NumPy

NumPy (Numeric Python) – это сторонний пакет, предоставляющий методы для работы с массивами.

Когда мы разбирали работу с пакетами, мы уже установили NumPy через командную строку с помощью строки:

pip install numpy

Для работы с NumPy импортируем модуль «numpy»:

import numpy as np

Так как мы будем много использовать данный модуль, мы даем ему короткое название «np».

# Массивы

Главной особенностью NumPy является объект array (массив). Массив в NumPy схож с листом Python, за исключением того, что в нем можно хранить элементы только одинакового типа, а также он является неизменяемым по длине (т.е. мы не можем добавлять или удалять из него элементы).

Объявим массив в NumPy с помощью функции array(), которой аргументом укажем лист:

import numpy as np  
  
a = np.array([1, 2, 3.0])  
print('a =', a)  
  
b = np.array([1, 2, 3])  
print('b =', b)  
  
c = np.array([1., 2., 3.0], dtype=int)  
print('c =', c)

Вывод на консоль:

a = [1. 2. 3.]

b = [1 2 3]

c = [1 2 3]

Из примера видно, что если не указать тип функции «array», то тип данных определяется автоматически, иначе – все элементы приводятся к указанному типу.

Массивы могут быть многомерными. Многомерный массив можно определить с помощью листа, элементы которого тоже являются листами одинаковой длины. Array() трансформирует вложенные последовательности в массивы, в итоге мы получаем массив массивов:

import numpy as np  
  
a = np.array([[1, 2], [3, 4]])  
print(a)

Вывод на консоль:

[[1 2]

[3 4]]

Если мы указываем листы разной длины, то получаем массив, тип элементов которого является «list», т.е. мы получим не массив массивов, а массив листов:

import numpy as np  
  
b = np.array([[1, 2], [3, 4, 5]])  
print(b)

Вывод на консоль:

[list([1, 2]) list([3, 4, 5])]

Кроме функции array(), массив можно создать еще некоторыми способами, например:

import numpy as np  
  
a = np.zeros((2, 3))  
print(a)  
  
b = np.ones((2, 3))  
print(b)

Вывод на коносоль:

[[0. 0. 0.]

[0. 0. 0.]]

[[1. 1. 1.]

[1. 1. 1.]]

Функции np.zeros() и np.ones() создают массив указанной размерности, состоящий из нулей и единиц соответственно. Размерность (2, 3) означает, что в массиве будет два элемента, которые являются массивами, содержащими 3 элемента.

К элементам массива NumPy можно обращаться также, как и к элементам листа:

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

Вывод на консоль:

2

[2 3]

К элементам многомерного массива можно обращаться также, используя индекс или срез для каждого измерения, указывая их через запятую или несколько квадратных скобок:

import numpy as np  
  
a = np.array([[1, 2, 3], [4, 5, 6]])  
print(a[1, 1]) # аналогично a[1][1]  
print(a[1:, 1:]) # аналогично a[1:][1:]

Вывод на консоль:

5

[[5 6]]

Массивы поддерживают in и len():

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

Вывод на консоль:

True

3

Однако, если применить функцию len() к многомерному массиву, то она вернет длину его первого измерения:

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

Вывод на консоль:

2

Чтобы узнать реальный размер массива используется метод shape():

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

Вывод на консоль:

(2, 3)

Мы можем транспонировать массив, используя метод transpose() – т.е. поменять строки и столбцы местами:

import numpy as np  
  
print("Исходный: ")  
a = np.array([[1, 2, 3], [4, 5, 6]])  
print(a)  
print(np.shape(a))  
  
print("Транспонированный: ")  
a = np.transpose(a)  
print(a)  
print(np.shape(a))

Вывод на консоль:

Исходный:

[[1 2 3]

[4 5 6]]

(2, 3)

Транспонированный:

[[1 4]

[2 5]

[3 6]]

(3, 2)

Также, мы можем изменить размерность массива с помощью функции reshape(), которая возвращает указанный массив приведенный к указанной размерности. Только при условии, что

shape\_1 \* shape\_2 \* … \* shape\_n = new\_shape\_1 \* new\_shape\_2 \* … \* new\_shape\_m:

import numpy as np  
  
print("Исходный: ")  
a = np.array([[1, 2, 3, 4, 5, 6]])  
print(a)  
print(a.shape)  
  
print("С измененным размером: ")  
a = np.reshape(a, (2, 3))  
print(a)  
print(a.shape)

Вывод на консоль:

Исходный:

[[1 2 3 4 5 6]]

(1, 6)

С измененным размером:

[[1 2 3]

[4 5 6]]

(2, 3)

# Операции над массивами

## Математические операции

Мы можем использовать стандартные математические операции, такие как сложение, умножение и т.д., которые будут применяться к массивам поэлементно, поэтому, чтобы они работали необходимо применять их к массивам одинаковой размерности:

import numpy as np  
  
a = np.array([2, 2, 2])  
print('a =', a)  
b = np.array([1, 2, 3])  
print('b =', b)  
  
print('\nОперации:')  
  
print('a + b =', a + b)  
print('a - b =', a - b)  
print('a \* b =', a \* b)  
print('a / b =', a / b)  
print('a \*\* b =', a \*\* b)

Вывод на консоль:

a = [2 2 2]

b = [1 2 3]

Операции:

a + b = [3 4 5]

a - b = [ 1 0 -1]

a \* b = [2 4 6]

a / b = [2. 1. 0.66666667]

a \*\* b = [2 4 8]

При несоответствии размеров возникает исключение:

import numpy as np  
  
a = np.array([2, 2, 2])  
print('a =', a)  
b = np.array([1, 2])  
print('b =', b)  
  
print('a + b =', a + b)

Вывод на консоль:

a = [2 2 2]

b = [1 2]

Traceback (most recent call last):

…

ValueError: operands could not be broadcast together with shapes (3,) (2,)

Однако, если размерности «подходят» друг другу, т.е. одну можно применить к другой, то сложить получится:

import numpy as np  
  
a = np.array([[1, 2], [3, 4]])  
print('a\n', a)  
b = np.array([1, 2])  
print('b\n', b)  
  
print('a + b\n', a + b)

Вывод на консоль:

a

[[1 2]

[3 4]]

b

[1 2]

a + b

[[2 4]

[4 6]]

В данном случае массив b был повторен 2 раза, т.е. Python преобразовал массив b из [1, 2] в [[1, 2], [1, 2]].

Кроме того, в NumPy включены стандартные математические функции, такие как: sin, cos, tan, log, log10 и т.д., которые применяются к массиву поэлементно.

import numpy as np  
  
a = np.array([[1, 10, 100]])  
print(np.log10(a))

Вывод на консоль:

[[0. 1. 2.]]

## операции над одним массивом

Иногда, нам нужно узнать сумму всех элементов в массиве. Или узнать наибольшее число в строке. Для этого NumPy предоставляет большой набор функций.

Разберем пример: в фирме работают три сотрудника, за 3 дня первый сотрудник сделал 3, 7, и 5 единиц товара соответственно, второй – 8, 4, 7, третий – 10, 2, 4. Мы хотим посчитать статистику их работы:

1. сколько единиц товара произведено каждым из сотрудников за 3 дня;
2. кто сделал наибольшее количество товаров;
3. каково максимальное количество товаров, сделанных одним сотрудником в один день;
4. среднее количество товаров в каждый из дней;
5. суммарное количество проделанной работы.

Для начала создадим массив: по нулевой оси будут располагаться сотрудники, по первой – дни.

import numpy as np  
  
products\_statistic = np.array(  
 [  
 [3, 7, 5],  
 [8, 4, 7],  
 [10, 2, 4]  
 ]  
)

Для того, чтобы посчитать сколько единиц товара произведено каждым из сотрудников за 3 дня, необходимо найти сумму каждой строки. Для этого используем функцию sum(), указав ось, по которой будем суммировать:

sum\_of\_days = np.sum(products\_statistic, axis=1)  
print(sum\_of\_days)

Вывод на консоль:

[15 19 16]

Вывод означает, что первый сотрудник за 3 дня сделал 15 единиц товара, второй – 19, а третий – 16. Теперь необходимо выяснить, кто же сделал наибольшее количество товаров. Конечно, когда у нас 3 сотрудника, мы без труда можем определить лучшего. Однако, если наша фирма разрастется и в ней будет, например, 500 человек, то это вызвало бы некоторые затруднения. Поэтому, для того, чтобы найти номер лучшего сотрудника, используем функцию argmax(), которая возвращает индекс наибольшего элемента по оси, но мы ее указывать не будем, так как имеем всего 1 измерение:

best\_employee = np.argmax(sum\_of\_days) + 1 # так как нумерация в Python начинается с 0, прибавляем 1  
print(best\_employee)

Вывод на консоль:

2

Теперь узнаем, каково максимальное количество товаров, сделанных одним сотрудником в один день. Для этого используем функцию max():

best\_result\_in\_one\_day = np.max(products\_statistic)  
print(best\_result\_in\_one\_day)

Вывод на консоль:

10

Чтобы узнать среднее количество товаров в каждый из дней, используем функцию mean(), указав ей 0 измерение:

mean\_result = np.mean(products\_statistic, axis=0)  
print(mean\_result)

Вывод на консоль:

[7. 4.33333333 5.33333333]

И наконец, с помощью знакомой уже функции sum(), которой в этот раз не будем указывать ось, так как нам нужна сумма всего массива, узнаем суммарное количество проделанной работы:

sum\_result = np.sum(products\_statistic)  
print(sum\_result)

Вывод на консоль:

50

Помимо перечисленных функций в NumPy существует множество других:

* prod() – возвращает произведение элементов массива;
* min() – возвращает минимальное значение в массиве;
* argmin() – возвращает индекс минимального элемента в массиве;
* sort() – возвращает отсортированный массив;
* и т.д.

При этом каждой из них можно указать ось, по которой будет производиться операция, а если она не указана – то по всему массиву.

## Операции с двумя массивами

NumPy имеет множество функций и для операции между двумя массивами. Так, например, чтобы получить скалярное произведение двух векторов (одномерных матриц) используем функцию dot():

import numpy as np  
  
a = np.array([1, 3])  
b = np.array([5, 2])  
  
print(np.dot(a, b))

Вывод на консоль:

11

С помощью той же функции мы получаем матричное перемножение:

import numpy as np  
  
a = np.array([  
 [1, 2],   
 [3, 4]  
])  
b = np.array([1, 2])  
  
print(np.dot(a, b))

Вывод на консоль:

[ 5 11]

# Генерация случайных чисел

Помимо работы с массивами NumPy умеет еще очень многое. Рассмотрим работу со случайными числами.

Чтобы сгенерировать случайное число, мы можем использовать функцию rand():

import numpy as np  
  
print(np.random.rand())

Вывод на консоль:

0.991578541962864

Чтобы сгенерировать случайное целое число в диапазоне [n, m] используем метод randint():

import numpy as np  
  
print(np.random.randint(0, 10))

Вывод на консоль:

6

Однако, random в NumPy очень полезен тем, что может генерировать не только числа, но и целые массивы случайных чисел. Укажем функциям rand() и randint() размер массива, который мы хотим сгенерировать:

import numpy as np  
  
print(np.random.rand(5))

print(np.random.randint(0, 10, (2, 3)))

Вывод на консоль:

[0.82417449 0.54673615 0.06949854 0.82871385 0.83762281]

[[3 9 9]

[3 0 4]]

Также, можно генерировать числа, согласно различным распределениям:

* uniform() – равномерное распределение;
* normal() – распределение Гаусса.

Также есть распределение Пуассона, Коши и многие другие.

Порой, нам необходимо не сгенерировать случайную последовательность, а перемешать существующую. Для этого используем метод shuffle().

import numpy as np  
  
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(a)  
np.random.shuffle(a)  
print(a)

Вывод на консоль:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[7, 0, 5, 1, 9, 6, 4, 2, 3, 8]

А с помощью функции choice() мы можем достать один или несколько элементов из массива в случайном порядке:

import numpy as np  
  
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(np.random.choice(a, 3))

Вывод на консоль:

[5 1 3]