# & - and
# | - or
# ^ - xor
# ~ - not
print(np.sum((x>=2) & (x<=4)))
```

```
3
```

In [55]:

```
# маскирование (отображение значение подходящих под условие)
print(x[x <= 3])
```

```
[1 2 3]
```

Прихотливая индексация (fancy indexing)

In [56]:

```
print(x)
print(x[[2,3]])
```

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

В случае «прихотливой» индексации форма результата отражает форму массивов индексов (index arrays), а не форму индексируемого массива.

In [57]:

```
inds = np.array([[3],[4]])
print(x[inds])
```

```
[[4]
 [5]]
```

In [58]:

```
# многомерный массив
xm = np.arange(12).reshape(3,4)
print(xm)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [59]:

```
row = np.array([0,1,2])
col = np.array([2,1,3])
print(xm[row, col])
```

```
[ 2  5 11]
```

In [60]:

```
# использование среза
print(xm[1:, col])
```

```
[[ 6  5  7]
 [10  9 11]]
```

In [61]:

```
# изменение значений
xx = np.array([1,2,3,4,5])
print(xx)
```

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

In [62]:

```
i = np.array([1,2,3])
xx[i] += 2
print(xx)
```

```
[1 4 5 6 5]
```

Сортировка массива

In [63]:

```
xs = np.array([3,6,1,7,0,9,4])
print(np.sort(xs))
```

```
[0 1 3 4 6 7 9]
```

In [64]:

```
# индексы отсортированных элементов
print(np.argsort(xs))
```

```
[4 2 0 6 1 3 5]
```

In [65]:

```
# многомернный массив
rnd = np.random.RandomState(42)
xm = rnd.randint(0, 10, (4, 6))
print(xm)
```

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

In [66]:

```
# сортировка по столбцам
print(np.sort(xm, axis=0))
```

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

In [67]:

```
# сортировка по строкам
print(np.sort(xm, axis=1))
```

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

In [68]:

```
# секционирование
# результат представляет собой новый массив с K наименьшими значениями
# слева от точки разбиения и остальные значения справа от нее
# в произвольном порядке
xc = np.array([7,2,3,1,6,5,4])
print(np.partition(xc, 3))
```

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