https://dfedorov.spb.ru

Массивная арифметика

Давайте создадим два массива NumPy, чтобы продемонстрировать их полезность. Назовем их **data** и **ones**:

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

data

1
2

```
ones = np.ones(2)
```

ones

1
1

Добавить их по позициям (то есть добавить значения каждой строки) так же просто, как ввести **data + ones**:

data + ones =

data

1
2

+

ones

1
1

=

2
3

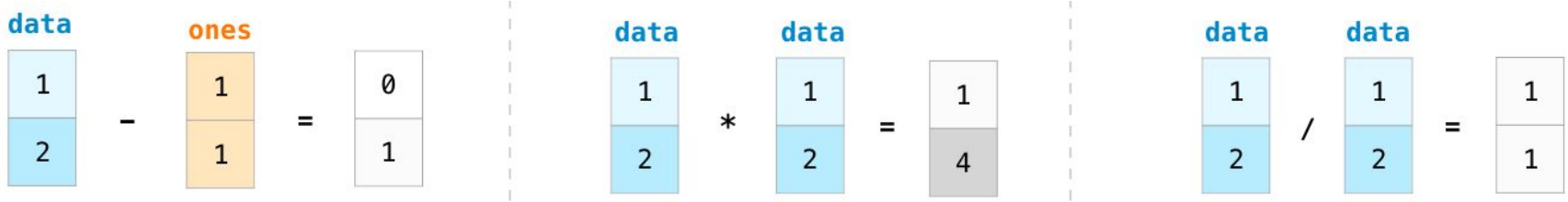
1	2	+	4	8	=	5 10
1	2	-	4	8	=	-3 -6

4	8	*	2	5	=	8 40	
4	8	/	2	5	=	2.0 1.6	np.float64
4	8	//	2	5	=	2 1	np.int32

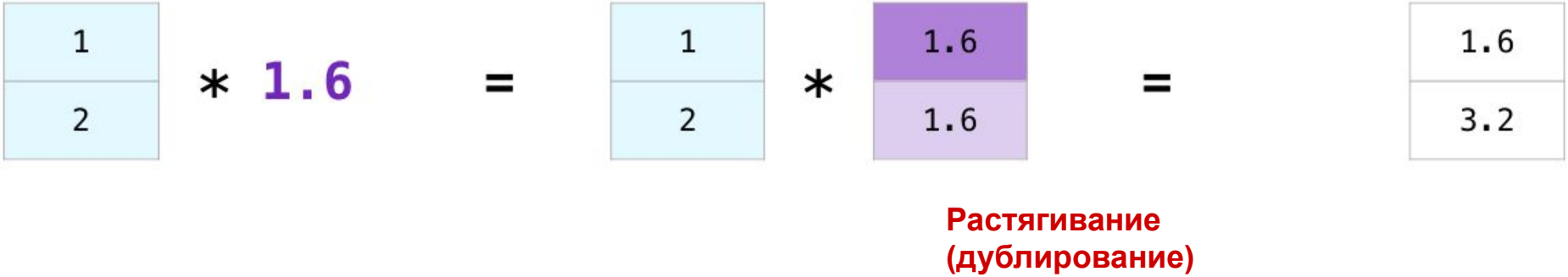
3	4	**	2	3	=	9 64

Векторизованные операции

Такая абстракция избавляет от необходимости программировать вычисления в циклах. Это прекрасная абстракция, которая позволяет думать о проблемах на более высоком уровне:



Часто бывают случаи, когда мы хотим выполнить операцию между массивом и числом (операция между вектором и скаляром).
Скажем, например, наш массив представляет расстояние в милях, и мы хотим преобразовать его в километры. Мы просто говорим **data * 1.6**:



Видите, как NumPy понял, что операция означает, что умножение должно происходить с каждой ячейкой? Эта концепция называется *трансляцией*.

1	2	+	3	=	4	5
1	2	-	3	=	-2	-1

1	2	*	3	=	3	6	
1	2	/	3	=	0.33	0.67	np.float64
1	2	//	2	=	0	1	np.int32

3	4	**	2	=	9	16

универсальные функции (ufuncs)

```
>>> x = np.arange(4)
>>> print("x     =", x)
>>> print("x + 5 =", x + 5)
>>> print("x - 5 =", x - 5)
>>> print("x * 2 =", x * 2)
>>> print("x / 2 =", x / 2)
>>> print("x // 2 =", x // 2)

>>> print("-x     = ", -x)
>>> print("x ** 2 = ", x ** 2)
>>> print("x % 2  = ", x % 2)

>>> -(0.5*x + 1) ** 2
```

Оператор	Эквивалентная универсальная функция	Описание
+	np.add	Сложение (например, 1 + 1 = 2)
–	np.subtract	Вычитание (например, 3 – 2 = 1)
–	np.negative	Унарная операция изменения знака (например, –2)
*	np.multiply	Умножение (например, 2 * 3 = 6)
/	np.divide	Деление (например, 3 / 2 = 1.5)
//	np.floor_divide	Деление с округлением в меньшую сторону (например, 3 // 2 = 1)
**	np.power	Возведение в степень (например, 2 ** 3 = 8)
%	np.mod	Модуль/остаток (например, 9 % 4 = 1)

```
>>> x = np.array([-2, -1, 0, 1, 2])
>>> abs(x) # опасность!

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


>>> np.absolute(x)

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

>>> np.abs(x)

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

```
theta = np.linspace(0, np.pi, 3) # массив из 3 элементов равномерно располож. между 0 и np.pi
print("theta      = ", theta)
print("sin(theta) = ", np.sin(theta))
print("cos(theta) = ", np.cos(theta))
print("tan(theta) = ", np.tan(theta))
```

```
theta      =  [ 0.          1.57079633  3.14159265]
sin(theta) =  [  0.00000000e+00   1.00000000e+00   1.22464680e-16]
cos(theta) =  [  1.00000000e+00   6.12323400e-17  -1.00000000e+00]
tan(theta) =  [  0.00000000e+00   1.63312394e+16  -1.22464680e-16]
```

```
x = [-1, 0, 1]
print("x         = ", x)
print("arcsin(x) = ", np.arcsin(x))
print("arccos(x) = ", np.arccos(x))
print("arctan(x) = ", np.arctan(x))
```

```
x         =  [-1, 0, 1]
arcsin(x) =  [-1.57079633  0.          1.57079633]
arccos(x) =  [ 3.14159265  1.57079633  0.        ]
arctan(x) =  [-0.78539816  0.          0.78539816]
```

```
x = [1, 2, 3]
print("x     =", x)
print("e^x   =", np.exp(x))
print("2^x   =", np.exp2(x))
print("3^x   =", np.power(3, x))
```

```
x     = [1, 2, 3]
e^x   = [  2.71828183   7.3890561   20.08553692]
2^x   = [ 2.  4.  8.]
3^x   = [ 3  9 27]
```

```
x = [1, 2, 4, 10]
print("x        =", x)
print("ln(x)    =", np.log(x))
print("log2(x)  =", np.log2(x))
print("log10(x) =", np.log10(x))
```

```
x        = [1, 2, 4, 10]
ln(x)    = [ 0.          0.69314718  1.38629436  2.30258509]
log2(x)  = [ 0.          1.          2.          3.32192809]
log10(x) = [ 0.          0.30103     0.60205999  1.        ]
```

```
x = [0, 0.001, 0.01, 0.1]
print("exp(x) - 1 =", np.expm1(x))
print("log(1 + x) =", np.log1p(x))
```

```
exp(x) - 1 = [ 0.          0.0010005   0.01005017  0.10517092]
log(1 + x) = [ 0.          0.0009995   0.00995033  0.09531018]
```

документация

a^2	=	2	3	** 2	=	4	9
$\sqrt{a}$	=	np.sqrt(4	9)		=	2.	3.
e^a	=	np.exp(1	2)		=	2.718	7.389
$\ln a$	=	np.log(np.e	np.e**2)		=	1.	2.

np.sin(	np.pi	np.pi/2	) =	0.	1.	

np.arcsin(	0.	1.	) =	0.	1.57	

sin	arcsin	sinh	arcsinh
cos	arccos	cosh	arccosh
tan	arctan	tanh	arctanh

np.hypot(	3.	5.	,	4.	12.	) =	5.	13.

np.floor(	1.1	1.5	1.9	2.5	) =	1.	1.	1.	2.
np.ceil(	1.1	1.5	1.9	2.5	) =	2.	2.	2.	3.
np.round(	1.1	1.5	1.9	2.5	) =	1.	2.	2.	2.

```python
from scipy import special


# Gamma functions (generalized factorials) and related functions
x = [1, 5, 10]
print("gamma(x)     =", special.gamma(x))
print("ln|gamma(x)| =", special.gammaln(x))
print("beta(x, 2)   =", special.beta(x, 2))


# Error function (integral of Gaussian)
# its complement, and its inverse
x = np.array([0, 0.3, 0.7, 1.0])
print("erf(x)  =", special.erf(x))
print("erfc(x) =", special.erfc(x))
print("erfinv(x) =", special.erfinv(x))
```

Документация

```
>>> x = np.arange(5)
>>> y = np.empty(5) # произвольный массив с “мусором”
>>> np.multiply(x, 10, out=y) # место для записи результатов
>>> print(y)
```

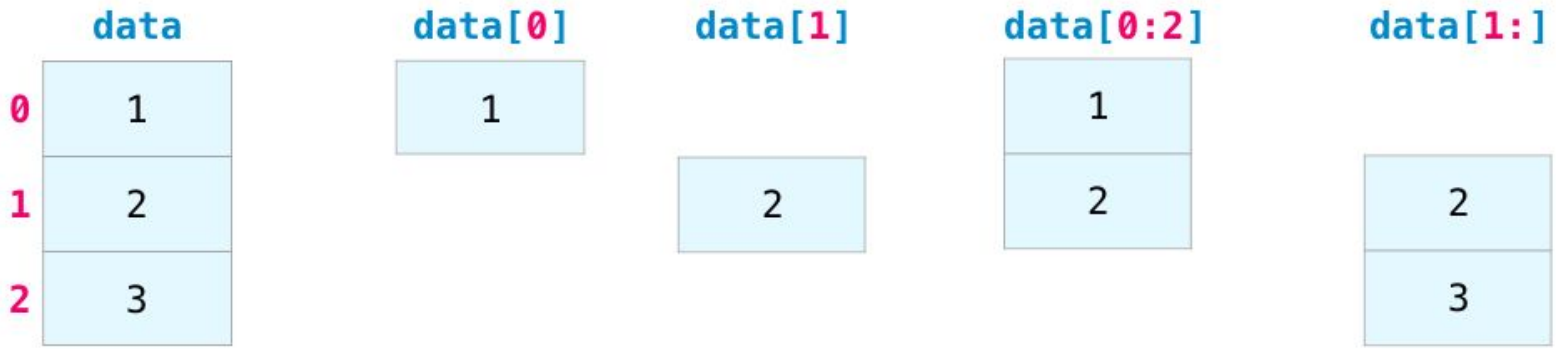
```
>>> y[::2] = 2 ** x
```

Создается временный массив с результатами операций **2 ** x**, затем он копируется в массив **y**.

```
>>> y = np.zeros(10)
# записать результат в каждый второй элемент заданного массива
>>> np.power(2, x, out=y[::2])
>>> print(y)
```

Индексирование

Мы можем индексировать и нарезать массивы NumPy всеми способами, которыми мы можем нарезать списки Python:




```
>>> data[-1] = 1.8
>>> data[-1]
```

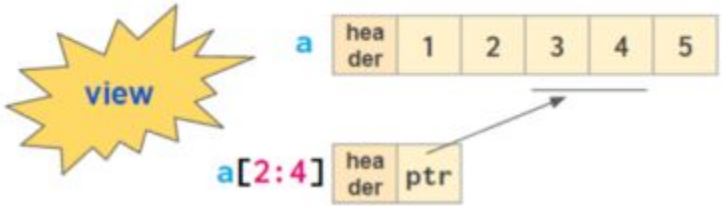
```
>>> x = np.arange(10)
>>> x
>>> x[:5]
>>> x[5:]
>>> x[4:7]
>>> x[::2]
>>> x[1::2]
>>> x[::-1]
```

```
a = np.arange(1, 6)
```

1	2	3	4	5
0	1	2	3	4

a[1]	a[2:4]		a[-2:]		a[::2]			a[[1,3,4]]		
2	3	4	4	5	1	3	5	2	4	5

"fancy indexing"




```
a[2:4] = 0
```

a

1	2	0	0	5

Представление

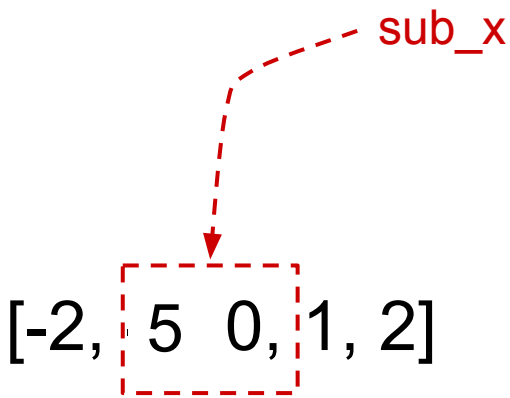
```
>>> x = np.array([-2, -1, 0, 1, 2])

>>> sub_x = x[1:3]

>>> sub_x

>>> sub_x[0] = 5

>>> x
```



Маскирование (masking)

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

>>> x < 3  # less than

      array([ True,  True, False, False, False], dtype=bool)

>>> x > 3  # greater than

      array([False, False, False,  True,  True], dtype=bool)

>>> x <= 3  # less than or equal

      array([ True,  True,  True, False, False], dtype=bool)

>>> x >= 3  # greater than or equal

      array([False, False,  True,  True,  True], dtype=bool)

>>> x != 3  # not equal

      array([ True,  True, False,  True,  True], dtype=bool)

>>> x == 3  # equal

      array([False, False,  True, False, False], dtype=bool)
```

Массив с булевым типом данных

Оператор	Эквивалентная универсальная функция
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

Булевы массивы как маски!

Прихотливая индексация (fancy indexing)

```
>>> import numpy as np
>>> rand = np.random.RandomState(42) # seed(42)

>>> x = rand.randint(100, size=10)
>>> print(x)

[51 92 14 71 60 20 82 86 74 74]
```

Прихотливая, т.к. форма результата отражает форму массива индексов

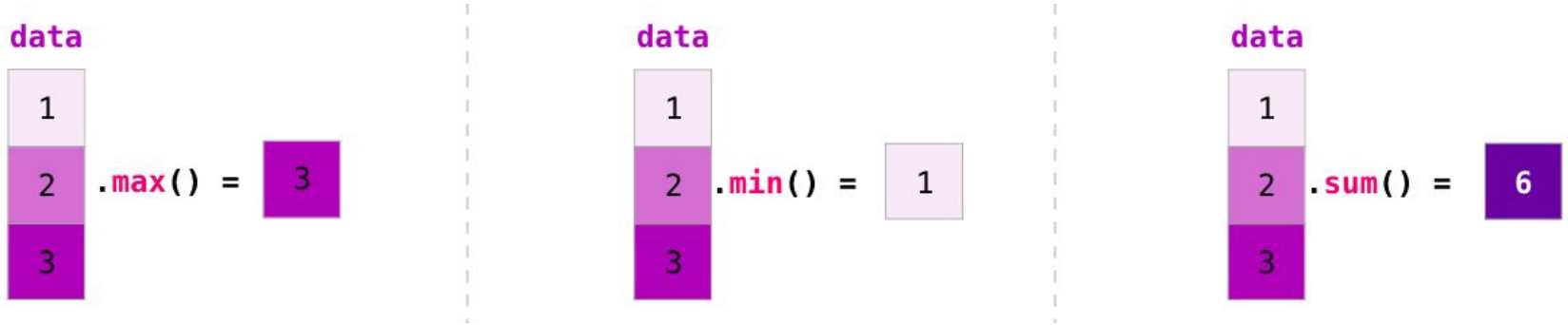
```
>>> ind = [3, 7, 4]
>>> x[ind]

array([71, 86, 60])
```

Список индексов

Агрегация

Дополнительные преимущества NumPy - это *функции агрегирования*:




```
>>> import numpy as np
>>> L = np.random.random(100)
>>> L.max() # работает быстрее, чем max(L)
```

a

np.max(1 2 3) = 3

1 2 3 .max() = 3

1 2 3 .argmax() = 2

1 2 3 .min() = 1

1 2 3 .argmin() = 0

0 1 2

1 2 3 .sum() = 6

1 2 3 .mean() = 2

1 2 3 .var() = 0.67

1 2 3 .std() = 0.82

$$\overline{S}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2, \quad \overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

a = 2 ± 0.82

```
>>> import numpy as np
>>> L = np.random.random(100)
>>> np.sum(L) # работает быстрее, чем sum(L)
```

Игнорируют отсутствующие значения

Имя функции	NaN-безопасная версия	Описание
np.sum	np.nansum	Вычисляет сумму элементов
np.prod	np.nanprod	Вычисляет произведение элементов
np.mean	np.nanmean	Вычисляет среднее значение элементов
np.std	np.nanstd	Вычисляет стандартное отклонение
np.var	np.nanvar	Вычисляет дисперсию
np.min	np.nanmin	Вычисляет минимальное значение
np.max	np.nanmax	Вычисляет максимальное значение
np.argmin	np.nanargmin	Возвращает индекс минимального значения
np.argmax	np.nanargmax	Возвращает индекс максимального значения
np.median	np.nanmedian	Вычисляет медиану элементов
np.percentile	np.nanpercentile	Вычисляет квантили элементов
np.any	N/A	Проверяет, существуют ли элементы со значением true
np.all	N/A	Проверяет, все ли элементы имеют значение true

a	1	2	3	4	5	6	7	6	5	4	3	2	1
a > 5	False	False	False	False	False	True	True	True	False	False	False	False	False

`np.any(a > 5)`

True

`a[a > 5]`

6	7	6

`np.all(a > 5)`

False

`a[a > 5] = 0`

a	1	2	3	4	5	0	0	0	5	4	3	2	1

`a[(a >= 3) & (a <= 5)] = 0`

a	1	2	0	0	0	6	7	6	0	0	0	2	1

&	and
|	or
^	xor
~	not

https://dfedorov.spb.ru