Лабораторная работа №7

Библиотека NumPy: Массивы и векторные вычисления.

1. Подключение библиотеки NumPy в проект

Для подключения библиотеки NumPy в создаваемый Вами проект на PyCharme выберите в меню File -> Settings. Откроется окно «Settings». Выберите в дереве слева «Project» -> «Project Interpreter» (рисунок 1):

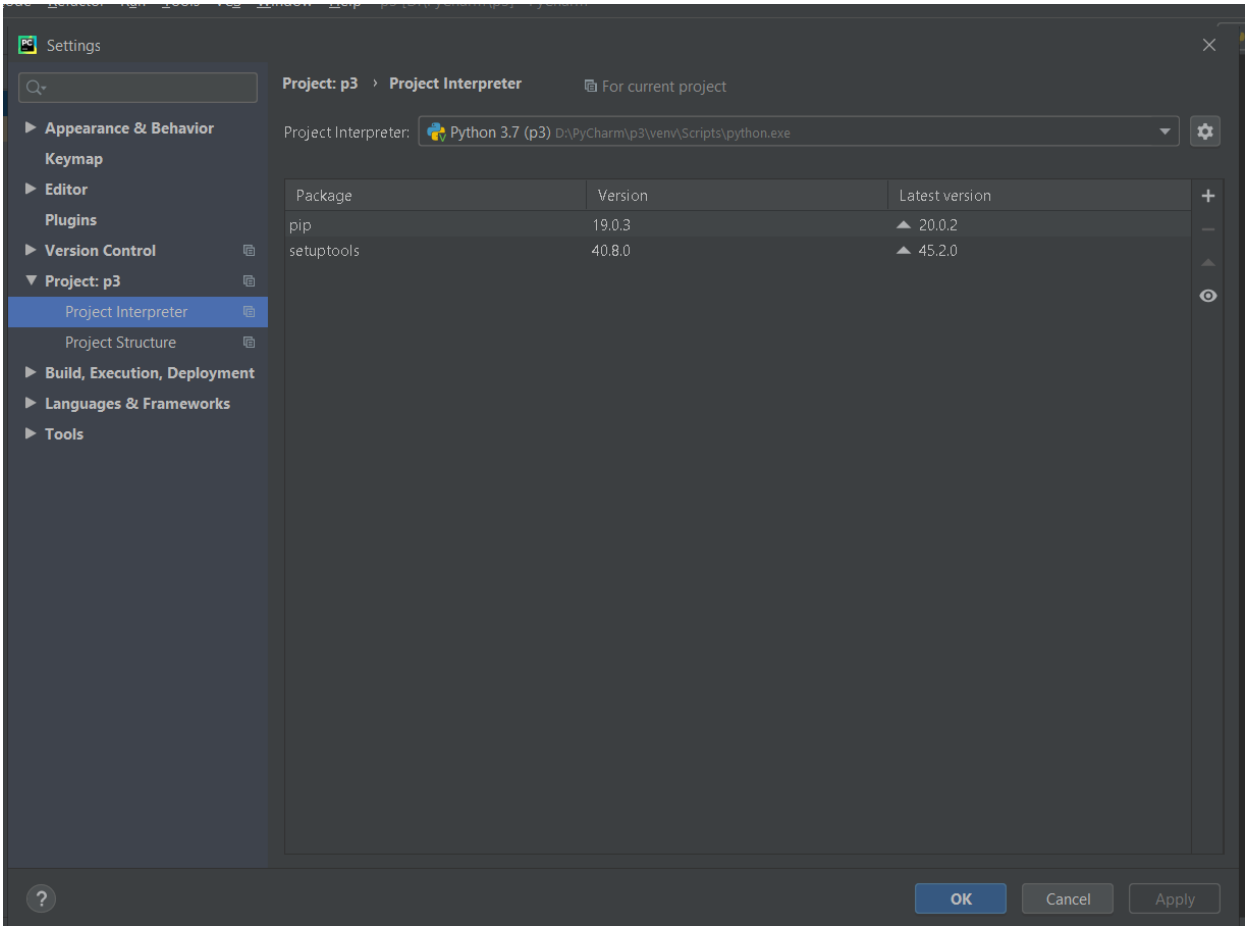

Рисунок 1 – Окно Settings

Для выбора пакета для установки в проект нажмите на знак «+» справа (рисунок 2):

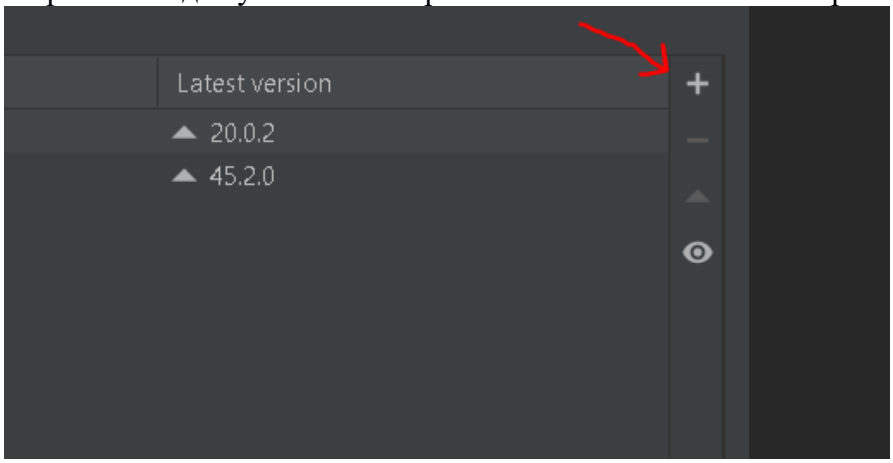

Рисунок 2 – Добавление пакета в проект

В открывшемся окне «Available Packages» введите в строке поиска «numpy». Справа появится описание пакета (Description). Нажмите на кнопку «Install Package» (рисунок 3).

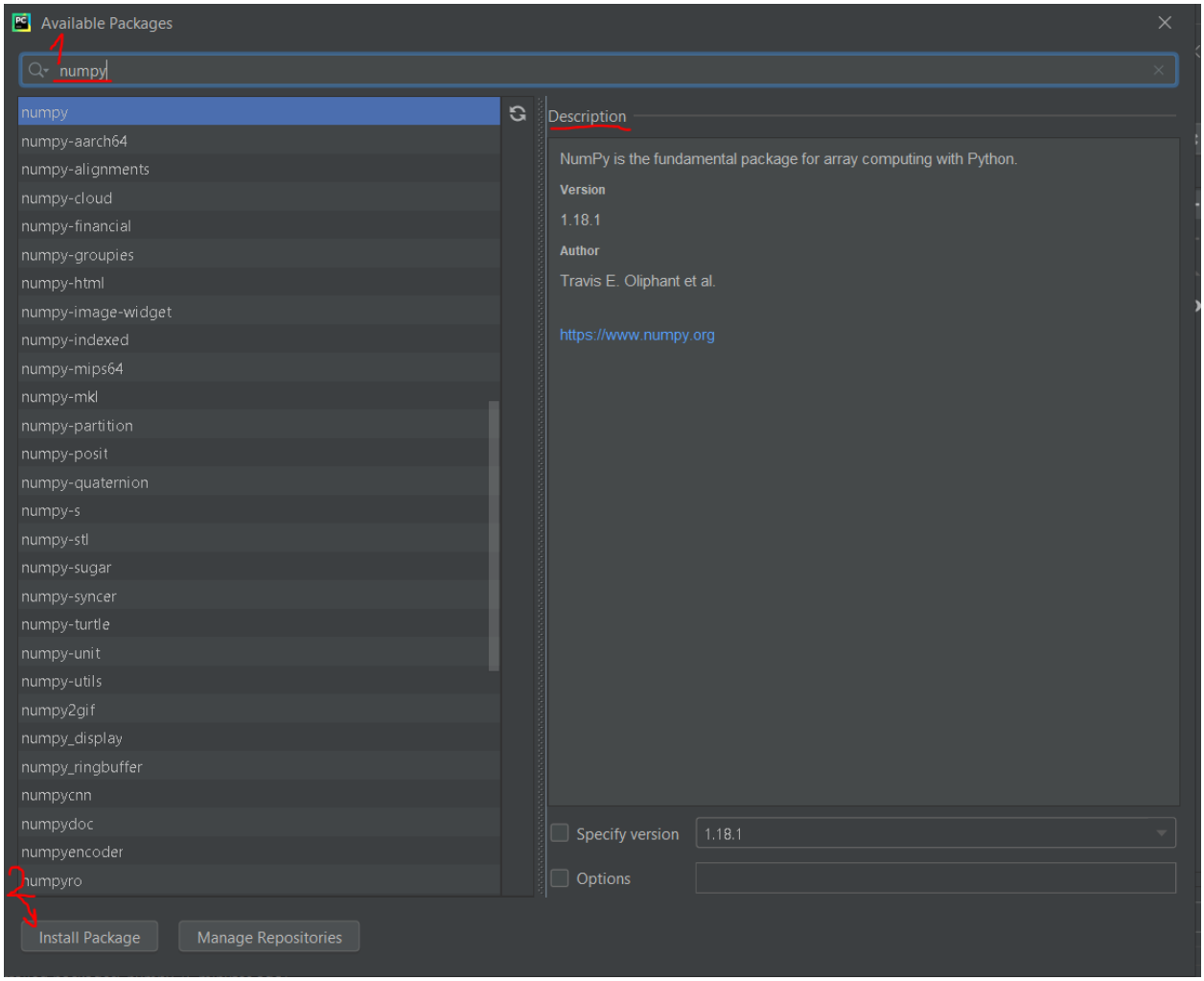

Рисунок 3 – Установка пакета

Внизу появится надпись об успешной установке пакета numpy (рисунок 4).

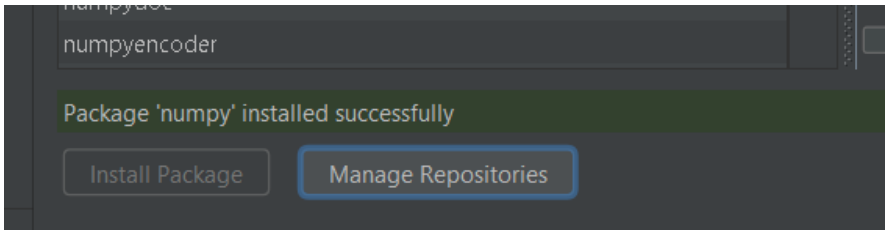

Рисунок 4 – Пакет успешно установлен

Теперь он есть в списке пакетов проекта (рисунок 5).

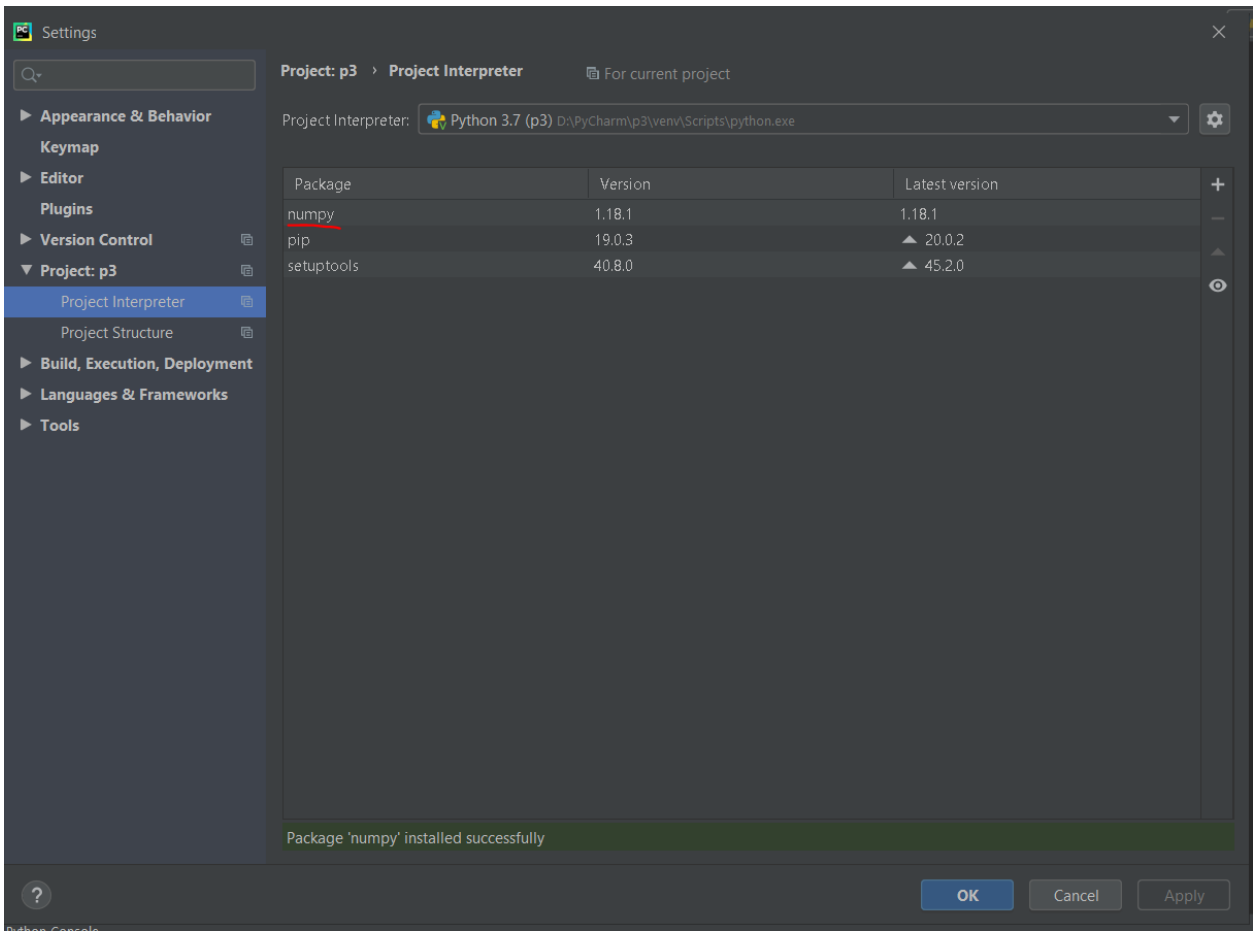

Рисунок 5 – Пакет numpy установлен

Ознакомиться с документацией numpy v1.18 можно по ссылке: https://numpy.org/doc/1.18/reference/index.html

1. Импорт библиотеки

Библиотека NumPy (сокращение от Numerical Python — «числовой Python») обеспечивает эффективный интерфейс для хранения и работы с плотными буферами данных. Массивы библиотеки NumPy похожи на встроенный тип данных языка Python list, но обеспечивают гораздо более эффективное хранение и операции с данными при росте размера массивов. NumPy предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия, и другие прикладные математические техники.

Сообщество NumPy и SciPy поддерживает онлайн руководство, включающие гайды и туториалы, тут: docs.scipy.org/doc.

Импорт модуля numpy

Есть несколько путей импорта. Стандартный метод это — использовать простое выражение:

```
>>> import numpy
```

Тем не менее, для большого количества вызовов функций numpy, становиться утомительно писать numpy.X снова и снова. Вместо этого намного легче сделать это так:

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

Это выражение позволяет нам получать доступ к numpy объектам используя np.X вместо numpy.X. Также можно импортировать numpy прямо в используемое пространство имен, чтобы вообще не использовать функции через точку, а вызывать их напрямую:

```
>>> from numpy import *
```

Однако, этот вариант не приветствуется в программировании на python, так как убирает некоторые полезные структуры, которые модуль предоставляет.

2. Создание массива и работа с ним

Главной особенностью numpy является объект array. Массивы схожи со списками в python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных, как float и int. С массивами можно проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее чем со списками.

Создание массива из списка:

```
a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<class 'numpy.ndarray'>
```

Здесь функция array принимает два аргумента: список для конвертации в массив и тип для каждого элемента. Ко всем элементам можно получить доступ и манипулировать ими также, как вы бы это делали с обычными списками:

```
>>> a[:2]
array([ 1.,  4.])
>>> a[3]
8.0
>>> a[0] = 5.
>>> a
array([ 5.,  4.,  5.,  8.])
```

Массивы могут быть и многомерными. В отличии от списков можно задавать команды в скобках. Вот пример двумерного массива (матрица):

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
[ 4.,  5.,  6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Array slicing работает с многомерными массивами аналогично, как и с одномерными, применяя каждый срез, как фильтр для установленного измерения. Используйте ":" в измерении для указывания использования всех элементов этого измерения:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
```

```
array([ 4.,  5.,  6.])
>>> a[:,2]
array([ 3.,  6.])
>>> a[-1:, -2:]
array([[ 5.,  6.]])
```

Метод shape возвращает количество строк и столбцов в матрице:

```
>>> a.shape
(2, 3)
```

Метод dtype возвращает тип переменных, хранящихся в массиве:

```
>>> a.dtype
dtype('float64')
```

Тут float64, это числовой тип данных в numpy, который используется для хранения вещественных чисел двойной точности. Так как же float в Python.

Метод len возвращает длину первого измерения (оси):

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> len(a)
2
```

Метод in используется для проверки на наличие элемента в массиве:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> 2 in a
True
>>> 0 in a
False
```

Массивы можно переформировать при помощи метода, который задает новый многомерный массив. Следуя следующему примеру, переформатируем одномерный массив из десяти элементов во двумерный массив, состоящий из пяти строк и двух столбцов:

```
>>> a = np.array(range(10), float)
>>> a
array([ 0.,  1.,  2., 3.,  4.,  5.,  6.,  7.,  8.,  9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0.,  1.],
[ 2.,  3.],
[ 4.,  5.],
[ 6.,  7.],
[ 8.,  9.]])
>>> a.shape
(5, 2)
```

Обратите внимание, метод reshape создает новый массив, а не модифицирует оригинальный.

Имейте ввиду, связывание имен в python работает и с массивами. Метод copy используется для создания копии существующего массива в памяти:

```
>>> a = np.array([1, 2, 3], float)
>>> b = a
>>> c =  a.copy()
>>> a[0] = 0
```

```
>>> a
array([0., 2., 3.])
>>> b
array([0., 2., 3.])
>>> c
array([1., 2., 3.])
```

Списки можно тоже создавать с массивов:

```
>>> a = np.array([1, 2, 3], float)
>>> a.tolist()
[1.0, 2.0, 3.0]
>>> list(a)
[1.0, 2.0, 3.0]
```

Заполнение массива одинаковым значением.

```
>>> a = array([1, 2, 3], float)
>>> a
array([ 1.,  2.,  3.])
>>> a.fill(0)
>>> a
array([ 0.,  0.,  0.])
```

Транспонирование массивов также возможно, при этом создается новый массив:

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0.,  1.,  2.],
[ 3.,  4.,  5.]])
>>> a.transpose()
array([[ 0.,  3.],
[ 1.,  4.],
[ 2.,  5.]])
```

Многомерный массив можно переконвертировать в одномерный при помощи метода flatten:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
[ 4.,  5.,  6.]])
>>> a.flatten()
array([ 1.,  2.,  3.,  4.,  5.,  6.])
```

Два или больше массивов можно сконкатенировать при помощи метода concatenate:

```
>>> a = np.array([1,2], float)
>>> b = np.array([3,4,5,6], float)
>>> c = np.array([7,8,9], float)
>>> np.concatenate((a, b, c))
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

Если массив не одномерный, можно задать ось, по которой будет происходить соединение. По умолчанию (не задавая значения оси), соединение будет происходить по первому измерению:

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1.,  2.],
[ 3.,  4.],
[ 5.,  6.],
[ 7.,  8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1.,  2.],
[ 3.,  4.],
[ 5.,  6.],
[ 7.,  8.]])
>>>
np.concatenate((a,b), axis=1)
array([[ 1.,  2.,  5.,  6.],
[ 3.,  4.,  7.,  8.]])
```

В заключении, размерность массива может быть увеличена при использовании константы newaxis в квадратных скобках:

```
>>> a = np.array([1, 2, 3], float)
>>> a
array([1., 2., 3.])
>>> a[:,np.newaxis]
array([[ 1.],
[ 2.],
[ 3.]])
>>> a[:,np.newaxis].shape
(3,1)
>>> b[np.newaxis,:]
array([[ 1.,  2.,  3.]])
>>> b[np.newaxis,:].shape
(1,3)
```

3. Другие пути создания массивов

Функция arange аналогична функции range, но возвращает массив:

```
>>> np.arange(5, dtype=float)
array([ 0.,  1.,  2.,  3.,  4.])
>>> np.arange(1, 6, 2, dtype=int)
array([1, 3, 5])
```

Функции zeros и ones создают новые массивы с установленной размерностью, заполненные этими значениями. Это, наверное, самые простые в использовании функции для создания массивов:

```
>>> np.ones((2,3), dtype=float)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])
```

Также есть некоторое количество функций для создания специальных матриц. Для создания квадратной матрицы с главной диагональю, которая заполненная единицами, воспользуемся методом identity:

```
>>> np.identity(4, dtype=float)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Функция eye возвращает матрицу с единичками на к-атой диагонали:

```
>>> np.eye(4, k=1, dtype=float)
array([[ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.]])
```

4. Математические операции над массивами

Когда для массивов мы используем стандартные математические операции, должен соблюдаться принцип: элемент--элемент. Это означает, что массивы должны быть одинакового размера во время сложения, вычитания и тому подобных операций:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a - b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

Для двухмерных массивов, умножение остается поэлементным и не соответствует умножению матриц. Для этого существуют специальные функции, которые мы изучим позже.

```
>>> a = np.array([[1,2], [3,4]], float)
>>> b = np.array([[2,0], [1,3]], float)
>>> a * b
array([[2., 0.], [3., 12.]])
```

Вдобавок к стандартным операторам, в numpy включена библиотека стандартных математических функций, которые могут быть применены поэлементно к массивам. Собственно функции: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, и arctanh.

```
>>> a = np.array([1, 4, 9], float)
>>> np.sqrt(a)
array([ 1.,  2.,  3.])
```

Функции floor, ceil и rint возвращают нижние, верхние или ближайшие (округлённое) значение:

```
>>> a = np.array([1.1, 1.5, 1.9], float)
>>> np.floor(a)
array([ 1.,  1.,  1.])
```

```
>>> np.ceil(a)
array([ 2.,  2.,  2.])
>>> np.rint(a)
array([ 1.,  2.,  2.])
```

Также в numpy включены две важные математические константы:

```
>>> np.pi
3.1415926535897931
>>> np.e
2.7182818284590451
```

5. Перебор элементов массива

Проводить итерацию массивов можно аналогично спискам:

```
>>> a = np.array([1, 4, 5], int)
>>> for x in a:
...   print x
1
4
5
```

Для многомерных массивов итерация будет проводиться по первой оси, так, что каждый проход цикла будет возвращать «строку» массива:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for x in a:
...   print x
[ 1.  2.]
[ 3.  4.]
[ 5.  6.]
```

Множественное присваивание также доступно при итерации:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for (x, y) in a:
...   print x * y
2.0
12.0
30.0
```

6. Базовые операции над массивами

Для получения каких-либо свойств массивов существует много функций. Элементы могут быть суммированы или перемножены:

```
>>> a = np.array([2, 4, 3], float)
>>> a.sum()
9.0
>>> a.prod()
24.0
```

В этом примере были использованы функции массива. Также можно использовать собственные функции numpy:

```
>>> np.sum(a)
```

```
9.0
>>> np.prod(a)
24.0
```

Для большинства случаев могут использоваться оба варианта.Некие функции дают возможность оперировать статистическими данными. Это функции mean (среднее арифметическое), вариация и девиация:

```
>>> a = np.array([2, 1, 9], float)
>>> a.mean()
4.0
>>>
a.var()
12.666666666666666
>>> a.std()
3.5590260840104371
```

Можно найти минимум и максимум в массиве:

```
>>> a = np.array([2, 1, 9], float)
>>> a.min()
1.0
>>> a.max()
9.0
```

Функции argmin и argmax возвращают индекс минимального или максимального элемента:

```
>>> a = np.array([2, 1, 9], float)
>>> a.argmin()
1
>>> a.argmax()
2
```

Для многомерных массивов каждая из функций может принять дополнительный аргумент axis и в зависимости от его значения выполнять функции по определенной оси, помещая результаты исполнения в массив:

```
>>> a = np.array([[0, 2], [3, -1], [3, 5]], float)
>>> a.mean(axis=0)
array([ 2.,  2.])
>>> a.mean(axis=1)
array([ 1.,  1.,  4.])
>>> a.min(axis=1)
array([ 0., -1.,  3.])
>>> a.max(axis=0)
array([ 3.,  5.])
```

Как и списки, массивы можно отсортировать:

```
>>> a = np.array([6, 2, 5, -1, 0], float)
>>> sorted(a)
[-1.0, 0.0, 2.0, 5.0, 6.0]
>>> a.sort()
>>> a
array([-1.,  0.,  2.,  5.,  6.])
```

Значения в массиве могут быть «сокращены», чтобы принадлежать заданному диапазону. Это тоже самое что применять min(max(x, minval), maxval) к каждому элементу x:

```
>>> a = np.array([6, 2, 5, -1, 0], float)
>>> a.clip(0, 5)
array([ 5.,  2.,  5.,  0.,  0.])
```

Уникальные элементы могут быть извлечены вот так:

```
>>> a = np.array([1, 1, 4, 5, 5, 5, 7], float)
>>> np.unique(a)
array([ 1.,  4.,  5.,  7.])
```

Для двухмерных массивов диагональ можно получить так:

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> a.diagonal()
array([ 1.,  4.])
```

7. Операторы сравнения и тестирование значений

Булево сравнение может быть использовано для поэлементного сравнения массивов одинаковых длин. Возвращаемое значение это массив булевых True/False значений:

```
>>> a = np.array([1, 3, 0], float)
>>> b = np.array([0, 3, 2], float)
>>> a > b
array([ True, False, False], dtype=bool)
>>> a == b
array([False,  True, False], dtype=bool)
>>> a <= b
array([False,  True,  True], dtype=bool)
```

Результат сравнения может быть сохранен в массиве:

```
>>> c = a > b
>>> c
array([ True, False, False], dtype=bool)
```

Массивы могут быть сравнены с одиночным значением:

```
>>> a = np.array([1, 3, 0], float)
>>> a > 2
array([False,  True, False], dtype=bool)
```

Операторы any и all могут быть использованы для определения истинны ли хотя бы один или все элементы соответственно:

```
>>> c = np.array([ True, False, False], bool)
>>> any(c)
True
>>> all(c)
False
```

Комбинированные булевы выражения могут быть применены к массивам по принципу элемент — элемент используя специальные функции logical_and, logical_or и logical_not:

```
>>> a = np.array([1, 3, 0], float)
>>> np.logical_and(a > 0, a < 3)
array([ True, False, False], dtype=bool)
>>> b = np.array([True, False, True], bool)
>>> np.logical_not(b)
array([False,  True, False], dtype=bool)
>>> c = np.array([False, True, False], bool)
>>> np.logical_or(b, c)
array([ True,  True,  False], dtype=bool)
```

Функция where создает новый массив из двух других массивов одинаковых длин используя булев фильтр для выбора межу двумя элементами. Базовый синтаксис: where(boolarray,truearray, falsearray):

```
>>> a = np.array([1, 3, 0], float)
>>> np.where(a != 0, 1 / a, a)
array([ 1.        ,  0.33333333,  0.        ])
```

С функцией where так же может быть реализовано «массовое сравнение»:

```
>>> np.where(a > 0, 3, 2)
array([3, 3, 2])
```

Дополнительно краткое описание операций с numpy на русском можно найти по ссылке:

https://physics.susu.ru/vorontsov/language/numpy.html

8. Практические задания

1. Создать вектор (одномерный массив) размера 15, заполненный нулями
2. Создать вектор размера 8, заполненный числом 3.2
3. Создать вектор размера 15, заполненный нулями, но пятый элемент равен 1
4. Создать вектор со значениями от 12 до 43
5. Создать массив 3x3x2 со случайными значениями
6. Создать массив 12x12 со случайными значениями, найти минимум и максимум
7. Создать матрицу с 1 внутри, и 0 на границах
8. Создать 8x8 матрицу и заполнить её в шахматном порядке
9. Создать 8x8 матрицу и заполнить её в шахматном порядке, используя функцию tile
10. Перемножить матрицы 4x2 и 2x5
11. Дан массив, поменять знак у элементов, значения которых между 4 и 7
12. Создать 6x6 матрицу со значениями в строках от 0 до 5
13. Отсортировать вектор
14. Проверить, одинаковы ли 2 numpy массива
15. Заменить максимальный элемент на -1
16. Найти ближайшее к заданному значению число в заданном массиве

17. Создать структурированный массив, представляющий координату (x,y) и цвет (r,g,b)

18. Отнять среднее из каждой строки в матрице

19. Поменять 2 строки в матрице

20. Рассмотрим 2 набора точек P0, P1 описания линии (2D) и точку p, как вычислить расстояние от p до каждой линии i (P0[i],P1[i]). Поиск расстояния от токи p до линии i (P0[i],P1[i]) задать отдельной функцией.

21. Найти n наибольших значений в массиве

22. Дано две матрица A и B одинакового размера. Сравнить в скольких элементах они отличаются.

23. Дан массив A. Выберите все элементы больше 4 и кратные 3.

24. Дана матрица A(2,3) измените её размер на A(3,2)

25. Продемонстрируйте скалярное умножение матриц.