У каждого n-мерного массива есть форма. Например, форма матрицы (двумерного массива) из 3 строк и 5 колонок — это (3, 5), форма вектора (вертикального массива) из 10 элементов — это (10, 1). В общем случае форма n-мерного массива — это кортеж (последовательность чисел) из n элементов (*n*1,…,*n**k*), где *n**i* — количество компонент массива в *i*-м измерении.

Форма массива в NumPy хранится в атрибуте shape. Например,

>>> a = np.array([[1,2,3], [4,5,6]]) # создаём массив

>>> print(a) # смотрим на массив

[[1 2 3]

[4 5 6]]

>>> a.shape # смотрим на форму массива

(2, 3)

Для n-мерных массивов соответствующим образом определены основные арифметические и логические операции (+ - \* / // % \*\* < <= == >= > != выполняются поэлементно; если вторым аргументом бинарной операции указано число (скаляр), то он преобразуется в массив той же формы, что и первый аргумент). Также для n-мерных массивов реализованы все основные методы, которые могут понадобиться при работе (дальше мы покажем те из них, которые будут особенно нужны при решении заданий курса).

**Как создать массив:**

* array(object) — n-мерный массив из любой (возможно, вложенной) последовательности,
* eye(N, M=N, k=0) -- двумерный массив с N строками с единицами на диагонали и нулями во всех остальных позициях. Число столбцов M по умолчанию равно N, k — сдвиг диагонали (0 для основной диагонали, положительные числа для верхних диагоналей и отрицательные для нижних),
* zeros(shape) -- новый массив указанной формы, заполненный нулями,
* ones(shape) -- новый массив указанной формы, заполненный единицами,
* full(shape, fill\_value) -- новый массив указанной формы, заполненный fill\_value.

>>> print(np.eye(5, 3, k=-1))

[[ 0. 0. 0.]

[ 1. 0. 0.]

[ 0. 1. 0.]

[ 0. 0. 1.]

[ 0. 0. 0.]]

>>> print(np.array([[1,2,3], [4,5,6]])) # матрица формы (2, 3) из последовательности последовательностей

[[1 2 3]

[4 5 6]]

**Как получить доступ к элементу массива или выбрать кусок массива:**

* в NumPy используется стандартное  выделение подмножеств: a[start:stop:step] выбирает элементы из a с индексами от start до stop с шагом step, т.е. a[start], a[start + step], a[start + 2\*step] и так далее, пока индексы меньше stop; при step = 1 этот параметр можно опустить, оставив start:stop, при start, равном минимальному индексу в массиве, или stop, равному максимальному индексу, эти индексы также можно опустить: это будет выглядеть как a[:];
* отрицательные индексы;
* поддерживается синтаксис многоточия: a[..., 1] выбирает элементы с любым индексом в первом измерении и с индексом, равным 1, во втором измерении:  a[:, 1] оказывается эквивалентно a[..., 1];
* можно указать индексы сразу в нескольких измерениях, указав их через запятую в квадратных скобках (как a[..., 1] в примере выше).

## **Основные методы ndarray**

Для работы с многомерными массивами в NumPy реализованы самые часто требующиеся операции. Некоторые из них (которые особенно часто будут нужны в нашем курсе) мы сейчас покажем.  
  
**Форма массива**

* a.flatten() — превращает массив в одномерный.
* a.T или a.transpose(\*axes) — транспонирование (или смена порядка осей в случае, когда размерность массива больше двух).
* a.reshape(shape) — смена формы массива. Массив "распрямляется" и построчно заполняется в новую форму.

>>> import random

>>> w = np.array(random.sample(range(1000), 12)) # одномерный массив из 12 случайных чисел от 1 до 1000

>>> w = w.reshape((2,2,3)) # превратим w в трёхмерную матрицу

>>> print(w)

[[[536 986 744]

  [543 248 544]]

[[837 235 415]

  [377 141 751]]]

>>> print(w.transpose(0,2,1))

[[[536 543]

  [986 248]

  [744 544]]

[[837 377]

  [235 141]

  [415 751]]]

## **Основные методы ndarray**

**Базовые статистики**

* a.min(axis=None), a.max(axis=None), a.mean(axis=None), a.std(axis=None) — минимум, максимум, среднее арифметическое и стандартное отклонение вдоль указанной оси. По умолчанию ось не указана и статистика считается по всему массиву. a.argmin(axis=None), a.argmax(axis=None) — индексы минимального и максимального элемента. Пример:

>>> print(v)

[[1 2 3 4]

[1 2 3 4]

[1 2 3 4]]

>>> print(v.mean(axis=0)) # вдоль столбцов

[ 1. 2. 3. 4.]

>>> print(v.mean(axis=1)) # вдоль строк

[ 2.5 2.5 2.5]

>>> print(v.mean(axis=None)) # вдоль всего массива

2.5

Чтобы лучше понять, почему говорят «усреднение вдоль оси» — можно нарисовать эту матрицу на бумажке и прямыми линиями соединить те элементы, которые сливаются в один при усреднении. Чтобы было совсем понятно — можно ещё добавить координатные оси, чтобы каждый элемент *w**i**j* оказался над точкой (*i*,*j*).

* a.sum(axis=None), a.prod(axis=None) — сумма и произведение всех элементов вдоль указанной оси. a.cumsum(axis=None), a.cumprod(axis=None) — частичные суммы и произведения (для (*a*1,⋯,*a**n*) вектор частичных сумм — это (*a*1,*a*1+*a*2,⋯,*a*1+⋯+*a**n*)).

**Линейная алгебра**  
Пакет numpy.linalg содержит большую часть стандартных операций и разложений матриц. Некоторые самые популярные функции вынесены в корень пакета NumPy.

* a.dot(b) — матричное произведение двух массивов (размерности должны быть согласованы),
* linalg.matrix\_power(M, n) — возведение матрицы M в степень n,
* a.T — транспонирование
* linalg.norm(a, ord=None) — норма матрицы a, по умолчанию норма Фробениуса для матриц и L2-норма для векторов; подробное описание возможных норм — в [справке](http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.norm.html" \l "numpy.linalg.norm),
* linalg.inv(a) — матрица, обратная к a (если a необратима, выбрасывается LinAlgError; псевдообратная считается через linalg.pinv(a))

>>> a = w.dot([1,2,3])

>>> print(a)

[[4740 2671]

[2552 2912]]

>>> ainv = np.linalg.inv(a)

>>> print(a.dot(ainv))

[[ 1.00000000e+00 0.00000000e+00]

[ -2.22044605e-16 1.00000000e+00]]

Подробные описания с указанием полного списка аргументов, а также описания всех остальных функций находятся на сайте проекта [NumPy](http://docs.scipy.org/doc/numpy/reference/index.html).

## **Как считать данные из файла:**

>>> sbux = np.loadtxt("sbux.csv", usecols=(0,1,4), skiprows=1, delimiter=",",

                      dtype={'names': ('date', 'open', 'close'),

                            'formats': ('datetime64[D]', 'f4', 'f4')})

>>> print(sbux[0:4])

[(datetime.date(2015, 9, 1), 53.0, 57.2599983215332)

(datetime.date(2015, 8, 3), 58.619998931884766, 54.709999084472656)

(datetime.date(2015, 7, 1), 53.86000061035156, 57.93000030517578)

(datetime.date(2015, 6, 1), 51.959999084472656, 53.619998931884766)]

Здесь использованы не все параметры функции loadtxt (полный их список можно посмотреть в [справке](http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html)). Разберём имеющиеся, так как они являются наиболее часто встречающимися.

"sbux.csv" — имя файла (или сюда же **можно передать объект файла**, такой пример вы увидите в следующей задаче урока), из которого считываются данные.

usecols — список колонок, которые нужно использовать. Если параметр не указан, считываются все колонки.

skiprows — количество рядов в начале файла, которые нужно пропустить. В нашем случае пропущен ряд заголовков. По умолчанию (если значение параметра не указано явно) skiprows = 0.

delimiter — разделитель столбцов в одной строке, в csv-файлах это запятая, по умолчанию разделителем является любой пробел (в том числе — знак табуляции).

dtype — словарь из названий колонок (переменных) и типов хранящихся в них значений. NumPy использует свою собственную систему типов, и названия именно этих типов нужно указать. По умолчанию функция попытается самостоятельно угадать, какому типу принадлежат подаваемые на вход значения.

Нфтву

np.hstack((array1, array2, ...)) # склеивает по строкам массивы, являющиеся компонентами кортежа, поданного на вход; массивы должны совпадать по всем измерениям, кроме второго np.ones\_like(array) # создаёт массив, состоящий из единиц, идентичный по форме массиву array "delim".join(array) # возвращает строку, состоящую из элементов array, разделённых символами "delim" map(str, array) # применяет функцию str к каждому элементу array