NumPy

Голубцов В. А.

Зачем?

Массивы (вектора)

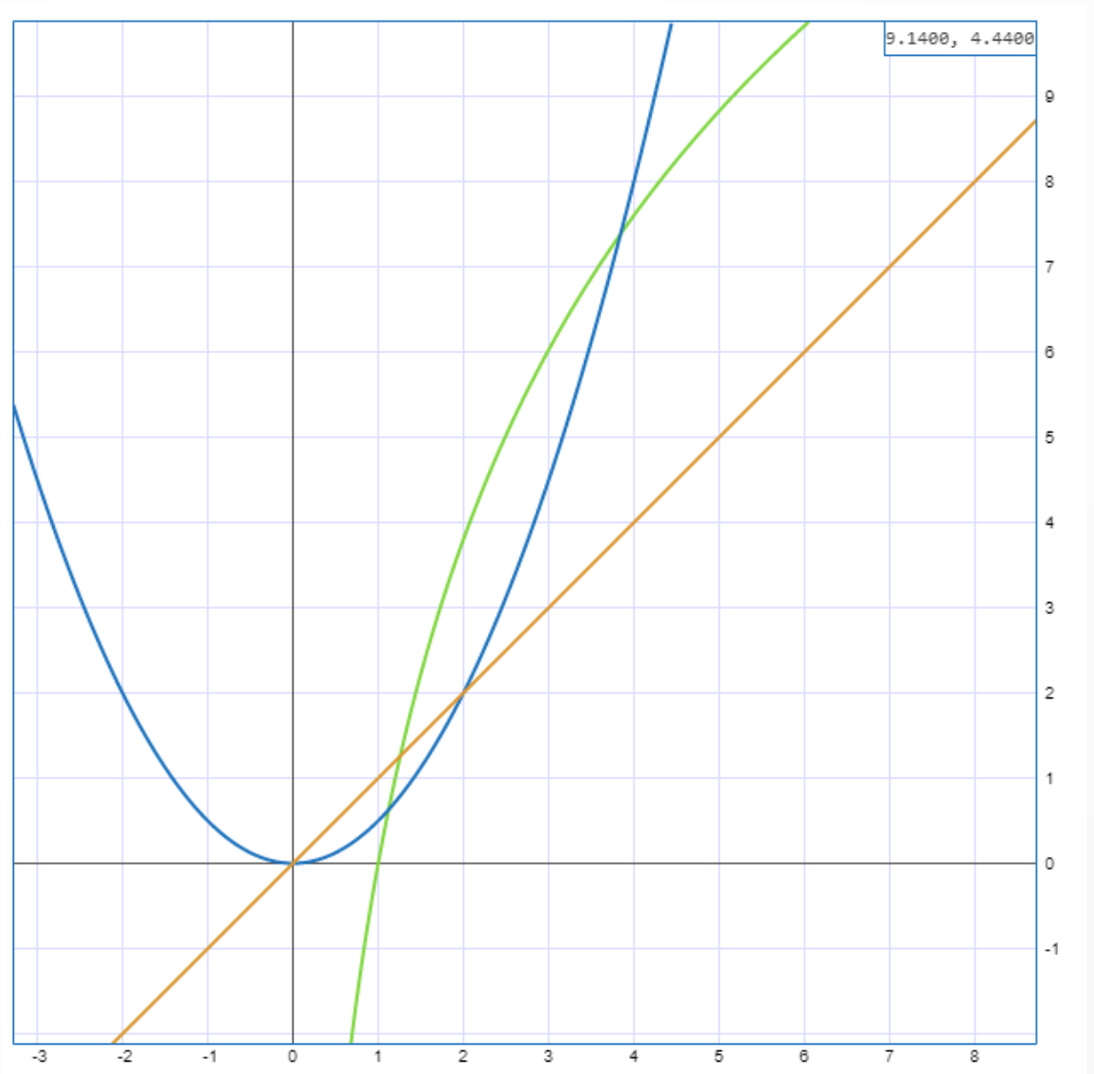
Списки

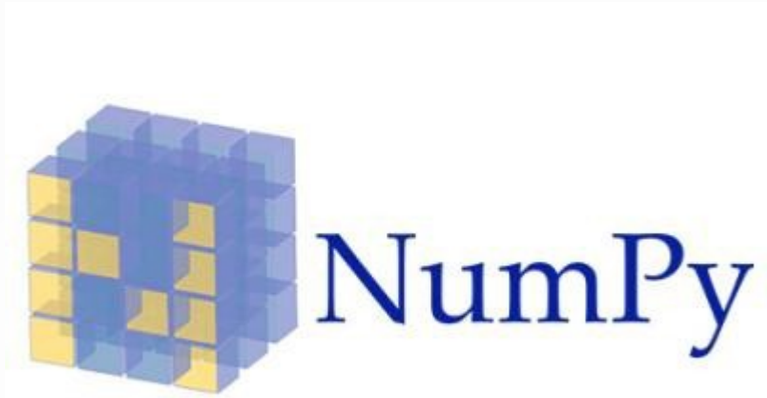
[A, B, C]

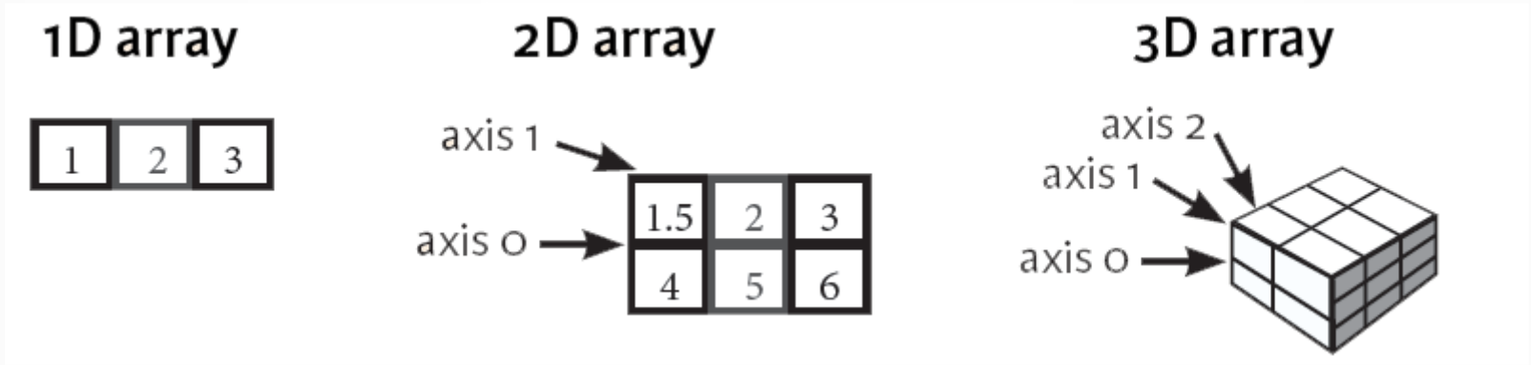
https://wiki.python.org/moin/TimeComplexity

O(1) < O(log(n)) < O(k) <= O(n) < O(n*log(n)) < O(n^2)

- y = log(1.2, x)
- y = 0.5*x*x
- y = x




NumPy


1D array
1
2
3
2D array
axis 1
axis 0
1.5
2
3
4
5
6
3D array
axis 2
axis 1
axis 0

Пакетный менеджер Pip

- В Python 3.4+ уже есть
- >>> pip install NumPy

Библиотеки для Windows от Gohlke - https://www.lfd.uci.edu/~gohlke/pythonlibs/

```
>>> import numpy as np
```

np.array()

- np.array.ndim - число измерений ("оси") массива.
- np.array.shape — кортеж размеров массива
- np.array.size - количество элементов массива
- np.array.dtype - объект, описывающий тип элементов массива.
- np.array.itemsize - размер каждого элемента массива в байтах.
- np.array.data - буфер, содержащий фактические элементы массива. (Обычно не используется)

```
>>> import numpy as np
>>>
>>>a = np.array([1, 2, 3])
```

```
>>> import numpy as np
>>>
>>>a = np.array([1, 2, 3])
>>>print(a)

[1, 2, 3]
```

```
>>> import numpy as np
>>>
>>>a = np.array([1, 2, 3])
>>>b = np.array([[1.5, 2, 3], [4, 5, 6]])
>>>print b

[[ 1.5  2.   3. ]
 [ 4.   5.   6. ]]
```

```
>>> import numpy as np
>>>
>>>a = np.array([1, 2, 3])
>>>b = np.array([[1.5, 2, 3], [4, 5, 6]], dtype=np.complex)
>>>print b

[[ 1.5+0.j  2.0+0.j  3.0+0.j]
 [ 4.0+0.j  5.0+0.j  6.0+0.j]]
```

Почему?

```
>>> import numpy as np
>>>
>>>b = np.array([[1.5, 2, 3], [4, 5, 6, 7]])
>>>print b

[[ 1.5+0.j  2.0+0.j  3.0+0.j]
 [ 4.0+0.j  5.0+0.j  6.0+0.j]]
```

```
>>> import numpy as np
>>>
>>>b = np.array([[1.5, 2, 3], [4, 5, 6, 7]], dtype=np.complex)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a float is required
```

Больше способов инициализации!!!11!

```
>>> np.zeros((3, 5)  )
[[ 0.,  0.,  0.,  0.,  0.],
     [ 0.,  0.,  0.,  0.,  0.],
     [ 0.,  0.,  0.,  0.,  0.]]
```

```
>>> np.ones((2, 2, 2))
[[[ 1.,  1.],
  [ 1.,  1.]],
 [[ 1.,  1.],
  [ 1.,  1.]]]
```

```
np.eye(5)
[[ 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.]]
```

```
>>> r = np.random.random((3,3))
>>> m = r.mean()
```

```
>>> r = np.random.random((3,3))
>>> m = r.mean()
>>> r2 = r + 1

>>> print ( r )
[[ 0.37007418  0.75402447  0.48608901]
 [ 0.00485012  0.34278094  0.21657563]
 [ 0.85417297  0.0667909   0.60559997]]

>>> print ( r2 )
[[ 1.37007418,  1.75402447,  1.48608901],
 [ 1.00485012,  1.34278094,  1.21657563],
 [ 1.85417297,  1.0667909 ,  1.60559997]])
```

```
>>> r = np.random.random((3,3))
>>> m = r.mean()
>>> r2 = r + 1
>>> C = np.cos((r + r) ** 3)
>>> print(C)
[[ 0.91891799, -0.9588054 ,  0.60674687],
 [ 1.        ,  0.94853768,  0.99669959],
 [ 0.26993558,  0.99999716, -0.2045857 ]]
```

```
>>> r = np.random.random((3,3))
>>> m = r.mean()
>>> r2 = r + 1
>>> C = np.cos((r + r) ** 3)

>>> print(C)
[[ 0.91891799, -0.9588054 ,  0.60674687],
 [ 1.        ,  0.94853768,  0.99669959],
 [ 0.26993558,  0.99999716, -0.2045857 ]]

>>> print(C<0)
[[ False, True ,  False],
 [ False,  False,  False],
 [ False,  False, True]]
```

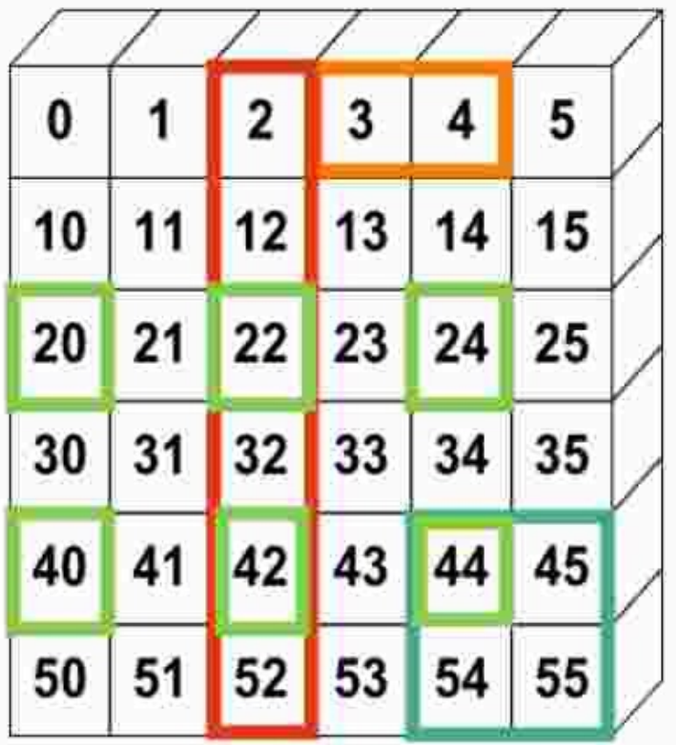
Индексы и срезы

```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



```
>>> p = np.random.random((5))

>>print(p)
[ 0.27551045  0.78631163  0.12102608  0.15976966  0.99517535]

>>> print(p[2])
0.12102608488760458
```

```
>>> p = np.random.random((5))

>>print(p)
[ 0.27551045  0.78631163  0.12102608  0.15976966  0.99517535]

>>> print(p[2:])
[ 0.12102608,  0.15976966,  0.99517535]
```

```
>>> p = np.random.random((5))

>>print(p)
[ 0.27551045  0.78631163  0.12102608  0.15976966  0.99517535]

>>> print(p[2:4])
[ 0.12102608,  0.15976966]
```

```
>>> p = np.random.random((5))

>>print(p)
[ 0.27551045  0.78631163  0.12102608  0.15976966  0.99517535]

>>> print(p[1:-2])
[ 0.78631163,  0.12102608]
```

```
>>> p = np.random.random((5))

>>print(p)
[ 0.27551045  0.78631163  0.12102608  0.15976966  0.99517535]

>>> print(p[1::2])
[ 0.78631163,  0.15976966]
```

```
>>> p = np.random.random((5, 5))
>>> print p
[[ 0.89357301  0.48482166  0.91910089  0.64431201  0.5046139 ]
 [ 0.31306256  0.5833581   0.33455025  0.01313503  0.34719377]
 [ 0.88634324  0.28359036  0.55375488  0.63467417  0.95944837]
 [ 0.00769992  0.01200226  0.72258102  0.50323211  0.99970244]
 [ 0.74205999  0.8855628   0.08830985  0.00869231  0.26656741]]

>>>print(p[1,1])
```

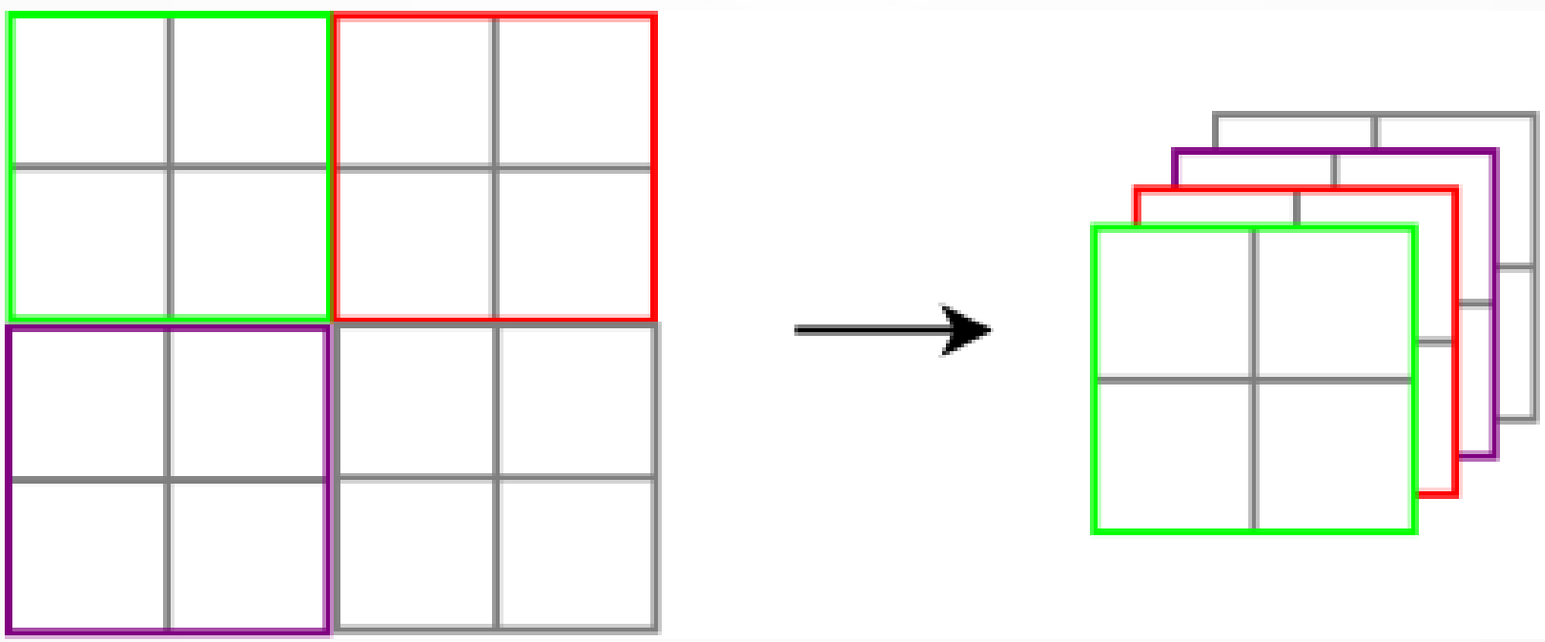
```
>>> p = np.random.random((5, 5))
>>> print p
[[ 0.89357301  0.48482166  0.91910089  0.64431201  0.5046139 ]
 [ 0.31306256  0.5833581   0.33455025  0.01313503  0.34719377]
 [ 0.88634324  0.28359036  0.55375488  0.63467417  0.95944837]
 [ 0.00769992  0.01200226  0.72258102  0.50323211  0.99970244]
 [ 0.74205999  0.8855628   0.08830985  0.00869231  0.26656741]]

>>>print(p[1::2,1:3])
[[ 0.5833581   0.33455025]
 [ 0.01200226  0.72258102]]
```

Задача

```
>>> p = np.random.random((5, 5))
>>> print p
[[ 0.89357301  0.48482166  0.91910089  0.64431201  0.5046139 ]
 [ 0.31306256  0.5833581   0.33455025  0.01313503  0.34719377]
 [ 0.88634324  0.28359036  0.55375488  0.63467417  0.95944837]
 [ 0.00769992  0.01200226  0.72258102  0.50323211  0.99970244]
 [ 0.74205999  0.8855628   0.08830985  0.00869231  0.26656741]]
```

Манипуляции с формой



```
>>> m_4x4 = np.random.random((4, 4))
>>> print(m_4x4)
[[ 0.73577237  0.37678939  0.98588422  0.11154924]
 [ 0.48292762  0.07229587  0.26343597  0.09408413]
 [ 0.76365441  0.34374507  0.62411017  0.81866665]
 [ 0.25025241  0.08725163  0.92850361  0.75520065]]
```

```
>>> m_4x4 = np.random.random((4, 4))
>>> print m_4x4.shape
(4, 4)
```

```
>>> m_4x4 = np.random.random((4, 4))
>>> m_4x4_tr = m_4x4.transpose()

>>> print(m_4x4)
[[ 0.73577237  0.37678939  0.98588422  0.11154924]
 [ 0.48292762  0.07229587  0.26343597  0.09408413]
 [ 0.76365441  0.34374507  0.62411017  0.81866665]
 [ 0.25025241  0.08725163  0.92850361  0.75520065]]

>>> print(m_4x4_tr)
[[ 0.73577237  0.48292762  0.76365441  0.25025241]
 [ 0.37678939  0.07229587  0.34374507  0.08725163]
 [ 0.98588422  0.26343597  0.62411017  0.92850361]
 [ 0.11154924  0.09408413  0.81866665  0.75520065]]
```

```
>>> m_4x4 = np.random.random((4, 4))
>>> m_4x4_reshaped = m_4x4.reshape(2,2,4)

>>> print(m_4x4)
[[ 0.73577237  0.37678939  0.98588422  0.11154924]
 [ 0.48292762  0.07229587  0.26343597  0.09408413]
 [ 0.76365441  0.34374507  0.62411017  0.81866665]
 [ 0.25025241  0.08725163  0.92850361  0.75520065]]

>>> print(m_4x4_reshaped)
[[[ 0.73577237  0.37678939  0.98588422  0.11154924]
  [ 0.48292762  0.07229587  0.26343597  0.09408413]]

 [[ 0.76365441  0.34374507  0.62411017  0.81866665]
  [ 0.25025241  0.08725163  0.92850361  0.75520065]]]

>>>print(m_4x4_reshaped[1,1,2])
0.928503612684
```

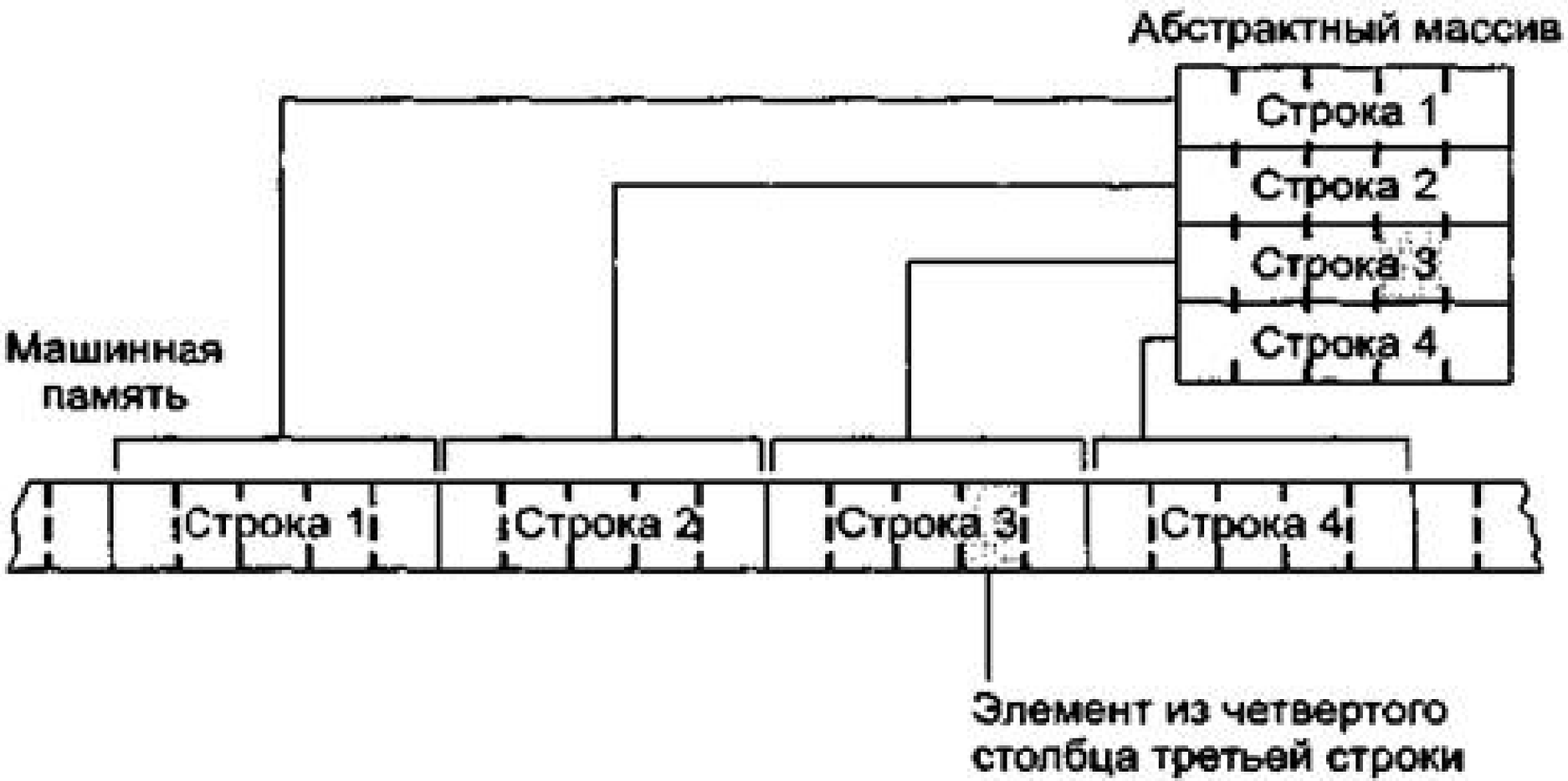
```
>>> m_4x4 = np.random.random((4, 4))
>>> m_4x4_reshaped = m_4x4.reshape(2,2,4)

>>> print(m_4x4_reshaped)
[[[ 0.73577237  0.37678939  0.98588422  0.11154924]
  [ 0.48292762  0.07229587  0.26343597  0.09408413]]

 [[ 0.76365441  0.34374507  0.62411017  0.81866665]
  [ 0.25025241  0.08725163  0.92850361  0.75520065]]]

>>> print m_4x4_reshaped.ravel()
[ 0.73577237  0.37678939  0.98588422  0.11154924  0.48292762  0.07229587
  0.26343597  0.09408413  0.76365441  0.34374507  0.62411017  0.81866665
  0.25025241  0.08725163  0.92850361  0.75520065]
```

Память компьютера



Итирирование

- >>> a = np.array([[1, 2], [3, 4]])
- >>>for row in a:
- >>> print(row)
- [1 2]
- [3 4]

Сложение массивов

- >>> a = np.array([[1, 2], [3, 4]])
- >>> b = np.array([[5, 6], [7, 8]])
- >>> np.vstack((a, b))
- array([[1, 2],
- [3, 4],
- [5, 6],
- [7, 8]])
- >>> np.hstack((a, b))
- array([[1, 2, 5, 6],
- [3, 4, 7, 8]])

Сложение массивов

- >>> np.column_stack((a, b))
- array([[1, 2, 5, 6],
- [3, 4, 7, 8]])

- >>> np.row_stack((a, b))
- array([[1, 2],
- [3, 4],
- [5, 6],
- [7, 8]])

Разбиения массивов

- >>> a = np.arange(12).reshape((2, 6))
- >>> a
- array([[0, 1, 2, 3, 4, 5],
- [6, 7, 8, 9, 10, 11]])
- >>> np.hsplit(a, 3) # Разбить на 3 части
- [array([[0, 1], [6, 7]]),
- array([[2, 3], [8, 9]]),
- array([[4, 5], [10, 11]])]
- >>> np.hsplit(a, (3, 4)) # Разрезать a после третьего и четвёртого столбца
- [array([[0, 1, 2], [6, 7, 8]]),
- array([[3], [9]]),
- array([[4, 5], [10, 11]])]

Представление или поверхностная копия

- >>> c = np.array([1,2])
- >>> a = c
- >>> a[0] = 7
- >>> print c
- [7 2]
- >>> a.base is c
- True
- >>> a is c
- True

Глубокие копии

- >>> c = np.array([1,2])
- >>> a = c.copy()
- >>> a[0] = 7
- >>> print a
- [7 2]
- >>> print c
- [1 2]
- >>> a.base is c
- False
- >>> a is c
- False

Вывод информации

- np.set_printoptions(threshold=np.nan)
- precision : количество отображаемых цифр после запятой (по умолчанию 8).
- threshold : количество элементов в массиве, вызывающее обрезание элементов (по умолчанию 1000).
- edgeitems : количество элементов в начале и в конце каждой размерности массива (по умолчанию 3).
- linewidth : количество символов в строке, после которых осуществляется перенос (по умолчанию 75).
- suppress : если True, не печатает маленькие значения в scientific notation (по умолчанию False).
- nanstr : строковое представление NaN (по умолчанию 'nan').
- infstr : строковое представление inf (по умолчанию 'inf').
- formatter : позволяет более тонко управлять печатью массивов.

https://docs.scipy.org/doc/numpy/reference/generated/numpy.set_printoptions.html

NumPy

100 задач по NumPy
-https://pythonworld.ru/numpy/100-exercises.html