



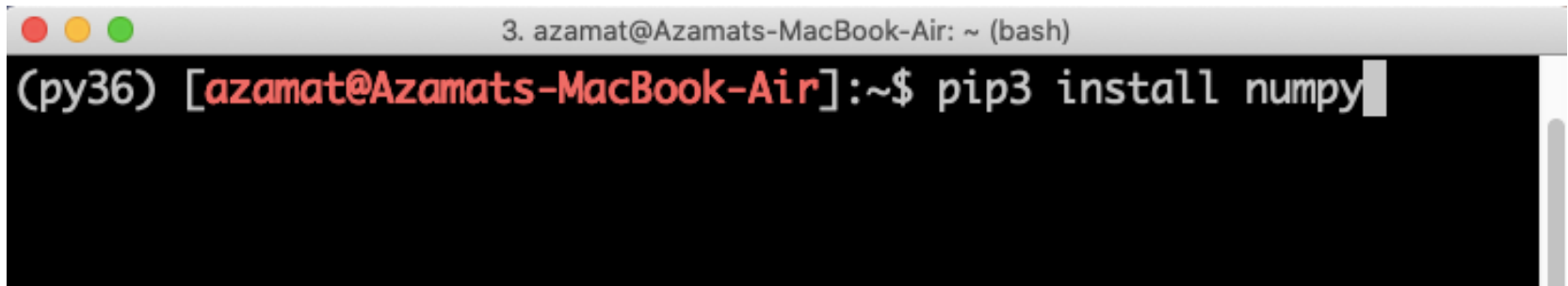
# **NumPy: Введение**

Data pre-processing and visualization

# Что такое NumPy?

- **NumPy** – это фундаментальный пакет для научных вычислений на Python. Он содержит:
  - мощный N-мерный массив объектов
  - полезная линейная алгебра, преобразование Фурье и возможности случайных чисел

# Как установить NumPy?



```
3. azamat@Azamats-MacBook-Air: ~ (bash)
(py36) [azamat@Azamats-MacBook-Air]:~$ pip3 install numpy
```

# Как импортировать NumPy?

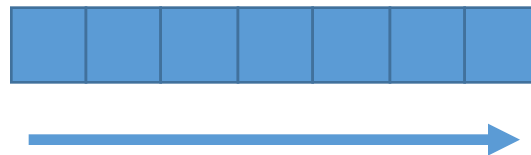
```
import numpy as np
```

```
print(dir(np))
```

```
['ALLOW_THREADS', 'AxisError', 'BUFSIZE', 'CLIP', 'ComplexWarning', 'DataSource', 'ERR_CALL',  
'ERR_DEFAULT', 'ERR_IGNORE', 'ERR_LOG', 'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT  
_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'In  
f', 'Infinity', 'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT', 'MachAr', 'ModuleDeprecatio  
nWarning', 'NAN', 'NINF', 'NZERO', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT_DIV  
IDEBYZERO', 'SHIFT_INVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'Tester', 'To  
oHardError', 'True_', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'VisibleDeprecationWarnin  
g', 'WRAP', '_NoValue', '_UFUNC_API', '_NUMPY_SETUP_', '_all_', '_builtins_', '_cached  
_', '_config_', '_doc_', '_file_', '_git_revision_', '_loader_', '_name_', '_pa  
ckage_', '_path_', '_spec_', '_version_', '_add_newdoc_ufunc', '_distributor_init', '_  
globals_', '_mat_', '_pytesttester', 'abs', 'absolute', 'absolute_import', 'add', 'add_docstrin  
g', 'add_newdoc', 'add_newdoc_ufunc', 'alen', 'all', 'allclose', 'alltrue', 'amax', 'amin',  
'angle', 'any', 'append', 'apply_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccos  
h', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argpartition',  
'argsort', 'argwhere', 'around', 'array', 'array2string', 'array_equal', 'array_equiv', 'arra  
y_repr', 'array_split', 'array_str', 'asanyarray', 'asarray', 'asarray_chkfinite', 'ascontigu  
ousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar', 'atleast_1d', 'atleast_2d',  
'atleast_3d', 'average', 'bartlett', 'base_repr', 'binary_repr', 'bincount', 'bitwise_and',  
'bitwise_not', 'bitwise_or', 'bitwise_xor', 'blackman', 'block', 'bmat', 'bool', 'bool8', 'bo  
ol_', 'broadcast', 'broadcast_arrays', 'broadcast_to', 'busday_count', 'busday_offset', 'busd  
aycalendar', 'byte', 'byte_bounds', 'bytes0', 'bytes_', 'c_', 'can_cast', 'cast', 'cbrt', 'cd  
ouble', 'ceil', 'cfloat', 'char', 'character', 'chararray', 'choose', 'clip', 'clongdouble',  
'clongfloat', 'column_stack', 'common_type', 'compare_chararrays', 'compat', 'complex', 'comp  
lex128', 'complex256', 'complex64', 'complex_', 'complexfloating', 'compress', 'concatenate',  
'conj', 'conjugate', 'convolve', 'copy', 'copysign', 'copyto', 'core', 'corrcoef', 'correlat  
e', 'cos', 'cosh', 'count_nonzero', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumpr  
oduct', 'cumsum', 'datetime64', 'datetime_as_string', 'datetime_data', 'deg2rad', 'degrees',  
'delete', 'deprecated', 'deprecated with doc', 'diag', 'diag_indices', 'diag_indices from', 'di
```

# Введение в NumPy

- Основной структурой данных в NumPy является **ndarray** (n-мерный массив).
- **ndarray** –это многомерный массив элементов, **ОДНОГО** типа.
- Пример:



Ось (измерение) массива

# Как создать **ndarray** (массив)?

- Самый простой метод – использовать **список**

```
import numpy as np
```

```
a = [1, 2, 3]
```

```
a_np = np.array(a)
```

```
print(a_np)
```

```
print(type(a_np))
```

```
[1 2 3]
```

```
<class 'numpy.ndarray'>
```

# Как создать **ndarray** (массив)?

- Самый простой метод – использовать **список**

numpy.ndarray

```
import numpy as np
```

```
a = [1, 2, 3]
```

```
a_np = np.array(a)
```

numpy.array(list)

```
print(a_np)
```

```
print(type(a_np))
```

```
[1 2 3]
```

```
<class 'numpy.ndarray'>
```

# Важные атрибуты ndarray (массива):

Атрибут:	Значение:
<code>ndarray.ndim</code>	число осей (измерений)
<code>ndarray.shape</code>	кортеж натуральных чисел, показывающий длину массива по каждой оси
<code>ndarray.size</code>	количество элементов массива
<code>ndarray.dtype</code>	объект, описывающий тип элементов массива
<code>ndarray.T</code>	транспонирование массива



# Типы данных в NumPy

- `bool`, `character`, `int`, `float`, `complex`

```
b_np = np.array([10, 20, 30], dtype=np.float)
```

```
print(b_np)  
print(b_np.dtype)
```

```
[10. 20. 30.]  
float64
```

# Типы данных в NumPy

- `bool`, `character`, `int`, `float`, `complex`

```
b_np = np.array([10, 20, 30], dtype=np.float)
```

```
print(b_np)  
print(b_np.dtype)
```

```
[10. 20. 30.]  
float64
```

# Типы данных в NumPy

- `bool`, `character`, `int`, `float`, `complex`

```
b_np = np.array([10, 20, 30], dtype=np.float)
```

```
print(b_np)  
print(b_np.dtype)
```

```
[10. 20. 30.]  
float64
```

# Одномерный массив

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

```
print(c_np)
```

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

```
print(c_np.shape)
```

```
(4,)
```

# Одномерный массив

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

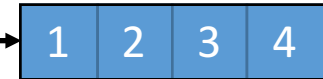
```
print(c_np)
```

[1 2 3 4]

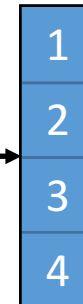
```
print(c_np.shape)
```

(4, )

Строка?



Столбец?



# Одномерный массив

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

```
print(c_np)
```

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

```
print(c_np.shape)
```

```
(4, )
```

Строка



Столбец



**Настоятельно  
НЕ  
рекомендуется !!!**

# Почему **НЕ** рекомендуется?

- Что произойдет, если вы выполните операции на 1-мерном массиве (**numpy.ndarray**) ???

```
a_np = np.array([1,2,3,4])  
print(a_np)  
print(a_np.ndim)  
print(a_np.shape)
```

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

- Решение: 1-мерный массив → 2-мерный массив

# Двухмерный массив

```
import numpy as np
```

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

```
print(two_d.ndim)
```

2

```
print(two_d.shape)
```

(2, 3)

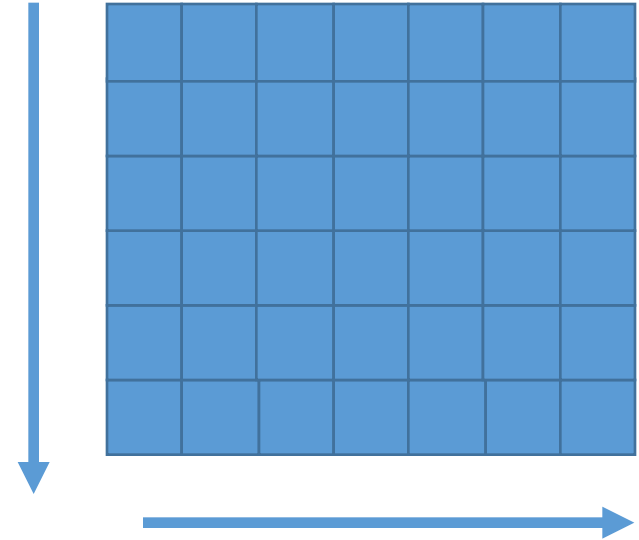
```
print(two_d)
```

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

```
print(two_d.size)
```

6

Ось (измерение) массива



Ось (измерение) массива



# Двухмерный массив

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

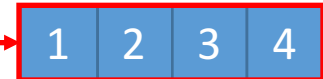
```
print(row)
```

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

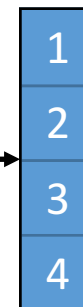
```
print(row.shape)
```

```
(1, 4)
```

Строка?



~~Столбец?~~



# Двухмерный массив

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

```
print(col)
```

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

```
print(col.shape)
```

```
(4, 1)
```



Столбец?



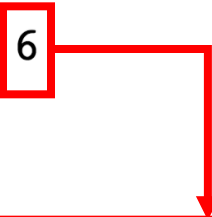
# Методы `ndarray` (массива)

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

```
max_val = arr.max() # min(), mean(), std()
```

```
print(max_val)
```

6



1	2	3
4	5	6

## Замечания:

работает со всем массивом  
(независимо от количества  
измерений)

# Методы `ndarray` (массива):

```
import numpy as np
```

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

```
max_val = arr.max(axis=0) # min(), mean(), std()
```

```
print(max_val)
```

```
[4 5 6]
```



Аргумент:  
**`axis < ndarray.ndim`**

`axis=0`

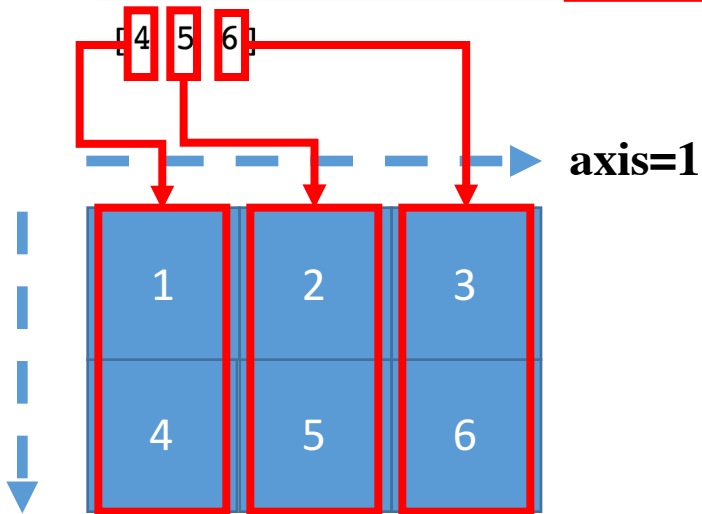
# Методы `ndarray` (массива):

```
import numpy as np
```

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

```
max_val = arr.max(axis=0) # min(), mean(), std()
```

```
print(max_val)
```



Аргумент:  
`axis < ndarray.ndim`

`axis=0`

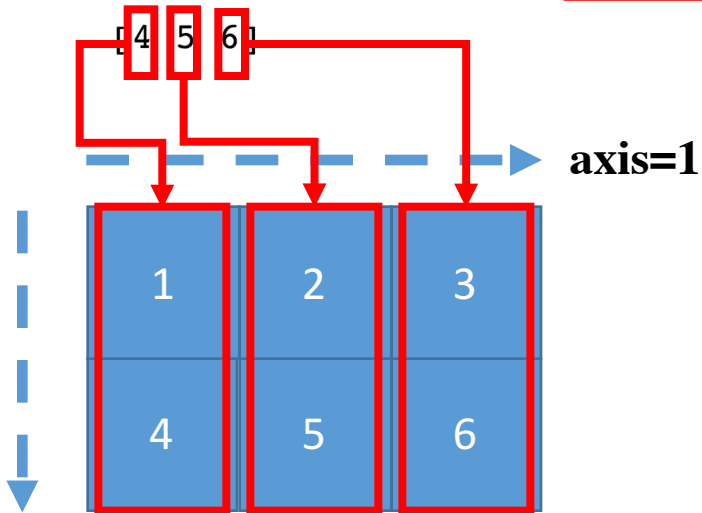
# Методы `ndarray` (массива):

```
import numpy as np
```

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

```
max_val = arr.max(axis=0) # min(), mean(), std()
```

```
print(max_val)
```



**Аргумент:**  
`axis < ndarray.ndim`

**Замечания:**  
 результатом является  
 одномерный массив

# Методы `ndarray` (массива):

```
import numpy as np
```

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

```
max_val = arr.max(axis=0, keepdims=True) # min(), mean(), std()
```

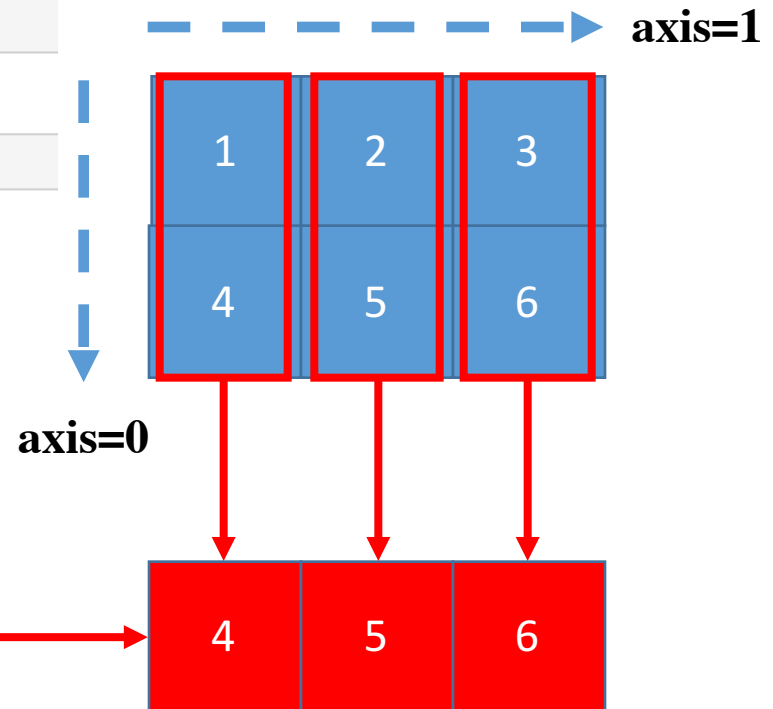
```
print(max_val)
```

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

```
print(max_val.shape)
```

```
(1, 3)
```

Аргумент:  
**`keepdims = True`**



# Методы `ndarray` (массива):

## Методы `ndarray`:

<code>ndarray.reshape()</code>	<code>ndarray.argmin()</code>
<code>ndarray.ravel()</code>	<code>ndarray.argmax()</code>
<code>ndarray.tolist()</code>	<code>ndarray.cumsum()</code>
<code>ndarray.tofile()</code>	<code>ndarray.cumprod()</code>
<code>ndarray.sort()</code>	<code>ndarray.dot()</code>
<code>ndarray.transpose()</code>	<code>ndarray.squeeze()</code>
<code>ndarray.flatten()</code>	<code>ndarray.repeat()</code>





# **NumPy: Индексы & Срезы**

Data pre-processing and visualization

# Индексы `ndarray` vs `list`

- Одномерный массив (`ndarray`) = список (`list`)

Индексы:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элементы:	1	2	3	4	5	2	1	7	9	2	8	4	5	2	2	0

# Индексы `ndarray` vs `list`

- Одномерный массив (`ndarray`) = список (`list`)

Индексы:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Элементы:

1	2	3	4	5	2	1	7	9	2	8	4	5	2	2	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

`list` (список):

```
list_of_nums = [1,2,3,4,5,2,1,7,9,2,8,4,5,2,2,0]
print(list_of_nums[3])
print(list_of_nums[5])
```

4  
2

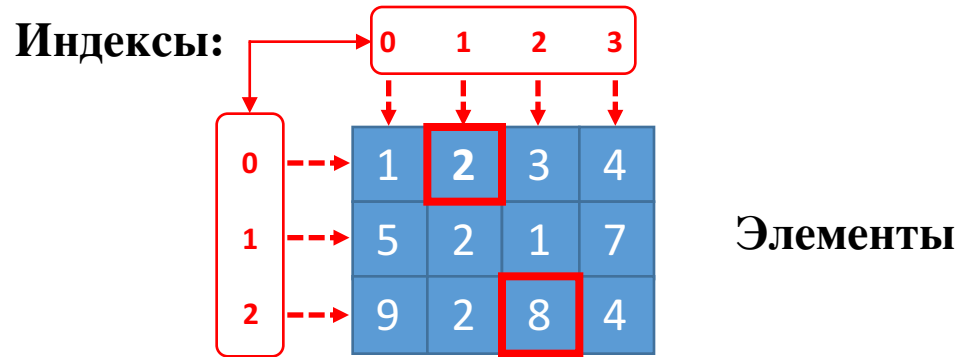
`ndarray` (массив):

```
ndarr_of_nums = np.array(list_of_nums)
print(ndarr_of_nums[3])
print(ndarr_of_nums[5])
```

4  
2

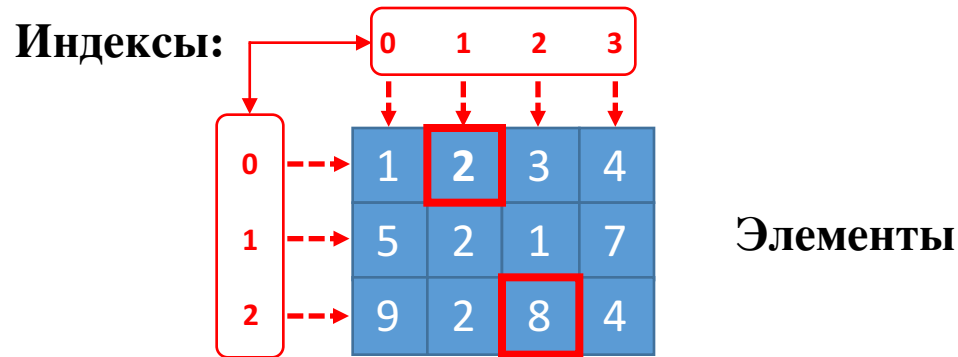
# Индексы `ndarray` vs `list`

- Двумерный массив (`ndarray`)  $\neq$  список (`list`)



# Индексы `ndarray` vs `list`

- Двумерный массив (`ndarray`)  $\neq$  список (`list`)



`list` (список):

```
list_2d_nums = [[1,2,3,4],[5,2,1,7],[9,2,8,4]]
print(list_2d_nums[0][1])
print(list_2d_nums[2][2])
```

2  
8

`ndarray` (массив):

```
ndarr_2d_nums = np.array(list_2d_nums)
print(ndarr_2d_nums[0,1])
print(ndarr_2d_nums[2,2])
```

2  
8

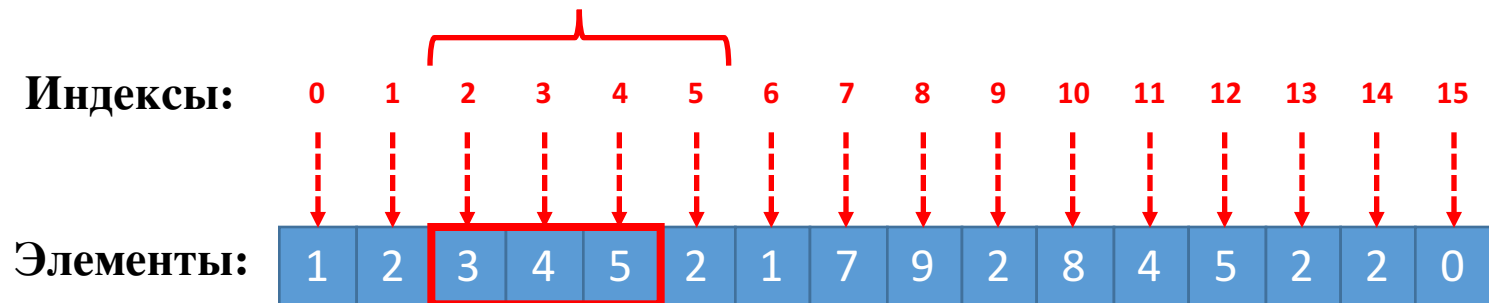
# Срезы `ndarray` vs `list`

- Одномерный массив (`ndarray`) = список (`list`)

Индексы:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Элементы:	1	2	3	4	5	2	1	7	9	2	8	4	5	2	2	0

# Срезы `ndarray` vs `list`

- Одномерный массив (`ndarray`) = список (`list`)



```
list_of_nums = [1,2,3,4,5,2,1,7,9,2,8,4,5,2,2,0]
print(list_of_nums[2:5])
```

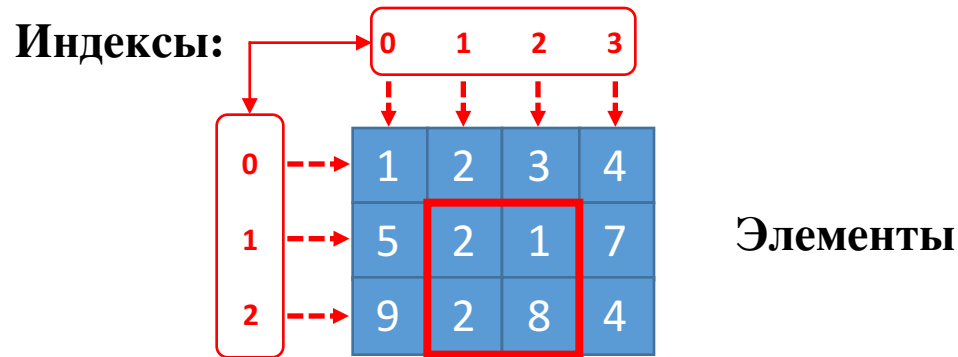
```
[3, 4, 5]
```

```
ndarr_of_nums = np.array(list_of_nums)
print(ndarr_of_nums[2:5])
```

```
[3 4 5]
```

# Срезы `ndarray` vs `list`

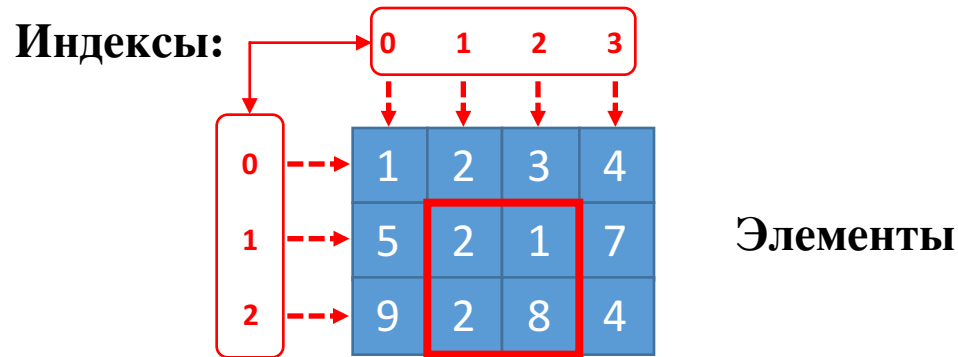
- Двумерный массив (`ndarray`)  $\neq$  список (`list`)





# Срезы `ndarray` vs `list`

- Двумерный массив (`ndarray`)  $\neq$  список (`list`)



`list` (список):

```
list_2d_nums = [[1,2,3,4],[5,2,1,7],[9,2,8,4]]
print(list_2d_nums[1:3][1:3])
```

```
[[9, 2, 8, 4]]
```

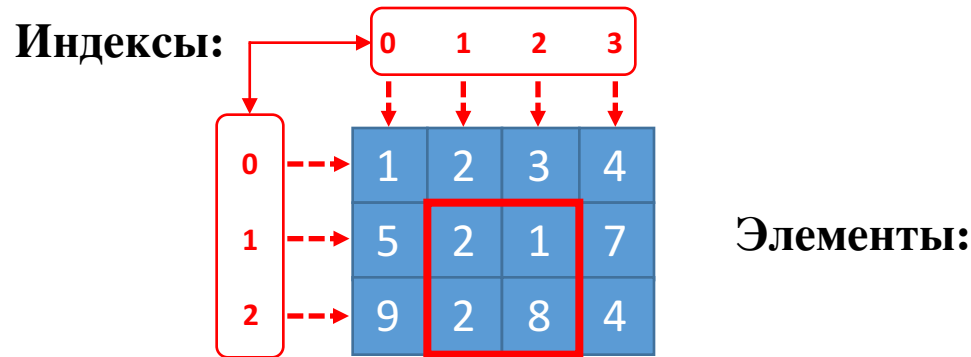
`ndarray` (массив):

```
ndarr_2d_nums = np.array(list_2d_nums)
print(ndarr_2d_nums[1:3,1:3])
```

```
[[2 1]
 [2 8]]
```

# Срезы `ndarray` vs `list`

- Двумерный массив (`ndarray`)  $\neq$  список (`list`)



`list` (список):

```
list_2d_nums = [[1, 2, 3, 4], [5, 2, 1, 7], [9, 2, 8, 4]]
print(list_2d_nums[1:3, 1:3])
```

**WRONG**

```
[[9, 2, 8, 4]]
```

`ndarray` (массив):

```
ndarr_2d_nums = np.array(list_2d_nums)
print(ndarr_2d_nums[1:3, 1:3])
```

```
[[2 1]
 [2 8]]
```



# **NumPy: Векторизация & Broadcasting**

Data pre-processing and visualization

# NumPy: Векторизация

- **Векторизованные операции**

- операции применяются к нескольким точкам данных одновременно.

1	+	2	=	3
5	+	2	=	7
3	+	2	=	5
3	+	4	=	7
1	+	7	=	8
5	+	4	=	9

**list** (список)

1		2		3
5		2		7
3		2		5
3	+	4	=	7
1		7		8
5		4		9

**ndarray** (массив)

# NumPy: Векторизация

- Избегаем использования циклов, и проводим вычисления с целыми массивами за один раз (или как минимум с большими частями массивов).

**list** (СПИСОК):

```
import numpy as np
```

```
list_of_nums = [[1,5,3,3,1,5],[2,2,2,4,7,4]]  
sum_of_nums = []  
for num1, num2 in zip(*list_of_nums):  
    sum_of_nums.append(num1+num2)  
print(sum_of_nums)
```

```
[3, 7, 5, 7, 8, 9]
```

**ndarray** (МАССИВ):

```
ndarr_of_nums = np.array(list_of_nums)  
ndarr_sum_of_nums = ndarr_of_nums[0] + ndarr_of_nums[1]  
print(ndarr_sum_of_nums)
```

```
[3 7 5 7 8 9]
```

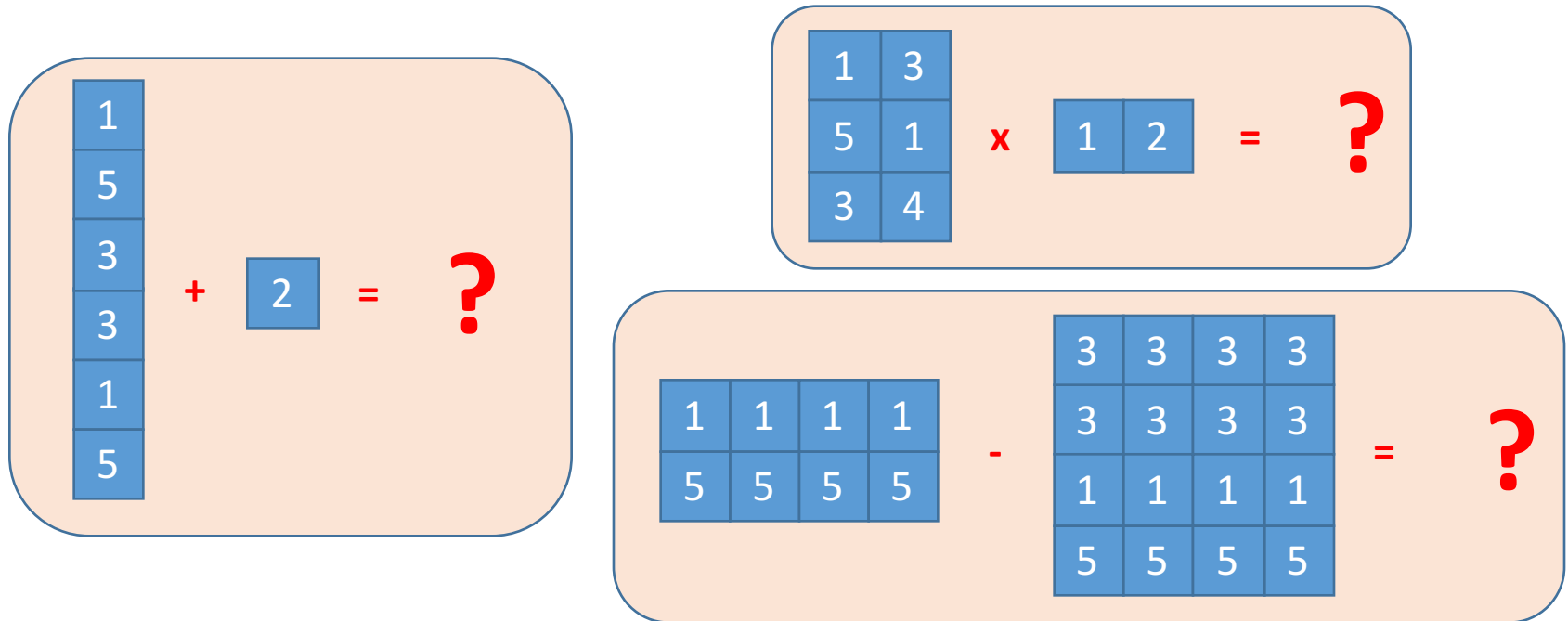
# NumPy: Векторизация

```
ndarr1=ndarr_of_nums[0]
ndarr2=ndarr_of_nums[1]
# Basic vectorized operations
print(ndarr1)
print(ndarr2)
print(ndarr1-ndarr2)
print(ndarr1*ndarr2)
print(ndarr1/ndarr2)
print(ndarr1%ndarr2)
print(ndarr1>ndarr2)
```

```
[1 5 3 3 1 5]
[2 2 2 4 7 4]
[-1  3  1 -1 -6  1]
[ 2 10  6 12  7 20]
[0.5          2.5          1.5          0.75          0.14285714  1.25          ]
[1 1 1 3 1 1]
[False  True  True False False  True]
```

# NumPy: Broadcasting

- Термин «**broadcasting**» описывает, как NumPy обрабатывает массивы (**ndarrays**) различной формы во время арифметических операций.

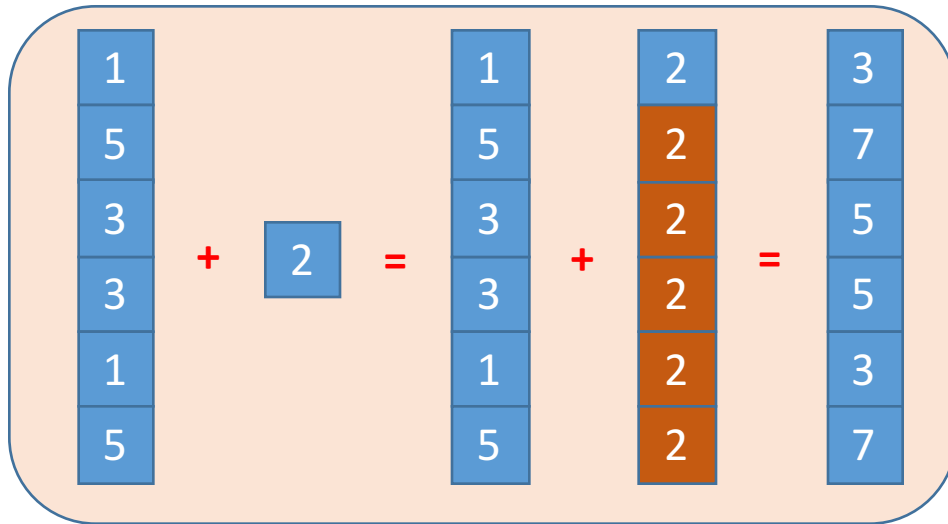


# NumPy: Broadcasting

- Как работает **broadcasting**?
  - При работе с двумя массивами **ndarray** сравнивает их формы (**ndarray.shape**) поэлементно. Начинается с задних размеров и движется вперед. Два измерения совместимы, когда
    - они равны или
    - один из них 1



# NumPy: Broadcasting

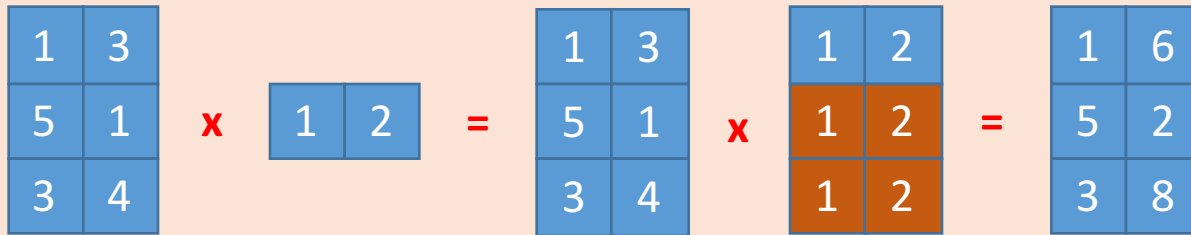


```
import numpy as np
```

```
# Broadcasting  
ndarr1 = np.array([1,5,3,3,1,5])  
print(ndarr1+2)
```

```
[3 7 5 5 3 7]
```

# NumPy: Broadcasting


$$\begin{bmatrix} 1 & 3 \\ 5 & 1 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 5 & 1 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 6 \\ 5 & 2 \\ 3 & 8 \end{bmatrix}$$

```
ndarr2 = np.array([[1,3],[5,1],[3,4]])  
ndarr3 = np.array([1,2])  
print(ndarr2*ndarr3)
```

```
[[1 6]  
 [5 2]  
 [3 8]]
```

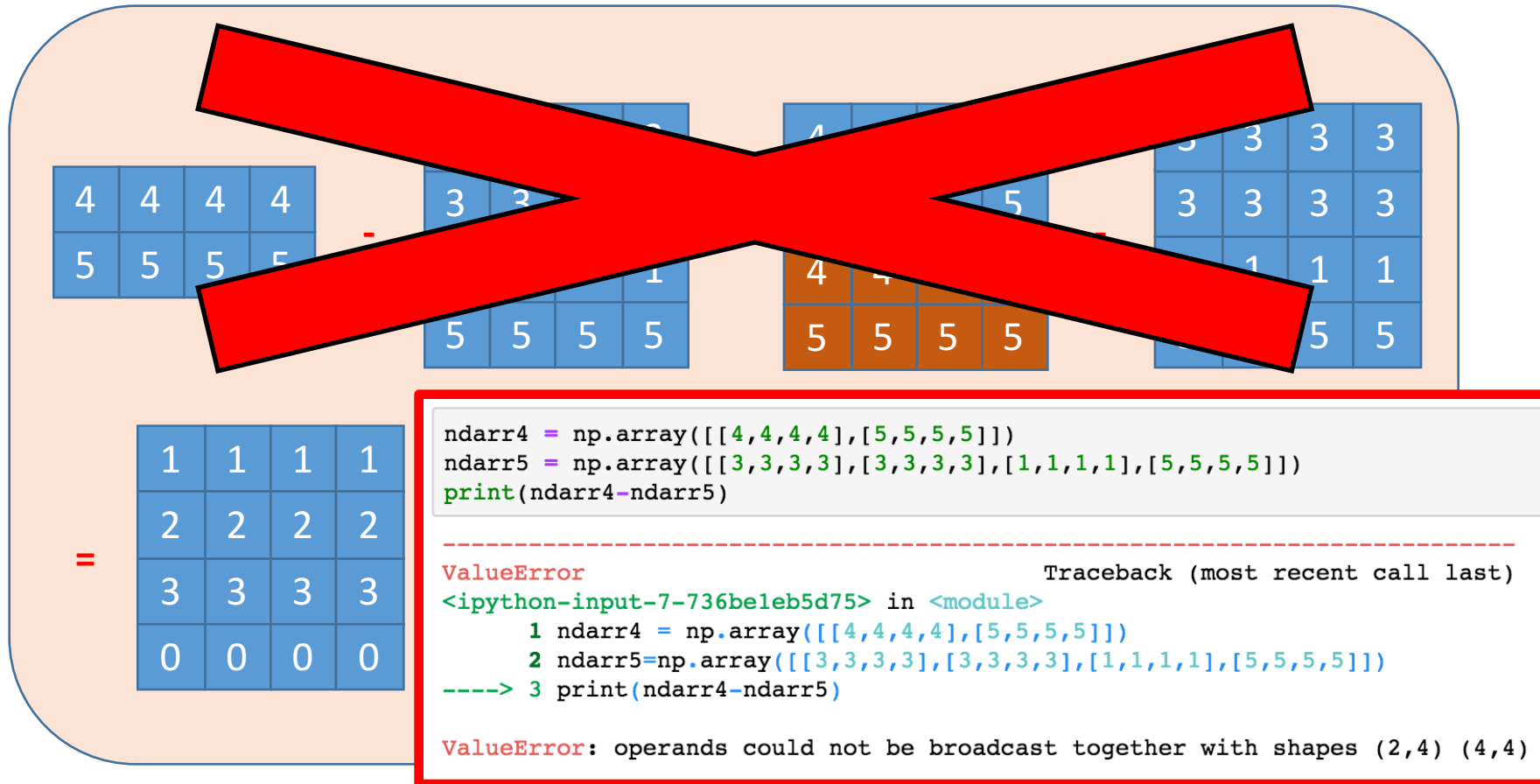
# NumPy: Broadcasting

4	4	4	4	-	3	3	3	3	=	4	4	4	4	-	3	3	3	3
5	5	5	5		3	3	3	3		5	5	5	5		3	3	3	3
					1	1	1	1		4	4	4	4		1	1	1	1
					5	5	5	5		5	5	5	5		5	5	5	5

1	1	1	1
2	2	2	2
3	3	3	3
0	0	0	0

# NumPy: Broadcasting



The diagram illustrates a broadcasting failure in NumPy. It shows three 2x4 arrays being subtracted from a 4x4 array. A large red 'X' is placed over the arrays, indicating that the operation fails due to incompatible shapes. Below the arrays, a 4x4 result array is shown with values [1, 2, 3, 0] in each column. To the right, a code block shows the Python code and the resulting ValueError.

```
ndarr4 = np.array([[4,4,4,4],[5,5,5,5]])
ndarr5 = np.array([[3,3,3,3],[3,3,3,3],[1,1,1,1],[5,5,5,5]])
print(ndarr4-ndarr5)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-7-736be1eb5d75> in <module>
      1 ndarr4 = np.array([[4,4,4,4],[5,5,5,5]])
      2 ndarr5=np.array([[3,3,3,3],[3,3,3,3],[1,1,1,1],[5,5,5,5]])
----> 3 print(ndarr4-ndarr5)
```

```
ValueError: operands could not be broadcast together with shapes (2,4) (4,4)
```

# NumPy: Broadcasting

## Примеры:

A	(4d array):	10 x 2 x 4 x 20
B	(5d array):	30 x 1 x 2 x 4 x 1
Result	(5d array):	30 x 10 x 2 x 4 x 20

---

A	(4d array):	10 x 2 x 4 x 20
B	(1d array):	5
Result	( ) :	ERROR

---

A	(3d array):	7 x 1 x 20
B	(2d array):	5 x 20
Result	(3d array):	7 x 5 x 20

# NumPy: Broadcasting

## Примеры:



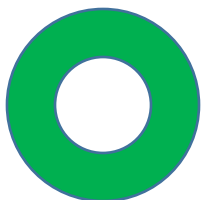
```
A      (4d array):      10 x 2 x 4 x 20
B      (5d array):    30 x 1  x 2 x 4 x 1
Result (5d array):    30 x 10 x 2 x 4 x 20
```

---



```
A      (4d array):      10 x 2 x 4 x 20
B      (1d array):      5
Result ( )              : ERROR
```

---



```
A      (3d array):      7 x 1 x 20
B      (2d array):      5 x 20
Result (3d array):      7 x 5 x 20
```



# **NumPy: Функции & Модули**

Data pre-processing and visualization

# Функции NumPy

Функции:	
<code>numpy.zeros()</code>	<code>numpy.multiply()</code>
<code>numpy.ones()</code>	<code>numpy.sqrt()</code>
<code>numpy.full()</code>	<code>numpy.dot()</code>
<code>numpy.eye()</code>	<code>numpy.sum()</code>
<code>numpy.arange()</code>	<code>numpy.zeros_like()</code>
<code>numpy.add()</code>	<code>numpy.empty_like()</code>
<code>numpy.subtract()</code>	<code>numpy.reshape()</code>
<code>numpy.sin()</code>	<code>numpy.isnan()</code>



# NumPy: Модуль **random**

## Функции модуля `numpy.random`:

`random.sample()`

`random.seed()`

`random.random()`

`random.shuffle()`

`random.randint()`

`random.permutation()`

`random.random_integers()`

`random.choice()`

`random.uniform()`



# **NumPy: Логическое индексирование**

Data pre-processing and visualization

# NumPy: Логическое индексирование

- Как мы можем отфильтровать массивы, чтобы не было чисел, меньших или равных 1?

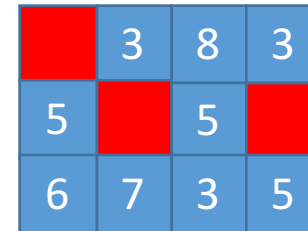
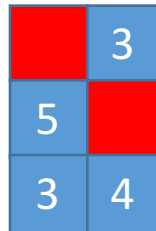
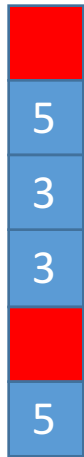
1
5
3
3
1
5

1	3
5	1
3	4

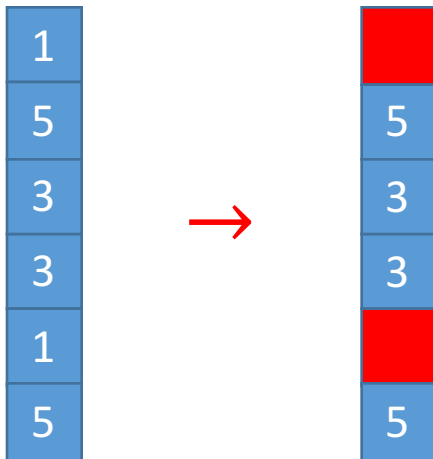
0	3	8	3
5	1	5	1
6	7	3	5

# NumPy: Логическое индексирование

- Как мы можем отфильтровать массивы, чтобы не было чисел, меньших или равных 1?



# NumPy: Логическое индексирование



```
arr1 = np.array([[1,5,3,3,1,5]])  
print(arr1)  
bool_arr1 = arr1 > 1  
print(bool_arr1)  
filt_arr1 = arr1[bool_arr1]  
print(filt_arr1)
```

```
[[1 5 3 3 1 5]]  
[[False True  True  True False  True]]  
[5 3 3 5]
```

# NumPy: Логическое индексирование

1	3
5	1
3	4




	3
5	
3	4

```
arr2 = np.array([[1,3],[5,1],[3,4]])  
print(arr2)  
bool_arr2 = arr2 > 1  
print(bool_arr2)  
filt_arr2 = arr2[bool_arr2]  
print(filt_arr2)
```

```
[[1 3]  
 [5 1]  
 [3 4]]  
[[False  True]  
 [ True False]  
 [ True  True]]  
[3 5 3 4]
```

# NumPy: Логическое индексирование

0	3	8	3
5	1	5	1
6	7	3	5



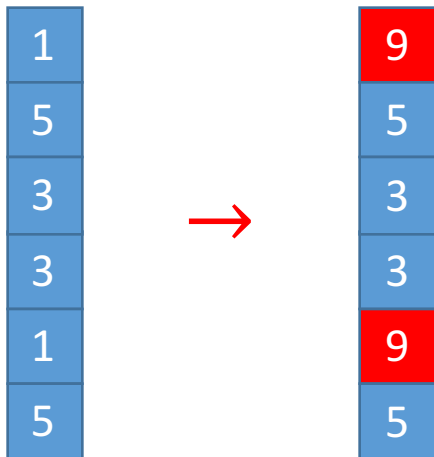
	3	8	3
5		5	
6	7	3	5

```
arr3 = np.array([[0,3,8,3],[5,1,5,1],[6,7,3,5]])  
print(arr3)  
bool_arr3 = arr3 > 1  
print(bool_arr3)  
filt_arr3 = arr3[bool_arr3]  
print(filt_arr3)
```

```
[[0 3 8 3]  
 [5 1 5 1]  
 [6 7 3 5]]  
[[False  True  True  True]  
 [ True False  True False]  
 [ True  True  True  True]]  
[3 8 3 5 5 6 7 3 5]
```

# NumPy: Логическое индексирование для назначения

- Как заменить все элементы массива, которые меньше 1, числом 9?



```
arr1 = np.array([[1,5,3,3,1,5]])
print(arr1)
bool_arr1 = arr1 <= 1
print(bool_arr1)
arr1[bool_arr1] = 9
print(arr1)
```

```
[[1 5 3 3 1 5]]
[[ True False False False  True False]]
[[9 5 3 3 9 5]]
```