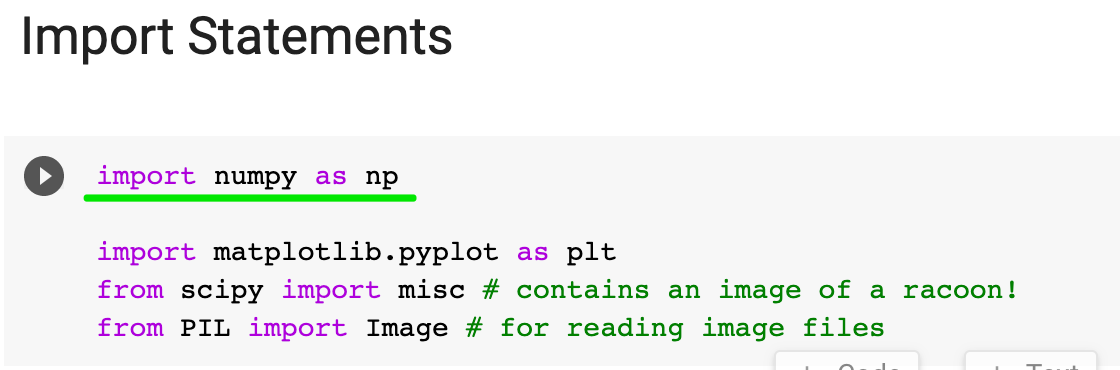
**NumPy's ndarray - Incredible Power at Your Fingertips!**

Let’s import NumPy

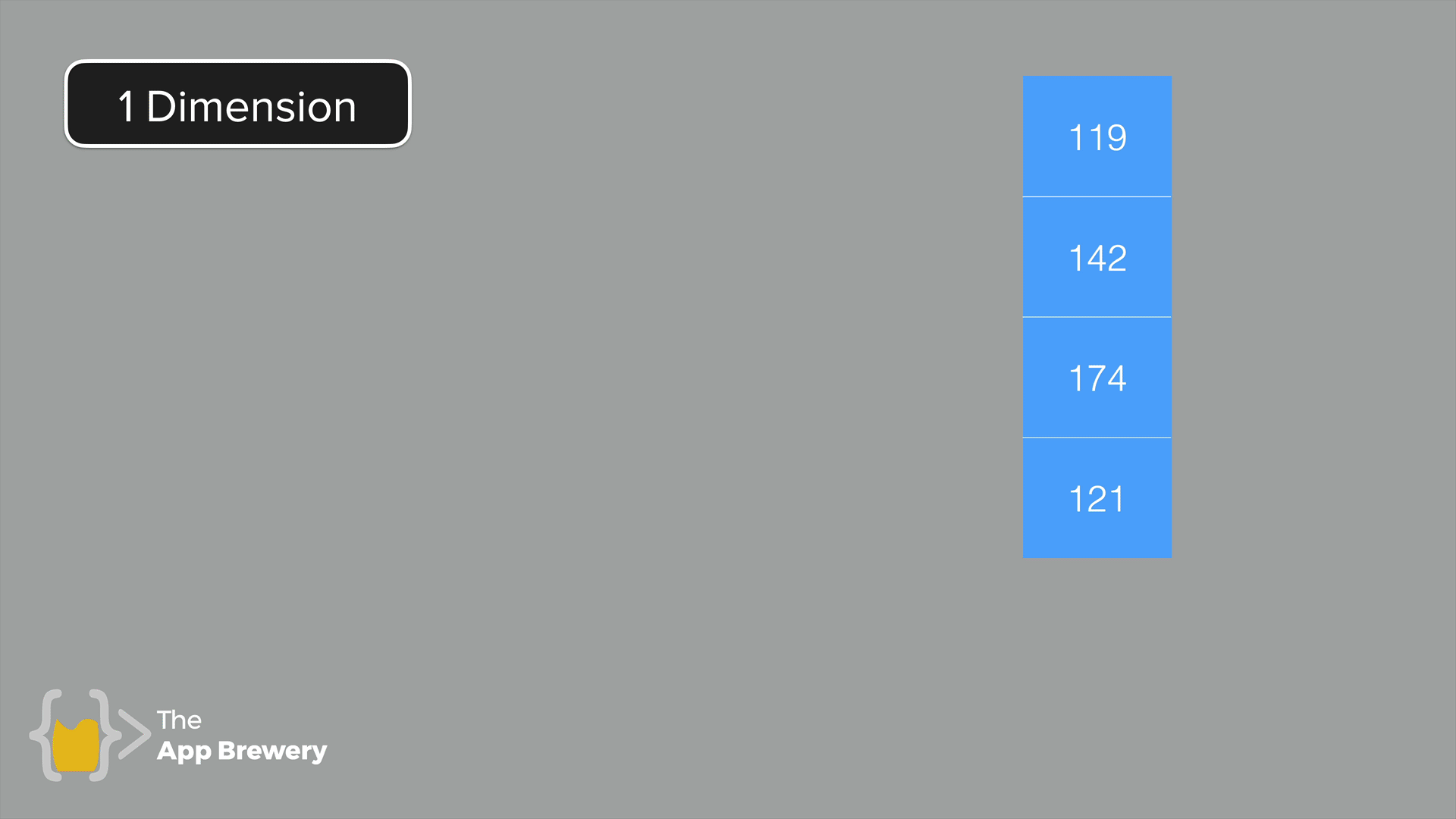


We’ll follow convention and use the name np.

The crown jewel of NumPy is the ndarray. The **ndarray** is a *homogeneous n-dimensional array* object. What does that mean? 🤨

A Python List or a Pandas DataFrame can contain a mix of strings, numbers, or objects (i.e., a mix of different types). **Homogenous** means all the data have to have the same data type, for example all floating-point numbers.

And **n-dimensional** means that we can work with everything from a single column (1-dimensional) to the matrix (2-dimensional) to a bunch of matrices stacked on top of each other (n-dimensional).



1-Dimension

Let’s create a 1-dimensional array (i.e., a “vector”)

1. my\_array = np.array([1.1, 9.2, 8.1, 4.7])

We can see my\_array is 1 dimensional by looking at its shape

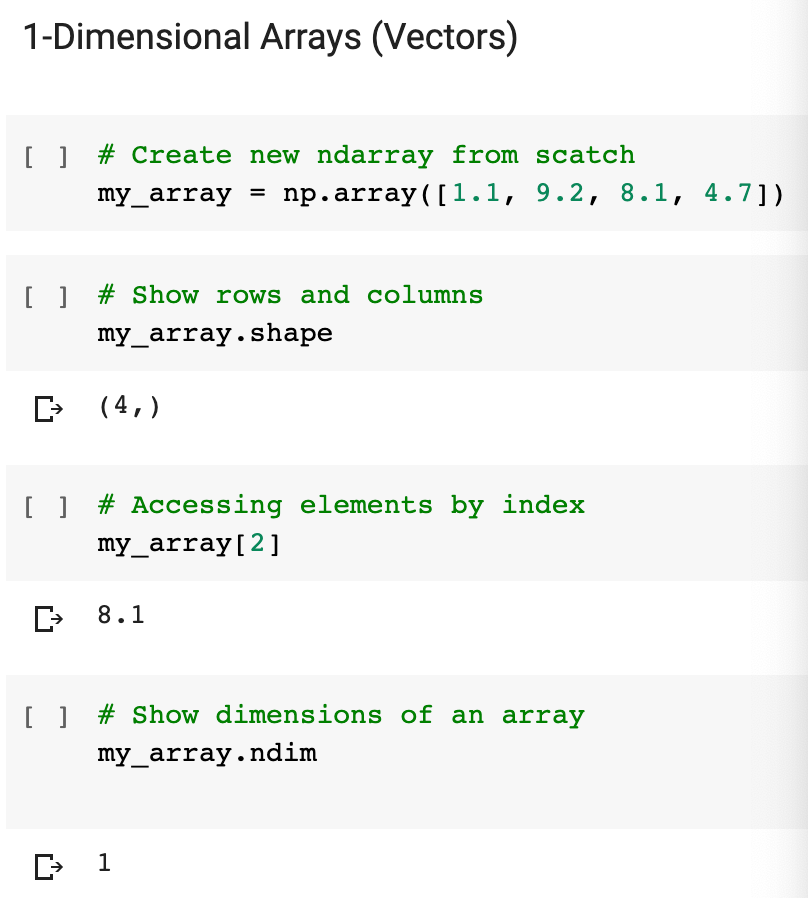
1. my\_array.shape

We access an element in a ndarray similar to how we work with a Python List, namely by that element's index:

1. my\_array[2]

Let’s check the dimensions of my\_array with the ndim attribute:

1. my\_array.ndim



2-Dimensions

Now, let’s create a 2-dimensional array (i.e., a “matrix”)

1. array\_2d = np.array([[1, 2, 3, 9], [5, 6, 7, 8]])

Note we have two pairs of square brackets. This array has 2 rows and 4 columns. NumPy refers to the dimensions as **axes**, so the first axis has length 2 and the second axis has length 4.

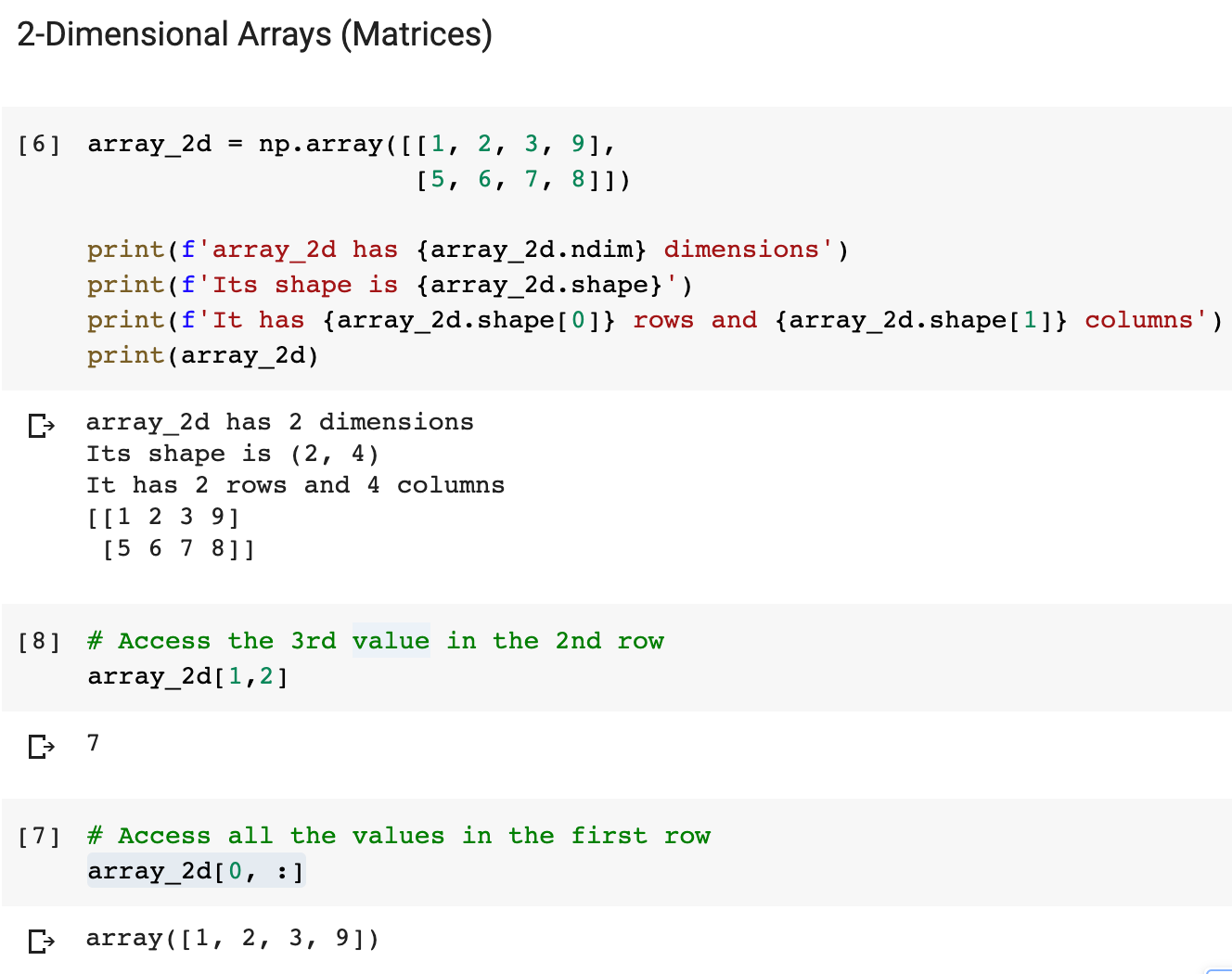
1. print(f'array\_2d has {array\_2d.ndim} dimensions')
2. print(f'Its shape is {array\_2d.shape}')
3. print(f'It has {array\_2d.shape[0]} rows and {array\_2d.shape[1]} columns')
4. print(array\_2d)

Again, you can access a particular row or a particular value with the square bracket notation. To access a particular value, you have to provide an index for each dimension. We have two dimensions, so we need to provide an index for the row and for the column. Here’s how to access the 3rd value in the 2nd row:

1. array\_2d[1,2]

To access an entire row and all the values therein, you can use the : operator just like you would do with a Python List. Here’s the entire first row:

1. array\_2d[0, :]



N-Dimensions

An array of 3 dimensions (or higher) is often referred to as a ”tensor”. Yes, that’s also where Tensorflow, the popular machine learning tool, gets its name. A tensor simply refers to an n-dimensional array. Using what you've learned about 1- and 2-dimensional arrays, can you apply the same techniques to tackle a more complex array?

**Challenge**

* How many dimensions does the array below have?
* What is its shape (i.e., how many elements are along each axis)?
* Try to access the value 18 in the last line of code.
* Try to retrieve a 1-dimensional vector with the values [97, 0, 27, 18]
* Try to retrieve a (3,2) matrix with the values [[ 0, 4], [ 7, 5], [ 5, 97]]

1. mystery\_array = np.array([[[0, 1, 2, 3],
2. [4, 5, 6, 7]],
4. [[7, 86, 6, 98],
5. [5, 1, 0, 4]],
7. [[5, 36, 32, 48],
8. [97, 0, 27, 18]]])

.

.

..

...

..

.

.

**Solution: Working with Higher Dimensions**

This is really where we have to start to wrap our heads around how ndarrays work because it takes some getting used to the notation.

The ndim and shape attributes show us the number of dimensions and the length of the axes respectively.

1. print(f'We have {mystery\_array.ndim} dimensions')
2. print(f'The shape is {mystery\_array.shape}')

The shape is (3, 2, 4), so we have 3 elements along axis #0, 2 elements along axis #1 and 4 elements along axis #3.

To access the value 18 we, therefore, have to provide three different indices - one for each axis. As such, we locate the number at index 2 for the first axis, index number 1 for the second axis, and index number 3 for the third axis.

1. mystery\_array[2, 1, 3]

The values [97, 0, 27, 18] live on the 3rd axis and are on position 2 for the first axis and position 1 on the second axis. Hence we can retrieve them like so:

1. mystery\_array[2, 1, :]

Finally, to retrieve all the first elements on the third axis, we can use the colon operator for the other two dimensions.

1. mystery\_array[:, :, 0]

With the square brackets serving as your guide, the ndarray is quite difficult to visualise for 3 or more dimensions. So if any of this was unclear or confusing. Pause on this lesson for a minute and play around with the array above. Try selecting different subsets from the array. That way you can get comfortable thinking along the different dimensions of the ndarray.

