**Broadcasting, Scalars and Matrix Multiplication**

Linear Algebra with Vectors

NumPy is designed to do math (and do it well!). This means that NumPy will treat vectors, matrices and tensors in a way that a mathematician would expect. For example, if you had two vectors:

1. v1 = np.array([4, 5, 2, 7])
2. v2 = np.array([2, 1, 3, 3])

And you add them together

1. v1 + v2

The result will be a ndarray where all the elements have been added together.

array([ 6, 6, 5, 10])

In contrast, if we had two Python Lists

1. list1 = [4, 5, 2, 7]
2. list2 = [2, 1, 3, 3]

adding them together would just concatenate the lists.

1. list1 + list2
2. # output: [4, 5, 2, 7, 2, 1, 3, 3]

Multiplying the two vectors together also results in an element by element operation:

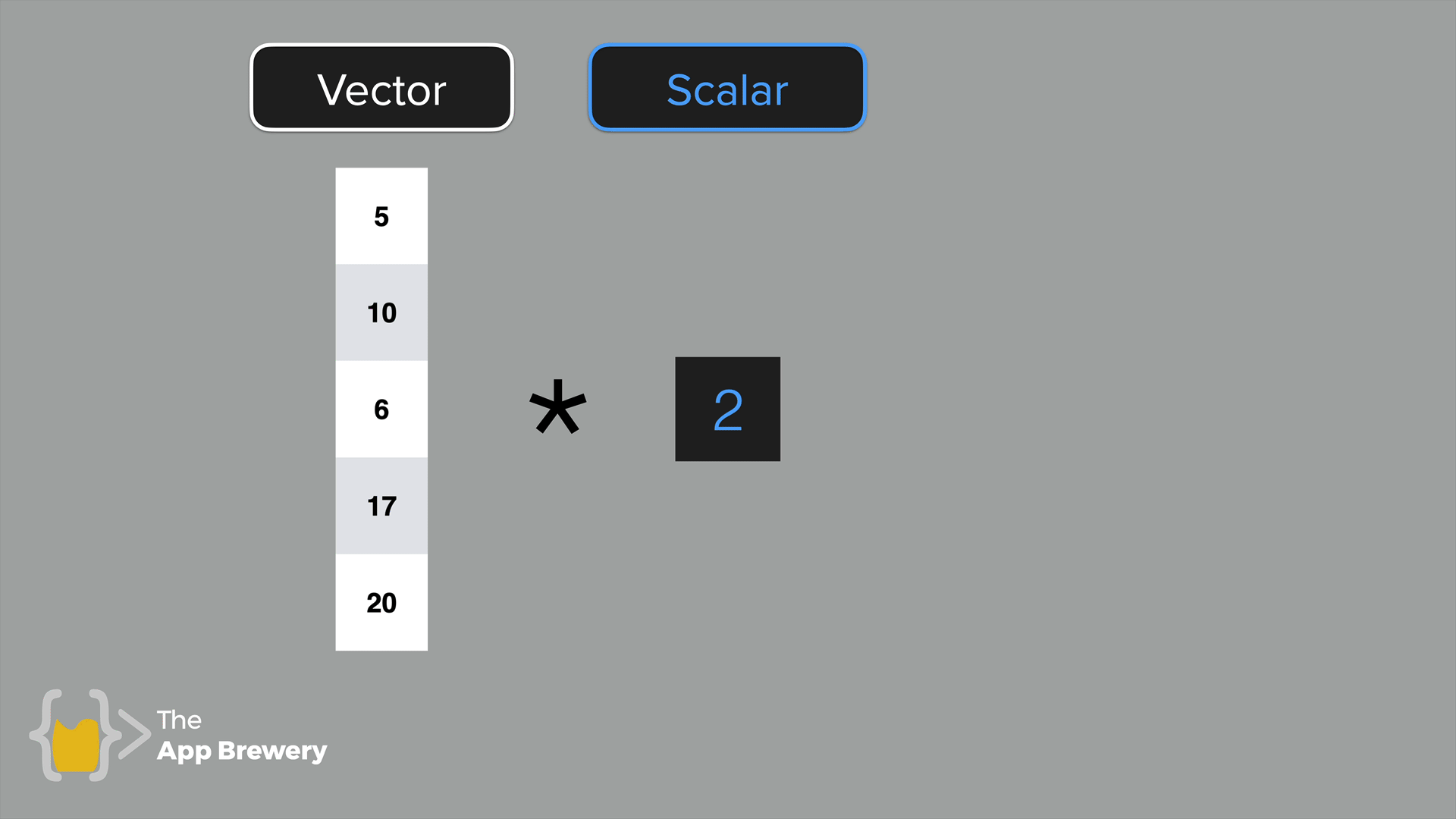
1. v1 \* v2

Gives us array([ 8, 5, 6, 21]) since 4x2=8, 5x1=5 and so on. And for a Python List, this operation would not work at all.

1. list1 \* list2 # error!

Broadcasting

Now, oftentimes you'll want to do some sort of operation between an array and a single number. In mathematics, this single number is often called a **scalar**. For example, you might want to multiply every value in your NumPy array by 2:

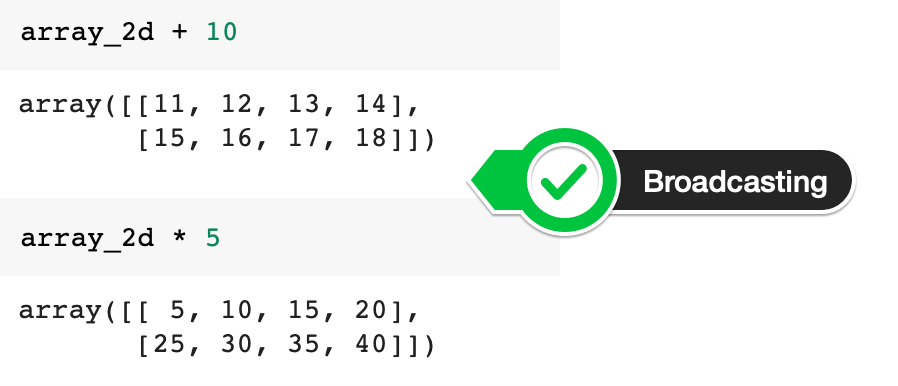


In order to achieve this result, NumPy will make the shape of the smaller array - our scalar - compatible with the larger array. This is what the documentation refers to when it mentions the term "broadcasting".

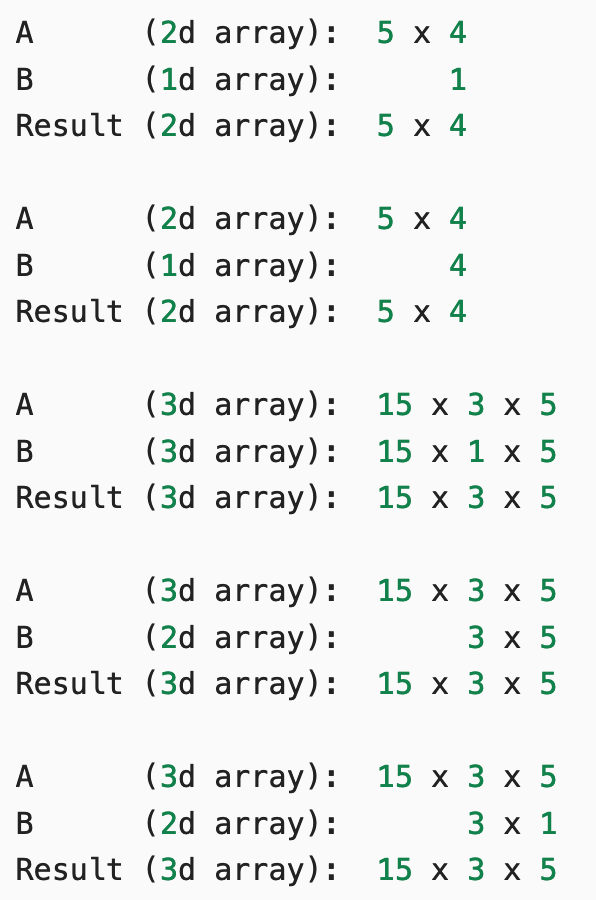
The same rules about 'expanding' the smaller ndarray hold true for 2 or more dimensions. We can see this with a 2-Dimensional Array:

1. array\_2d = np.array([[1, 2, 3, 4],
2. [5, 6, 7, 8]])

The scalar operates on an element by element basis.

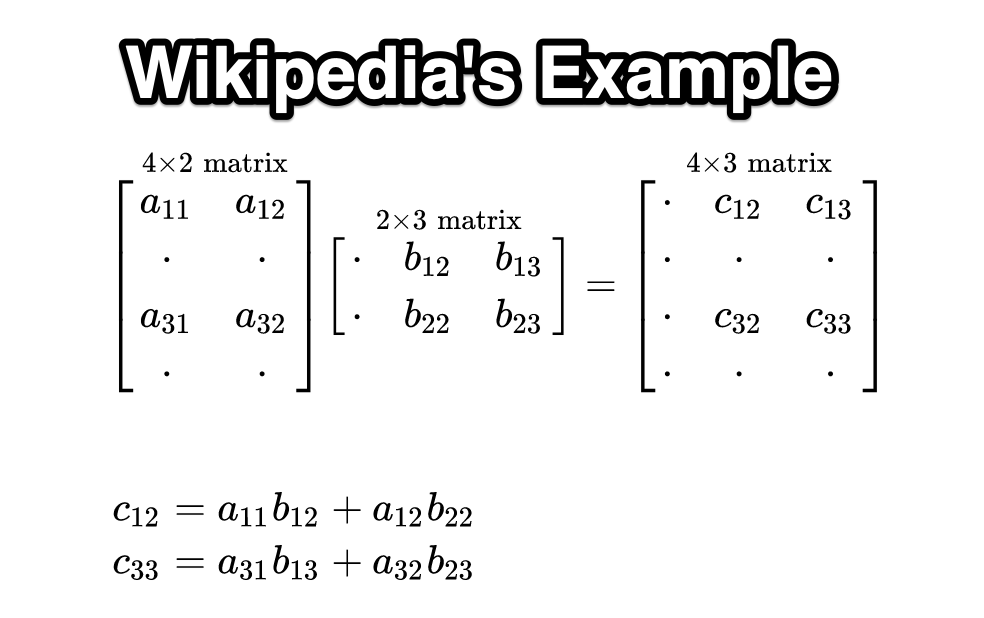


The documentation on broadcasting also shows us a few more examples:



Matrix Multiplication

But what if we're not multiplying our ndarray by a single number? What if we multiply it by another vector or a 2-dimensional array? In this case, we follow [the rules of linear algebra](https://en.wikipedia.org/wiki/Matrix_multiplication#Illustration).



1. a1 = np.array([[1, 3],
2. [0, 1],
3. [6, 2],
4. [9, 7]])
6. b1 = np.array([[4, 1, 3],
7. [5, 8, 5]])

**Challenge:**Let's multiply a1 with b1. Looking at the Wikipedia example above, work out the values for c12 and c33 on paper. Then use the [.matmul()](https://numpy.org/doc/stable/reference/generated/numpy.matmul.html) function or the @ operator to check your work.

.

.

..

...

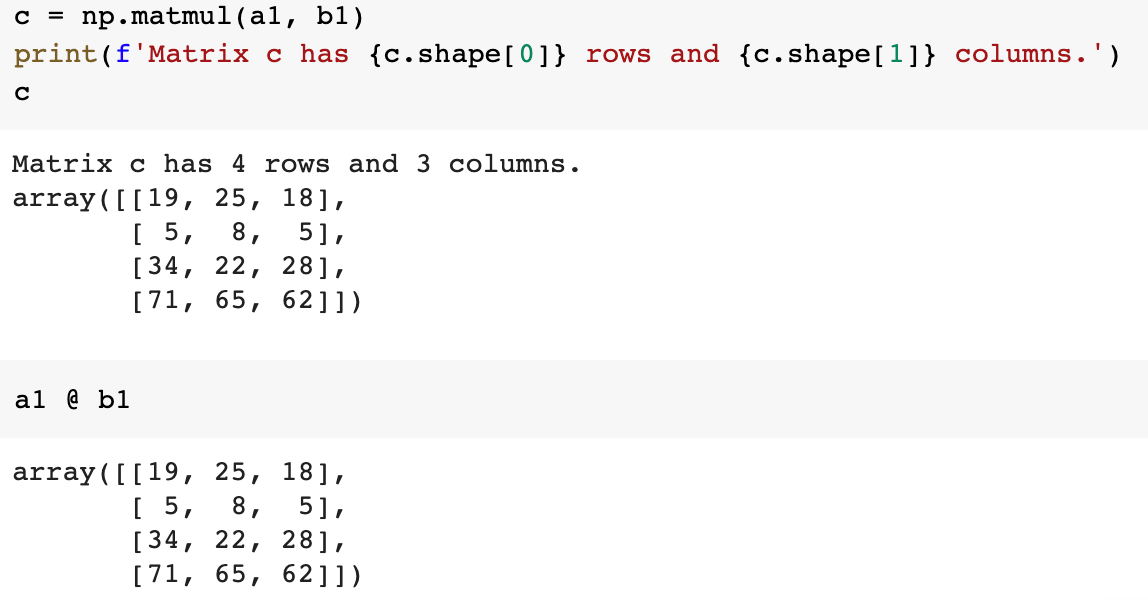
..

.

.

**Solution: Matrix multiplication with NumPy**

The solution code is pretty straightforward



But how did the calculations arrive at 25 for c12 and 28 for c33? Substituting the number into the formula we get:

c12 = 1\*1 + 3\*8 = 1 + 24 = 25

c33 = 6\*3 + 2\*5 = 18 + 10 = 28