

Theory: Array creation and indexing

🕒 30 minutes 0 / 5 problems solved

Skip this topic

Start practicing

314 users solved this topic. Latest completion was about 8 hours ago.

In this topic, we continue discussing NumPy. We will focus on new ways of creating arrays, as well as on methods of their indexing and slicing. The introduction to NumPy showed the simplest way to create an array — converting a Python list using `np.array()`. But there are other ways we may need to know.

§1. Array range and evenly spaced array

The first function that helps to create arrays in NumPy, is `np.arange()`. It is similar to the built-in `range()` generator of numbers, but `np.arange()` returns an array. Let's have a look at the example.

```
1 array_1 = np.arange(5)
2 print(array_1) # [0 1 2 3 4]
```

Click and drag to move

You can also create an array by specifying a start element, a stop element, and a step: `np.arange(start, stop, step)`. The default step value equals to 1. Note that the start element is *included in the range* while the stop element *is not*.

```
1 array_2 = np.arange(5, 9)
2 array_3 = np.arange(5, 6, 0.2)
3 print(array_2) # [5 6 7 8]
4 print(array_3) # [5. 5.2 5.4 5.6 5.8]
```

Click and drag to move

If you want to space all the elements in your array evenly, `np.linspace()` can help you. It takes the start value, the end value, and the `num` parameter which is the total number of elements. The default `num` value is 50, so in the second example below, there will be 50 elements in the array `array_5`, evenly spaced between 21 and 23. As opposed to the `np.arange()` function, the end value in `np.linspace()` is always *included* in the array.

```
1 array_4 = np.linspace(21, 23, num=5)
2 array_5 = np.linspace(21, 23)
3 print(array_4) # [21. 21.5 22. 22.5 23. ]
4
print(array_5) # [21.          21.04081633 21.08163265 21.12244898 ... 22.95918367
23.          ]
```

Click and drag to move

§2. Arrays filled with ones and zeros

Sometimes we need an array consisting of only zeroes or only ones. In NumPy, there are easy ways to create them. `np.zeros()` and `np.ones()` create new dimensioned arrays filled with the corresponding values.

```
1 array_6 = np.ones((3, 2))
2 array_7 = np.zeros(7)
3 print(array_6)
4 # [[1. 1.]
5 #  [1. 1.]
6 #  [1. 1.]]
7 print(array_7) # [0. 0. 0. 0. 0. 0. 0.]
```

Click and drag to move

The `np.zeros_like()` and `np.ones_like()` functions are similar to the previous ones, but they return an array of ones or zeros with the same shape and type as the given arrays.

Current topic:

[Array creation and indexing](#) ...


Topic depends on:

- ✓ [Slicing](#) ...
- ✓ [If statement](#) Stage 1 16★ ...
- ✗ [Intro to NumPy](#) ...

Table of contents:

- [1 Array creation and indexing](#)
- [§1. Array range and evenly spaced array](#)
- [§2. Arrays filled with ones and zeros](#)
- [§3. Converting NumPy arrays to Python lists](#)
- [§4. Indexing in arrays](#)
- [§5. Slicing in arrays](#)
- [§6. Conclusion](#)
- [Feedback & Comments](#)


```
1 x = np.array([[1, 1, 1], [2, 2, 2]])
2 y = np.array([1, 2, 3, 4, 5])
3 array_8 = np.ones_like(x)
4 array_9 = np.zeros_like(y)
5 print(array_8)
6 # [[1 1 1]
7 #  [1 1 1]]
8 print(array_9) # [0 0 0 0 0]
```

 Click and drag to move

§3. Converting NumPy arrays to Python lists


Finally, when you do not need to work with arrays anymore, you can easily transform them into lists. NumPy provides the `array.tolist()` function.

```
1 array_10 = np.array([[1, 2], [3, 4]])
2 lst1 = array_10.tolist()
3 print(lst1) # [[1, 2], [3, 4]]
```

 Click and drag to move

If we attempt to use the `list(array)` function on `numpy` arrays, it will also return a list but its elements will be `numpy` arrays.

```
1 array_11 = np.array([[5, 2], [7, 4]])
2 lst2 = list(array_11)
3 print(lst2) # [array([5, 2]), array([7, 4])]
4 print(type(lst2[0])) # <class 'numpy.ndarray'>
```

 Click and drag to move

§4. Indexing in arrays

Now, once we know how to create arrays, it is time to learn how to get access to their elements. Just like with Python lists, we can access the elements using indices. If you use a one-dimensional array in your program, there is no difference with how list indices work. However, when you face an n-dimensional array, do not forget about some key operations.

Let's have a look at the example and discuss it.

```
1 array_12 = np.array([[1, 12, 31], [4, 45, 64], [0, 7, 89]])
2 print(array_12[2, 2]) # 89
3 print(array_12[2][2]) # 89
```

 Click and drag to move

First of all, we created a two-dimensional array. Suppose, we are interested in getting the last value. The idea of multidimensional indexing is quite easy: if you have two dimensions, the first value addresses the *row* of your array, the second one addresses the *index of the value* you are searching for. You can see that the result of `array_12[2, 2]` and `array_12[2][2]` is the same. The first case is more efficient, however, because when we write `array_12[2][2]`, a new temporary array is created after the first index that is subsequently indexed by 2.

Similarly, if you have three dimensions, you need to print three values to execute multidimensional indexing, etc. Let's look at the three-dimensional example below.

```
1 array_13 = np.array([[[1, 12, 31], [4, 45, 64], [0, 7, 89]]])
2 print(array_13[0, 1, 1]) # 45
3 print(array_13[0][1][1]) # 45
```


 Click and drag to move

If you print an index that is out of the bounds, it will cause an error in your program.

§5. Slicing in arrays

NumPy arrays can also be sliced like lists in Python. Slicing a one-dimensional array is the same as slicing a Python list, so our main task is to understand how to slice *n-dimensional* arrays. Look at the example below:

```
1 array_14 = np.array([[100, 101, 102],
2                     [103, 104, 105],
3                     [106, 107, 108],
4                     [109, 110, 111]])
5 print(array_14[1:3, 1]) # [104 107]
```

 Click and drag to move

We start by creating a two-dimensional array `array_14` with four rows. When we write `array_14[1:3, 1]`, the first part addresses *the slice of the rows* to be taken into account, then the second part chooses an element of each row. As a result, we have a new one-dimensional array.


Let's have a look at some more examples. `array_15` below is a three-dimensional array. Just like in Python, we can use the *negative* index to choose elements from the end — the last two-dimensional array `[[11, 12, 13, 14, 15], [16, 17, 18, 19, 20]]` in this case. Then we just make a full copy of the array with the help of the `:` operator, and at last, in each row, we choose a particular slice.

```
1 array_15 = np.array([[[1, 2, 3, 4, 5],
2                      [6, 7, 8, 9, 10]],
3                      [[11, 12, 13, 14, 15],
4                      [16, 17, 18, 19, 20]]])
5 print(array_15[-1, :, 1:4])
6 # [[12 13 14]
7 #   [17 18 19]]
```

 Click and drag to move

In `array_16` we extract every row and element with a given step.

```
1 # two-dimensional array
2 array_16 = np.array([[1, 2, 3, 4, 5],
3                     [5, 4, 3, 2, 1],
4                     [6, 7, 8, 9, 10],
5                     [10, 9, 8, 7, 6],
6                     [11, 12, 13, 14, 15]])
7 print(array_16[:, ::2, ::2])
8 # [[ 1  3  5]
9 #   [ 6  8 10]
10 #   [11 13 15]]
```

 Click and drag to move

As you can see, slicing operations are the same as with Python lists. But here, we apply them with the dimensions in mind. This can lead to some errors, especially with high-dimensional arrays, so we need to be very careful.

§6. Conclusion

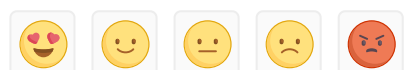
In this topic, we have learned how to:

- create arrays using different methods,
- transform arrays into lists,
- index and slice multidimensional arrays.

Now it's high time to move on to some practical tasks. Practice makes perfect!

 Report a typo

28 users liked this theory. 0 didn't like it. What about you?



Start practicing

Comments (3)

Hints (0)

Useful links (0)

Hide discussion

S

Share something, SOULDEV

▲▼

Post

☐ Place a bounty ?

Sort by: Last posted ▼

S

SOULDEV


in less than a minute

Image die

😊

Reply

Delete



Tomáš

about 2 months ago

Would you mind adding at least another example of slicing a three-dimensional array? It's not entirely clear to me. Otherwise, a gorgeous explanation, thank you very much!

😊

Reply

Report

U9

User 9522037

6 months ago

Can we have some more NumPy topics please - and maybe also some pandas / scipy ?

👍 4

😊

Reply

Report