Лекция №11.
Тема: **Многомерные массивы.**
Библиотека NumPy. Тип данных ndarray. Объявление и инициализация переменных типа ndarray. Операции над массивами. Стандартные методы по работе с массивами.
Ключевые слова: массив, ось, ранг, размер, индекс, NumPy, ndarray, операции над массивами, методы по работе с массивами.
Keywords: array, axe, rank, shape, index, NumPy, ndarray, ndarray operators, ndarray methods.

1 Библиотека NumPy. Тип данных ndarray

В языке Python для работы с массивами используется специальная библиотека NumPy (Numeric Python). NumPy представляет комплекс модулей, которые обеспяечивают эффективную обработку многомерных массивов, а также содержат функции, которые реализуют алгоритмы генерации случайных чисел, линейной алгебры и преобразование Фурье. NumPy распространяется под BSD-лицензией (см. https://en.wikipedia.org/wiki/BSD_licenses) и доступна на web-ресурсе SourceForge (см. http://sourceforge.net/projects/numpy/files). На официальном сайте NumPy доступна документация высокого качества (см. http://docs.scipy.org/doc/numpy/reference/index.html)

Для работы с NumPy необходимо в программе выполнить подключение библиотеки с помощью оператора import

```
>>> import numpy
```

В документации по библиотеке NumPy часто используют вместо полного имени библиотеки numpy псевдоним np, который задается с помощью оператора as

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

Основным типом данных в NumPy является ndarray (N-dimensional array) – N-мерный массив (см. http://docs.scipy.org/doc/numpy/reference/arrays.html). Особенности типа данных ndarray:

1. ndarray представляет собой N-мерную последовательность фиксированного числа элементов одного типа данных, в которой для доступа к отдельным элементам используется N целых чисел (индексов).
2. Для определения типа элементов ndarray в NumPy содержится большое количество типов данных (см. http://docs.scipy.org/doc/numpy/user/basics.types.html). Основные типы данных представлены в табл. 1.

Таблица 1 – Основные типы данных NumPy

Тип данных	Размер
Целочисленные типы данных	
bool_	1 байт
int_	4 байт / 8 байт
Вещественные типы данных	
float_	8 байт

complex_	16 байт
Строковый тип данных	
str_	

3. В памяти объект типа ndarray представляется последовательно.
4. Объект типа ndarray может быть создан с помощью функции array

```
array(...)
    array(object,      dtype=None,      copy=True,      order=None,
          subok=False, ndmin=0)
```

Описание параметров функции array может быть получено с помощью интерактивной подсказки в IDLE

```
>>> help(numpy.array)
```

Примеры создания N-мерных массивов:

```
>>> # Одномерный массив целых чисел на базе списка.
>>> x = np.array([1, 2, 4, 5])
>>> x
array([1, 2, 4, 5])
>>> type(x)
<class 'numpy.ndarray'>
```

```
>>> # Двухмерный массив целых чисел на базе вложенных списков.
>>> x = np.array([[1, 2, 3], [1, 2, 3]])
>>> x
array([[1, 2, 3],
       [1, 2, 3]])
```

5. Объекты типа ndarray описываются следующими свойствами:

- ndarray.ndim – ранг массива (количество осей массива или размерность);
- ndarray.shape – размер массива (набор целых чисел, описывающий размер массива по каждой оси);
- ndarray.size – количество элементов массива;
- ndarray.dtype – тип данных элементов массива;
- ndarray.itemsize – размер каждого элемента массива в байтах;
- ndarray.data – буфер, содержащий актуальные элементы массива.

2 Объявление и инициализация переменных типа ndarray

Объявление и инициализация переменной типа ndarray может быть выполнена несколькими способами (см. http://docs.scipy.org/doc/numpy/reference/routines.array-creation.html#routines-array-creation):

1. Создание массива на базе существующей последовательности, используя функцию array(). Функция array() трансформирует вложенную

последовательность в многомерный массив, тип элементов которого зависит от типа элементов исходной последовательности. Пример

```
>>> # Двумерный массив целых чисел на базе списка.
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

```
>>> # Двумерный массив строк на базе списка.
np.array([[1, 2], [3, 4]], np.str_)
array([['1', '2'],
       ['3', '4']],
      dtype='<U1')
```

2. Создание n-мерного массива заданной формы с помощью функции zeros(). Функция zeros() инициализирует элементы массива нулем заданного типа данных.

```
zeros(...) → ndarray
     zeros(shape, dtype= float64, order='C')
```

где shape – размер массива,
dtype – тип данных элементов массива, по умолчанию – numpy.float64,
order – способ хранения элементов массива в памяти (построчно – 'C', постолбцово – 'F').

Примеры:

```
>>> # Одномерный массив вещественных чисел.
>>> np.zeros(10)
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
>>> # Двумерный массив вещественных чисел.
>>> np.zeros((2, 2))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

```
>>> # Двумерный массив целых чисел.
>>> np.zeros((2, 2), np.int_)
array([[0, 0],
       [0, 0]])
```

```
>>> # Двумерный массив. Тип данных элементов - логический.
>>> np.zeros((2, 2), np.bool_)
array([[False, False],
       [False, False]], dtype=bool)
```

```
>>> # Двумерный массив. Тип данных элементов - строка.
>>> np.zeros((2, 2), np.str_)
array([['', ''],
       ['', '']],
      dtype='<U1')
```

3. Создание n-мерного массива заданной формы с помощью функции ones(). Функция ones() инициализирует элементы массива единицей заданного типа данных.

```
ones(...) → ndarray
     ones(shape, dtype= float64, order='C')
```

Примеры:

```
>>> # Одномерный массив вещественных чисел.
>>> np.ones(10)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
>>> # Двумерный массив вещественных чисел.
>>> np.ones((2, 2))
array([[ 1.,  1.],
       [ 1.,  1.]])
```

```
>>> # Двумерный массив целых чисел.
>>> np.ones((2, 2), np.int_)
array([[1, 1],
       [1, 1]])
```

```
>>> # Двумерный массив. Тип данных элементов - логический.
>>> np.ones((2, 2), np.bool_)
array([[ True,  True],
       [ True,  True]], dtype=bool)
```

```
>>> # Двумерный массив. Тип данных элементов - строка.
>>> np.ones((2, 2), np.str_)
array([['1', '1'],
       ['1', '1']],
      dtype='<U1')
```

4. Создание n-мерного массива заданной формы с помощью функции empty(). Функция empty() не инициализирует элементы массива (элементы массива содержат «мусор»).

```
empty(...) → ndarray
     empty (shape, dtype= float64, order='C')
```

Примеры:

```
>>> # Одномерный массив вещественных чисел.
>>> np.empty(10)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
>>> # Двумерный массив вещественных чисел.
>>> np.empty((2, 2))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

3 Операции над элементами массива и над массивом в целом

Рассмотрим особенности операций над массивами в языке Python (см. http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html) (табл. 2).

Таблица 2 – Операции над массивами

Оператор	Описание	Пример

=	Оператор присваивания. Результат работы оператора – копия ссылки на объект.	`a = np.zeros((2, 2, 2))`
		`>>> # Создание копии ссылки.`
		`>>> b = a`
		`>>> # a и b ссылаются на один объект.`
		`>>> id(a) == id(b)`
		`True`

+	Оператор сложения. Арифметические операции над массивами выполняются поэлементно. Результат арифметических операций – новый массив в памяти.	`>>> # Сумма массивов.`
		`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`
		`>>> c = a + b`
		`>>> c`
		`array([[ 6, 8],`
		`[10, 12]])`

+=	Составной оператор сложения.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`
		`>>> a += b`
		`>>> a`
		`array([[ 6, 8],`
		`[10, 12]])`

-	Оператор вычитания.	`>>> # Вычитание массивов.`
		`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`
		`>>> c = a - b`
		`>>> c`
		`array([[-4, -4],`
		`[-4, -4]])`

-=	Составной оператор вычитания.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`
		`>>> a -= b`
		`>>> a`
		`array([[-4, -4],`
		`[-4, -4]])`

*	Оператор умножения	`>>> # Произведение массивов.`
		`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`

		`>>> c = a * b`
		`>>> c`
		`array([[ 5, 12],`
		`[21, 32]])`
		
		`>>> # Произведение массива на число.`
		`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> c = a * 3`
		`>>> c`
		`array([[ 3, 6],`
		`[ 9, 12]])`

*=	Составной оператор умножения.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = np.array([[5, 6], [7, 8]])`
		`>>> a *= b`
		`>>> a`
		`array([[ 5, 12],`
		`[21, 32]])`

**	Оператор возведения в степень	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> c = a ** 3`
		`>>> c`
		`array([[ 1, 8],`
		`[27, 64]], dtype=int32)`

**=	Составной оператор возведения в степень	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> a **= 3`
		`>>> a`
		`array([[ 1, 8],`
		`[27, 64]])`

[]	Оператор доступа к элементам массива (нумерация элементов массива для каждой оси начинается с нуля; допускаются отрицательные индексы). На каждую ось массива приходится один индекс. Индексы передаются в виде последовательности чисел, разделенных запятыми.	`>>> a = np.array([`
		`[ 0, 1, 2, 3],`
		`[10, 11, 12, 13],`
		`[20, 21, 22, 23],`
		`[30, 31, 32, 33],`
		`[40, 41, 42, 43]])`
		`>>> a[3, 2]`
		`32`

[1, 2, ...]	Оператор извлечения среза (см. примечание 1).	`>>> # Создание поверхностной копии массива.`
		`>>> a = np.array([[1,2], [3,4]])`
		`>>> b = a[...]`
		`>>> b`
		`array([[1, 2],`
		`[3, 4]])`

		`>>> # Замена второго столбца матрицы.`
		`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> a[0:2, 1] = [8, 7]`
		`>>> a`
		`array([[1, 8],`
		`[3, 7]])`
		`>>> # Выборка последней строки матрицы.`
		`>>> a = np.array([[1,2], [3,4]])`
		`>>> a[a.ndim - 1]`
		`array([3, 4])`

in	Оператор проверки на вхождение – возвращает True, если элемент присутствует в списке; в противном случае возвращает False.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> 5 in a`
		`False`
		`>>> 4 in a`
		`True`

not in	Оператор проверки на вхождение – возвращает True, если элемент не присутствует в списке; в противном случае возвращает False.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> 4 not in a`
		`False`
		`>>> 5 not in a`
		`True`

is	Оператор идентичности – возвращает True, если списки хранятся в памяти по одному и тому же адресу; в противном случае возвращает False.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = a`
		`>>> b is a`
		`True`

is not	Оператор идентичности – возвращает True, если списки хранятся в памяти по разным адресам; в противном случае возвращает False.	`>>> a = np.array([[1, 2], [3, 4]])`
		`>>> b = a`
		`>>> b is not a`
		`False`

Примечание 1. Оператор извлечения среза для объектов типа ndarray разрешается применять к каждой оси. Срезы разделяются запятыми, а диапазон осей описывается с помощью «...». Например, задан 5-ти мерный массив x (то есть x имеет 5 осей), тогда:

- x[1, 2, ...] эквивалентно x[1, 2, :, :, :];
- x[... , 3] эквивалентно x[:, :, :, :, 3];
- x[4, ... , 5, :] эквивалентно x[4, :, :, 5, :].

Пример выборки элементов двумерного массива, которые стоят на пересечении четных строк и нечетных столбцов

```
>>> a = np.array([[ 0,  1,  2,  3],
                  [4, 5, 6, 7],
```

```
                    [ 8,  9,  10,  11],
                    [ 12,  13,  14,  15]])
>>> a[::2, 1::2]
array([[ 1,  3],
       [ 9, 11]])
```

4 Встроенные методы по работе с элементами массива и с массивом в целом

Классификация встроенных методов по работе с массивом:

- методы преобразования массива;
- математические функции;
- методы выборки элементов массива;
- методы манипуляции формой массива;
- специальные методы.

Рассмотрим некоторые методы по работе с массивом (см. http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html) (табл. 3).

Таблица 3 – Встроенные методы типа данных ndarray

Метод	Описание	Пример

a.reshape(…)	Возвращает массив новой размерности.	`>>> a = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])`
		`>>> b = a.reshape((2, 2, 2))`
		`>>> b`
		`array([[[0, 1],`
		`[2, 3]],`
		`[[4, 5],`
		`[6, 7]]])`

a.resize(…)	Изменяет размерность и размер массива.	`>>> a = np.array([[0, 1], [2, 3]])`
		`>>> a.resize((2, 1))`
		`>>> a`
		`array([[0],`
		`[1]])`

a.fill(…)	Заполняет элементы массива заданным значением.	`>>> a = np.zeros((3, 3))`
		`>>> a.fill(5)`
		`>>> a`
		`array([[ 5.,  5.,  5.],`
		`[ 5.,  5.,  5.],`
		`[ 5.,  5.,  5.]])`

a.view(…)	Создает поверхностную копию массива.	`>>> a = np.array([1, 2])`
		`>>> b = a.view()`
		`>>> b.fill(1)`
		`>>> a`
		`array([1, 1])`

a.copy(…)	Создает глубокую копию массива.	`>>> a = np.array([[1,2,3],[4,5,6]])`
		`>>> b = a.copy()`
		`>>> b.fill(0)`
		`>>> a`
		`array([[1, 2, 3],`
		`[4, 5, 6]])`

a.min(…)	Возвращает	`>>> a = np.array([[3,2,1],[5,-`

	минимальный элемент массива или заданной оси.	1,6]])
		>>> a.min()
		-1
		>>> a.min(axis=0)
		array([3, -1, 1])
		>>> a.min(axis=1)
		array([1, -1])

a.max(…)	Возвращает максимальный элемент массива или заданной оси.	>>> a = np.array([[3,2,1],[5,-1,6]])
		>>> a.max()
		6
		>>> a.max(axis=0)
		array([5, 2, 6])
		>>> a.max(axis=1)
		array([3, 6])

a.mean(…)	Возвращает среднее элементов массива или заданной оси.	>>> a = np.array([[3,2,1],[5,-1,6]])
		>>> a.mean()
		2.6666666666666665
		>>> a.mean(axis=0)
		array([4. , 0.5, 3.5])
		>>> a.mean(axis=1)
		array([2. ,
		3.33333333])

a.prod(…)	Возвращает произведение элементов массива или заданной оси.	>>> a = np.array([[3,2,1],[5,-1,6]])
		>>> a.prod()
		-180
		>>> a.prod(axis=0)
		array([15, -2, 6])
		>>> a.prod(axis=1)
		array([6, -30])

a.sum(…)	Возвращает сумму элементов массива или заданной оси.	>>> a = np.array([[3,2,1],[5,-1,6]])
		>>> a.sum()
		16
		>>> a.sum(axis = 0)
		array([8, 1, 7])
		>>> a.sum(axis = 1)
		array([6, 10])

a.sort(…)	Сортирует элементы каждой оси массива.	>>> a = np.array([[3,2,1],[5,-1,6]])
		>>> a.sort()
		>>> a
		array([[1, 2, 3],
		[-1, 5, 6]])

5 Примеры

Рассмотрим простые примеры программ, использующих стандартные операторы и методы по работе с массивами.

Пример №1. Дана целочисленная квадратная матрица размером nxm. Посчитать количество отрицательных элементов матрицы.

```
import numpy as np
n = int(input('Количество строк = '))
m = int(input('Количество столбцов = '))
#   Создание   двумерного   массива   размером   nxm.   Элементы
инициализируются нулевым значением.
a = np.zeros((n, m), dtype=np.int_)
# Пользовательский ввод элементов массива.
for i in range(n):
    for j in range(m):
        a[i, j] = int(input('A['+str(i)+', '+str(j)+'] = '))
# Подсчет количества отрицательных элементов.
count = 0
for i in range(n):
    for j in range(m):
        if a[i, j] < 0:
            count += 1
# Вывод результата вычислений на экран.
print('количество отрицательных элементов: ', count)
```

Результат работы программы:

```
Количество строк = 3
Количество столбцов = 4
A[0, 0] = -5
A[0, 1] = 0
A[0, 2] = 2
A[0, 3] = 3
A[1, 0] = -9
A[1, 1] = 4
A[1, 2] = 5
A[1, 3] = 6
A[2, 0] = -5
A[2, 1] = 5
A[2, 2] = 4
A[2, 3] = -6
количество отрицательных элементов:  4
```

Пример №2. Дана целочисленная квадратная матрица размером nxm. Выполнить преобразование матрицы по следующему правилу: каждый отрицательный элемент заменить произведением его номера строки и столбца.

```
import numpy as np
import random
n = int(input('Количество строк = '))
m = int(input('Количество столбцов = '))
#   Создание   двумерного   массива   размером   nxm.   Элементы
инициализируются нулевым значением.
a = np.zeros((n, m), dtype=np.int_)
# Заполнение массива случайными целыми числами.
for i in range(n):
    for j in range(m):
        a[i, j] = random.randint(-128, 127)
# Вывод на экран исходного массива.
print(a)
# Подсчет количества отрицательных элементов.
```

```
count = 0
for i in range(n):
    for j in range(m):
        if a[i, j] < 0:
            a[i, j] = i * j
# Вывод результата вычислений на экран.
print('Результирующий массив:\n', a)
```

Результат работы программы:

```
Количество строк = 3
Количество столбцов = 4
[[ 122    8   70  116]
 [ -32   97 -113   58]
 [-122 -122 -125 -107]]
Результирующий массив:
 [[122   8  70 116]
 [  0  97   2  58]
 [  0   2   4   6]]
```