Theory: Operations with an array

20 minutes 0 / 5 problems solved

This topic deals with the operations that can be carried out with a single array in NumPy. Previously, you have learned the basic operations with integers in Python, such as addition, subtraction, or multiplication. NumPy includes a set of operations for the arrays with various dimensions. Some of them are similar to the basic ones, but others are different.

§1. Standard functions with one-dimensional arrays

The easiest way to explain the basics of array operations is to show a few examples with one-dimensional arrays. You will learn to use them on multi-dimensional arrays later in this topic.

Let's start with finding the sum of the array elements. Similarly to basic addition in Python, we use the method `sum()` that returns the sum of all values in the given array:

```
array_1 = np.array([1, 2, 3, 4, 5])
print(array_1.sum())  # 15
```

The method `array.prod()` returns the product of the array elements. It is similar to multiplication in Python:

```
print(array_1.prod())  # 120
```

`array.max()` and `array.min()` return the maximum or minimum value in the array, respectively:

```
print(array_1.max())  # 5
print(array_1.min())  # 1
```

Finally, `array.argmax()` and `array.argmin()` return the **indexes** of the maximum or minimum values in the array.

```
print(array_1.argmax())  # 4
print(array_1.argmin())  # 0
```

Be careful with data types in your array! All operations described in this topic work only with integers and floats.

§2. Statistical functions with one-dimensional arrays

NumPy also provides tools to implement statistical calculations. Take a look at the main statistical functions and their descriptions:

NumPy methods and functions	Descriptions
`array.std()` or `np.std(array)`	Computation of the Standard Deviation
`array.mean()` or `np.mean(array)`	Computation of the Arithmetic Mean
`array.var()` or `np.var(array)`	Computation of the Variance
`np.median(array)`	Computation of the Array Element Median
`np.average(array)`	Computation of the Array Element Average

You can see the examples of these operations below:

Current topic:

Operations with an array

Topic depends on:

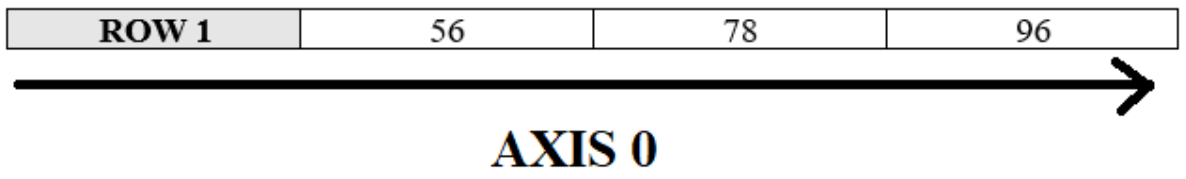
✕ Intro to NumPy

Table of contents:

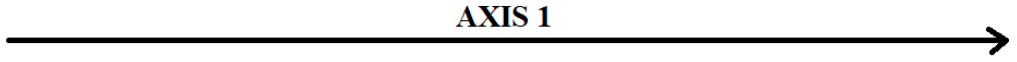
```
array_2 = np.array([3, 56, 78, 32, 65, 0, 9])
print(array_2.std())  # 29.576604101960076
print(array_2.mean())  # 34.714285714285715
print(array_2.var())  # 874.7755102040816
print(np.median(array_2))  # 32.0
print(np.average(array_2))  # 34.714285714285715
```

§3. Axes in arrays

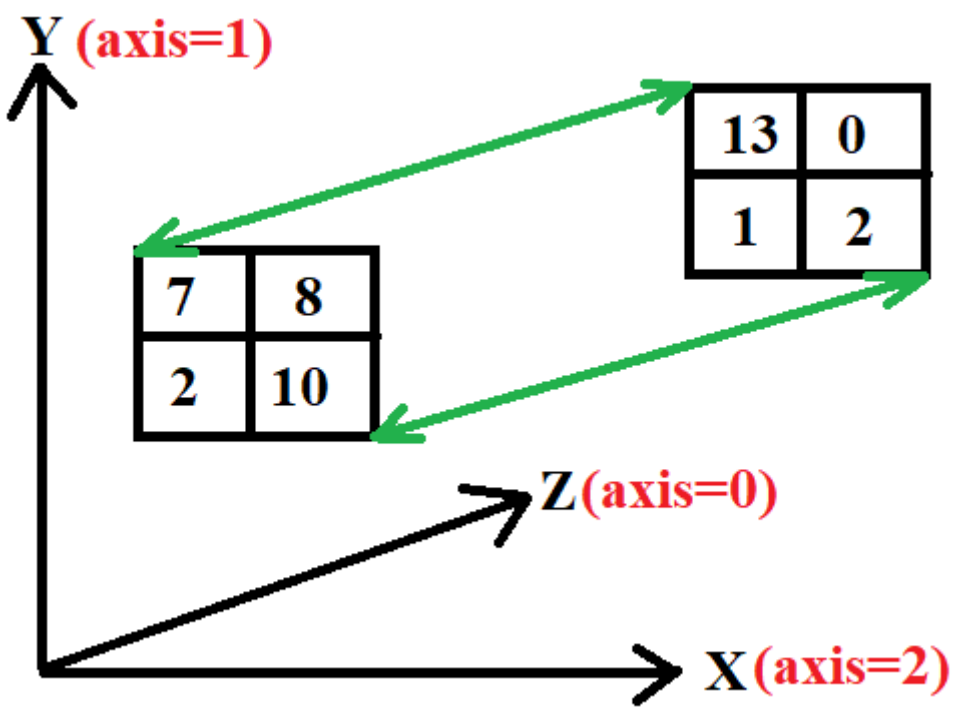
You already know that axes in the arrays have their dimensions. But what exactly is an axis? To put it short, an axis is a direction along the columns and rows of your array. One-dimensional arrays have only one row. Consider an example for the array `np.array([56, 78, 96])`:



Now, let's take a look at another example with the two-dimensional array `np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])`. Here, enumeration starts with columns. Here, the columns are Axis 0, and the rows are Axis 1:



Working with two-dimensional arrays is rather simple, but how do we analyze axes in a three-dimensional array? It is easier than you might think: just imagine a coordinate system. Once again, let's look at an example. We have an array `np.array([[[7, 8], [2, 10]], [[13, 0], [1, 2]]])` .



Our array consists of two subarrays, each consisting of two columns and two rows. Axis 1 here represents the vertical axis, Axis 2 represents the horizontal axis, and Axis 0 is the so-called Z axis that corresponds to the depth of the array.

So, the pairs `[7, 13]`, `[8, 0]`, `[2, 1]`, and `[10, 2]` will be analyzed along the Axis 0, the pairs `[7, 2]`, `[8, 10]`, `[13, 1]`, and `[0, 2]` will be analyzed along the Axis 1, and the pairs `[7, 8]`, `[2, 10]`, `[13, 0]`, and `[1, 2]` will be analyzed along the Axis 2.

Arrays with four or more dimensions require some imagination to represent their structure, but it works the same way: you just add one more axis to your coordinate system.

There is a cool trick to remember which index corresponds to which axis. When we have only one axis (that is, a row), it is always Axis 0. When a column appears, it becomes Axis 0, and the row switches to

Axis 1. Finally, in the example above, the new axis becomes Axis 0, and the other two shift to Axis 1 (columns) and Axis 2 (rows). The same logic also works for multi-dimensional arrays with more than three dimensions.

§4. Specifying axes for operations in multidimensional arrays

Why are axes so important in NumPy? Sometimes you need to specify the axis you are working with, for example, when you carry out some operations with your array. This section contains some particular examples.

There is no need to specify Axis 0 in one-dimensional arrays, but if you do, no errors will occur.

```
array_3 = np.array([1, 2, 3])
print(array_3.sum())  # 6
print(array_3.sum(axis=0))  # 6
```

Mind the number of axes you're dealing with. If you print an index that is out of bounds of a certain array dimension, this will cause an error.

Let's analyze a new two-dimensional array. If we don't specify any axes, the operations will be carried out on all elements, just like with a one-dimensional array:

```
array_4 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])
print(np.mean(array_4))  # 2.581988897471611
print(array_4.sum())  # 45
```

If you want to carry out an operation with a particular axis, use the `axis=X` command, where `X` is the index of the desired axis:

```
print(np.mean(array_4, axis=0))  # [4. 5. 6.]
print(array_4.sum(axis=1))  # [ 6 15 24]
```

As you can see, `np.mean(array_4, axis=0)` with the specified axis calculates the mean values for the elements in each column. For example, the mean of the first column is 4, because `(1 + 4 + 7) / 3 = 4`. Similarly, `array_4.sum(axis=1)` calculates the sum of all values in each row. For the first row, the sum is `1 + 2 + 3 = 6`.

Now let's take a look at another situation that may occur when we work with a three-dimensional array:

```
array_5 = np.array([[[1, 2], [3, 4]],
                    [[5, 6], [7, 8]]])
print(array_5.sum(axis=0))
# [[ 6  8]
#  [10 12]]
print(array_5.max(axis=1))
# [[3 4]
#  [7 8]]
print(array_5.mean(axis=2))

# [[1.5 3.5]

#  [5.5 7.5]]
```

`array_5.sum(axis=0)` and `array_5.max(axis=1)` work the same way as the example above; they calculate the sum of all values in each row and the maximum value in each column, respectively. As for `array_5.mean(axis=2)`, it adds each pair `[1, 2]`, `[3, 4]`, `[5, 6]`, `[7, 8]`, calculates every mean, and returns a new array with these mean values.

§5. Conclusion

In this topic, we have learned:

- how to carry out operations with one-dimensional arrays;
- why axes are important and how they work;
- how to specify axes when you're dealing with operations in NumPy.

Now that you have this knowledge, it's time to test it and make sure you can put it to practice.

Report a typo

24 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (1) Hints (0) Useful links (0) Show discussion