Numpy

(Numerical Python)

Why Numpy ?

NumPy

- Designed for efficiency on large arrays of data.
- stores data in a contiguous block of memory
- uses much less memory than Python lists

NumPy ndarray vs list

- One of the key features of NumPy is its N-dimensional array object, or ndarray, which is a fast, flexible container for large datasets in Python.
- Whenever you see “array,” “NumPy array,” or “ndarray” in the text, with few exceptions they all refer to the same thing: the ndarray object.

Creating ndarrays

Using list of lists

```
import numpy as np

data2 = [[1, 2, 3, 4],
         [5, 6, 7, 8] ]

arr = np.array(data2)
arr.ndim          ----->   # 2
arr.shape         ------> # (2,4)
arr.size          ------> 2*4 = 8
```

NdArrays

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

```
arr = ( [
          [ [1,2,3],
            [4,5,6] ] ,
          [ [1,2,3],
            [4,5,6] ]
        ] )
```

np.ndim ???

np.shape ???

</> CODE

Creating ndarrays

```
array = np.zeros((2,3))
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
array = np.ones((2,3))
[[1. 1. 1.]
 [1. 1. 1.]]
```

</>CODE

```
array = np.arrange(start , end, step_size)
array = np.arange(0, 10, 2)
[0, 2, 4, 6, 8]
```

```
array = np.random.randint(0, 10, (3,3))
[[6 4 3]
 [1 5 6]
 [9 8 5]]
```

Element-Wise Operations

- Can we perform this in python ?
- arr1 = [1,2,3]
- arr2 = [2,3,4]

arr1 * arr2

Let’s see

Arithmetic with NumPy Arrays

- Any arithmetic operations between equal-sized arrays applies the operation element-wise:

```
arr = np.array(
[[1., 2., 3.],
 [4., 5., 6.]]
)
print(arr * arr)
[[ 1.  4.  9.]
 [16. 25. 36.]]
print(arr - arr)
[[0. 0. 0.]
 [0. 0. 0.]]
```

Arithmatic with NumPy Arrays

- Comparisons between arrays of the same size yield boolean arrays:

```
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
print(arr2)
[[ 0.  4.  1.]
 [ 7.  2. 12.]]

print(arr2 > arr)
[[False  True False]
 [ True False  True]]
```

Slicing

```
arr = np.arange(10)
print(arr)          # [0 1 2 3 4 5 6 7 8 9]

arr [ start , end , step_size] ( step_size by default 1)

print(arr[5:8]) #[5 6 7]

arr[5:8] = 12
print(arr)      #[ 0 1 2 3 4 12 12 12 8 9]
```

</> CODE

Slicing

- As we know that the **end** is exclusive, so if we have:

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

arr [9:0:-1]

It will give us : [9, 8, 7, 6, 5, 4, 3, 2, 1] # 0 is not included

I want to get the 0th element as well i.e.

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

What will be the result if instead of

[9,0,-1]

I write

[9,-1,-1]

Will I get the 0th element ?

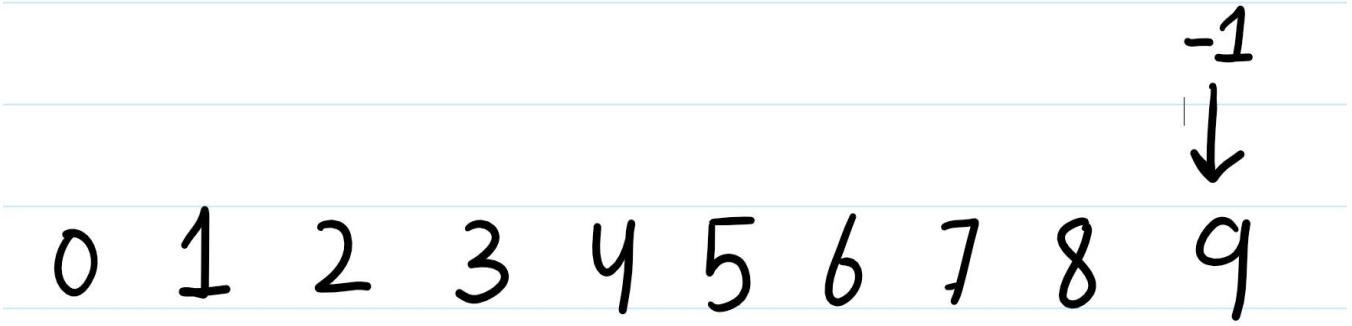
We will get :

[] (an empty numpy array)

Why ?

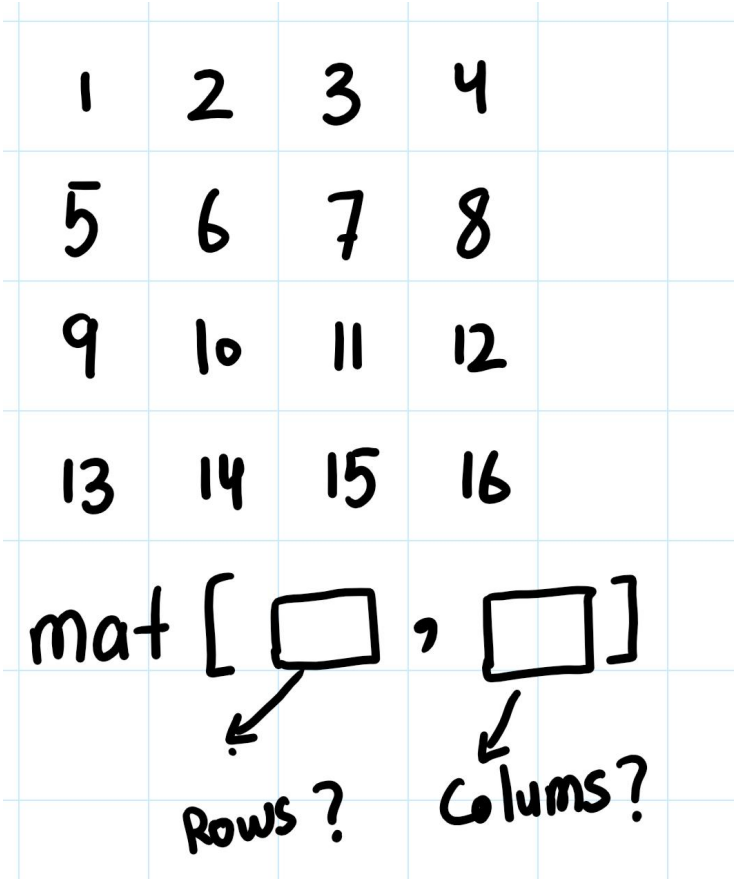
We have [9:-1:-1]

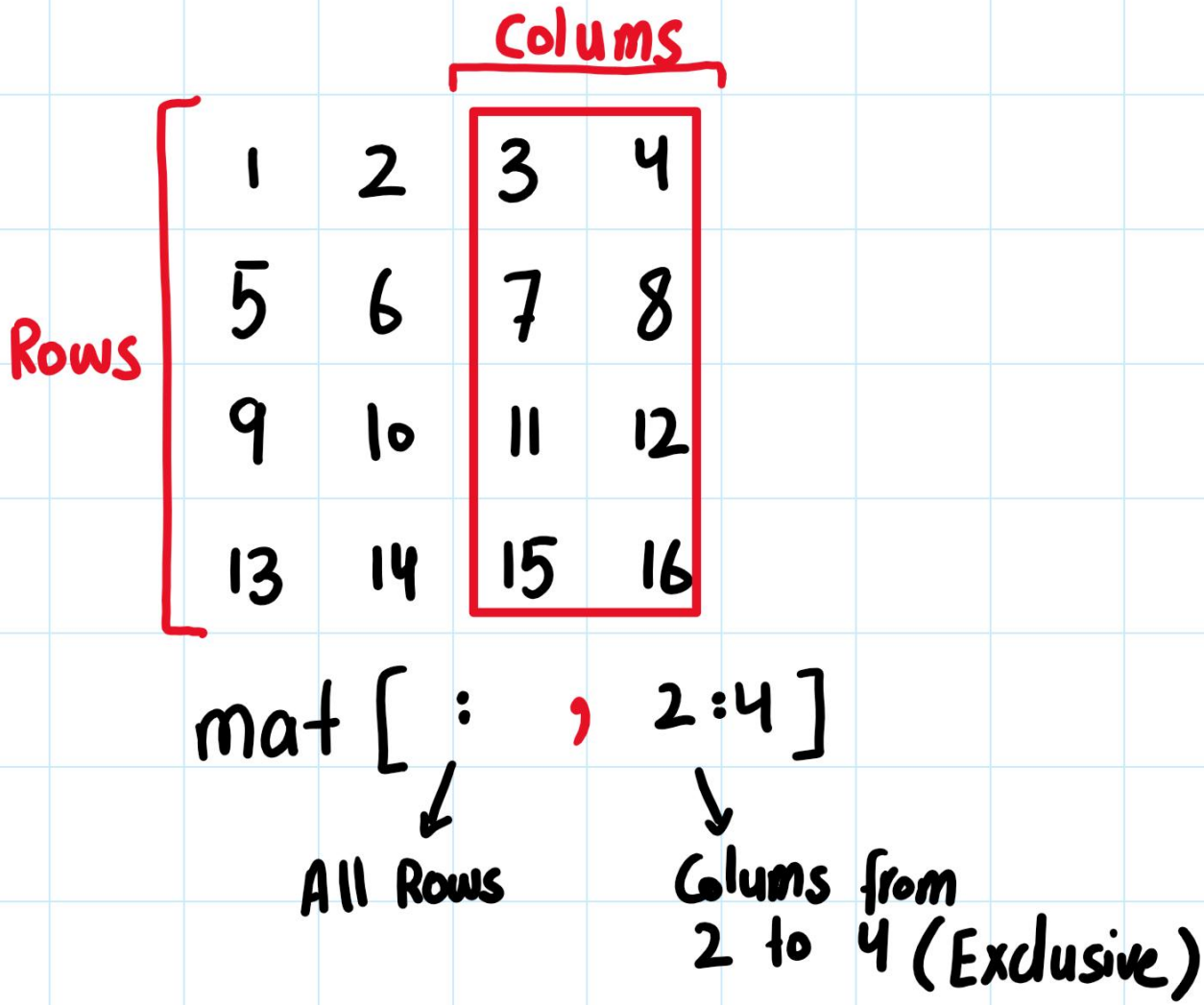
Due to negative indexing , it is considering that index -1 to be :

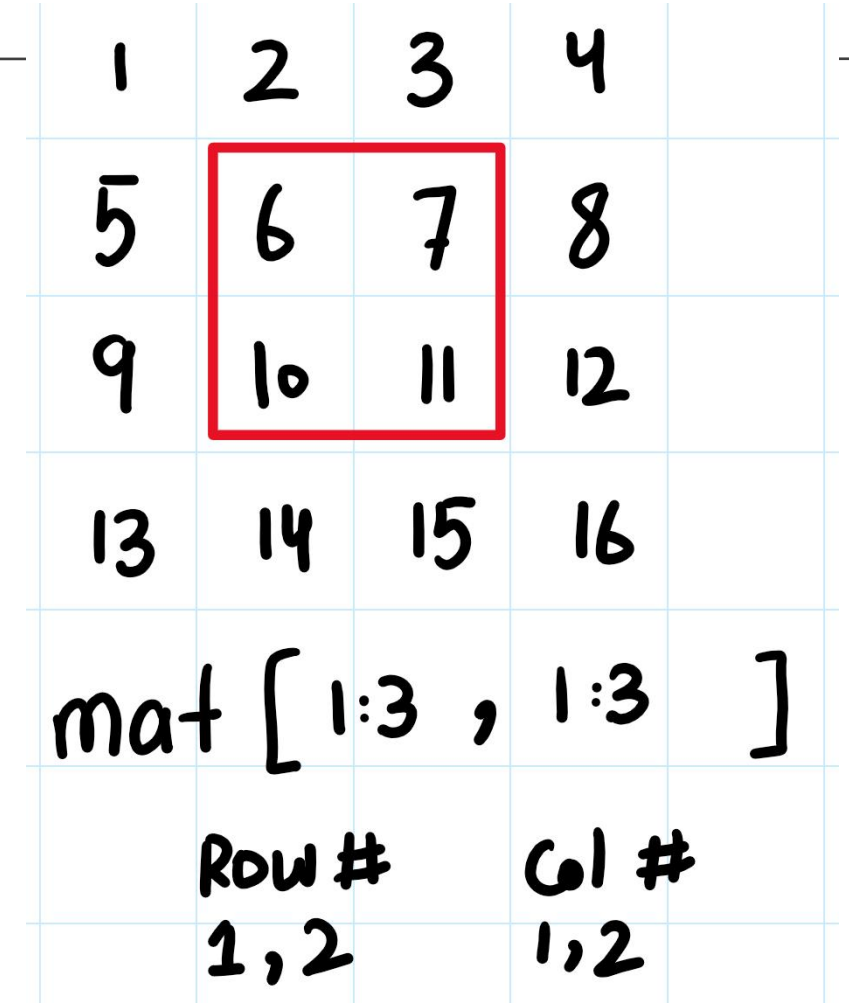


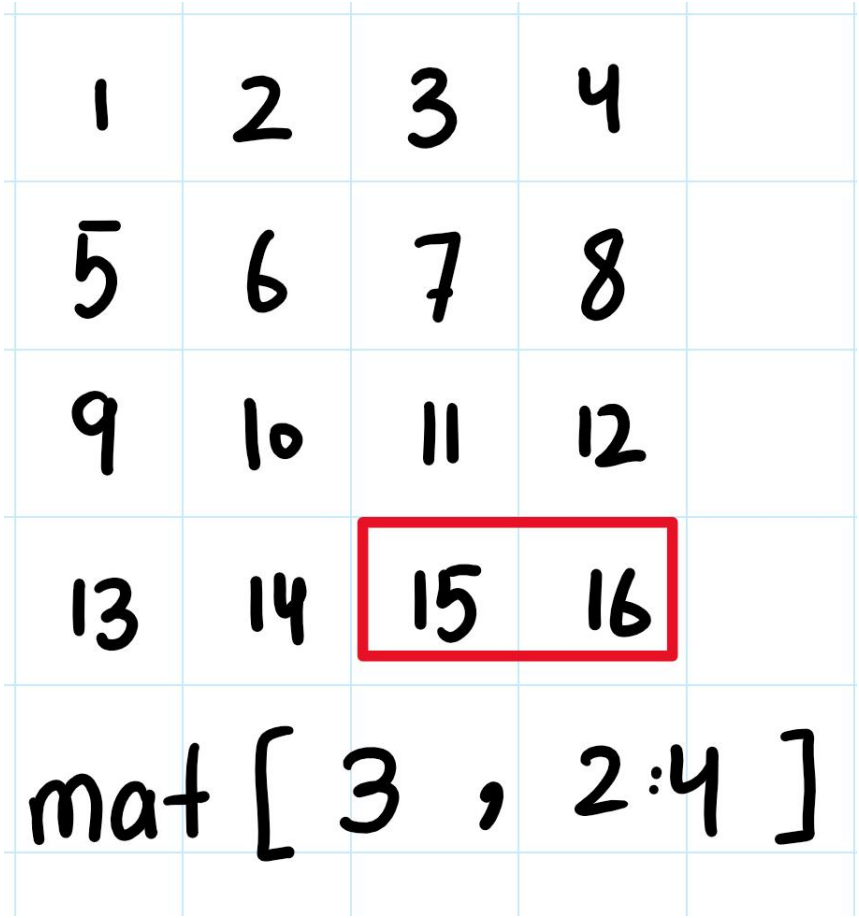
(We provided the same start and end points)

Slicing in multi-dimensional arrays



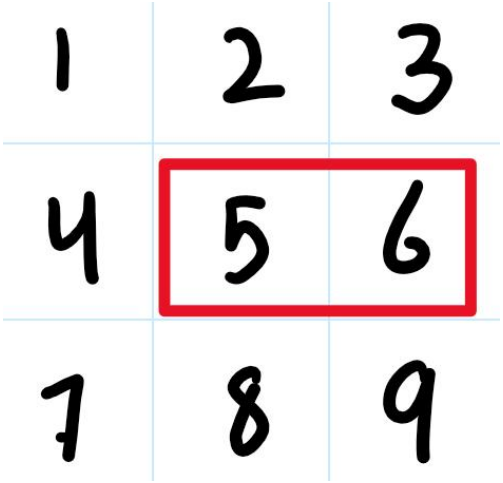

Colums
Rows
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
mat [: , 2:4]
All Rows
Colums from
2 to 4 (Exclusive)


1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
mat [1:3 , 1:3]
Row #
1,2
Col #
1,2


1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
mat [3 , 2:4]

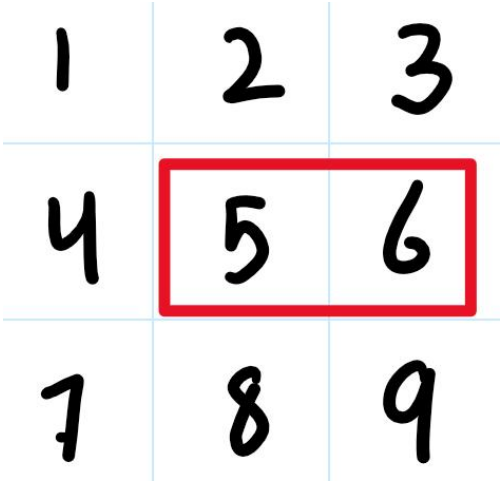
Activity

How can we slice it
to display the last 2 elements
of middle array ?



Activity

How can we slice it
to display the last 2 elements
of middle array?



mat [1 , 1:3]

Activity

Activity

</> CODE

Broadcasting

- We did the following earlier in numpy (Element-wise operation) :
- [1,2,3] * [2,3,4] (np arays)

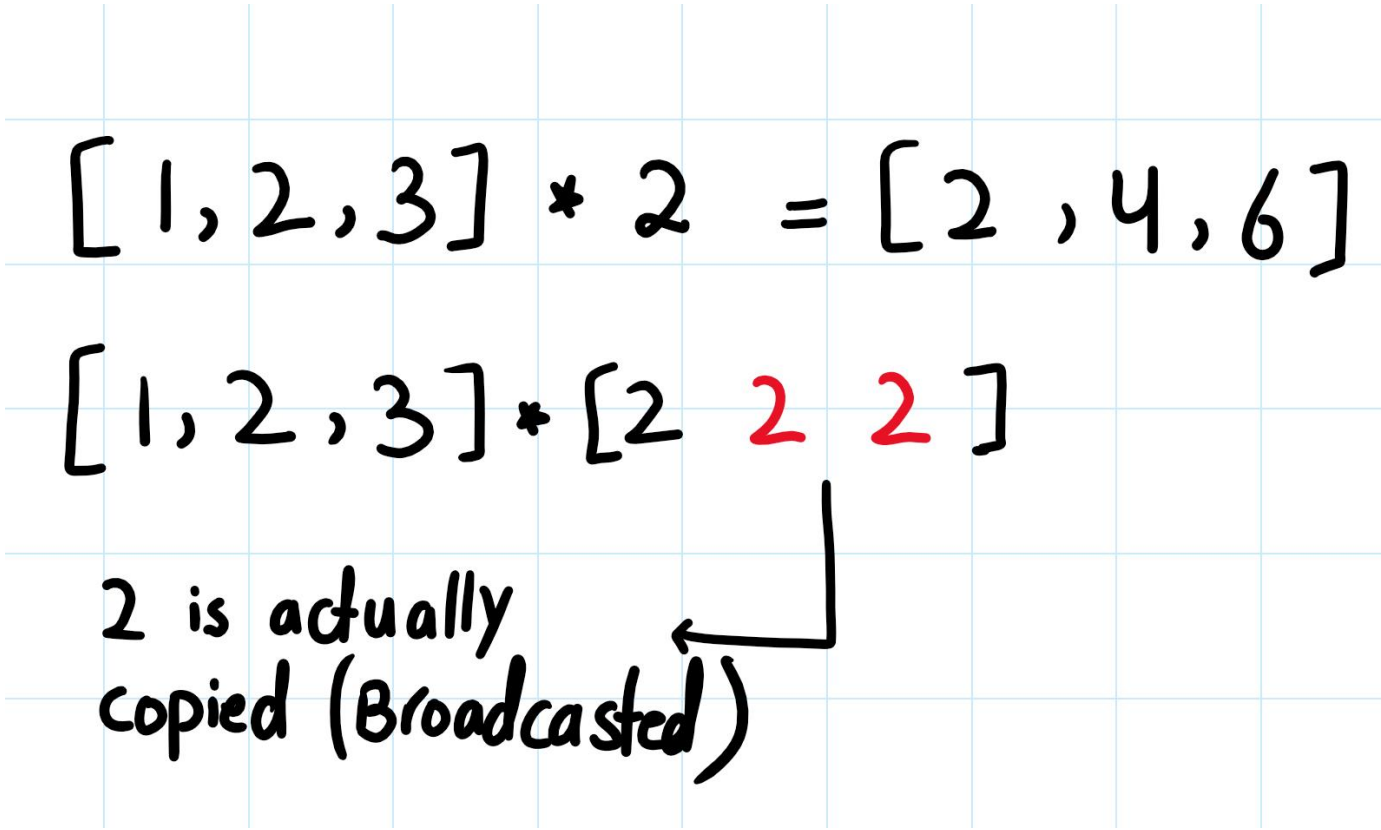
What if we have this scenario :

[1,2,3] * 2

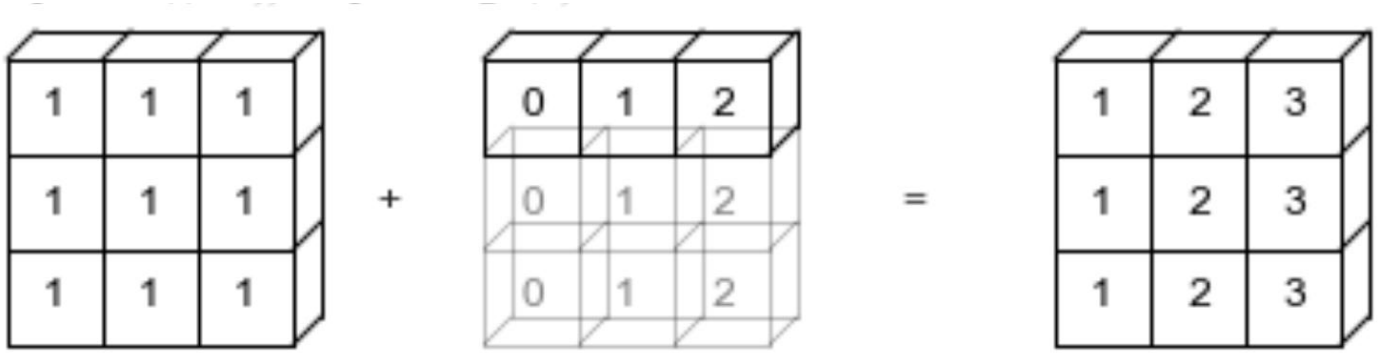
Let’s first see what happens with python lists.

</> CODE

Broadcasting



(3,3) (1,3) (3,3)



Examples

Rules

1) Dimensions of the array A and B are compared from **right to left.**

2) If there is a mismatch , either of them **should be 1**.

3) If there is **no dimension** , it is supposed to be 1

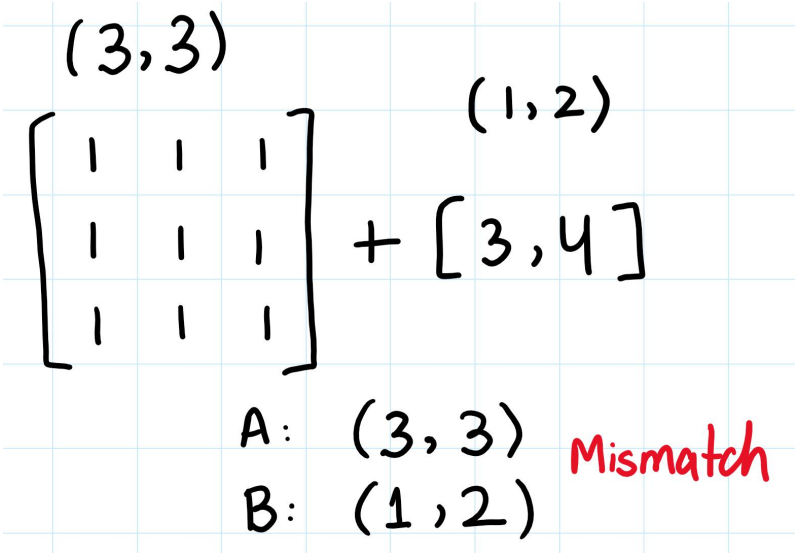
```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):      3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):      3 x 1
Result (3d array):  15 x 3 x 5
```

Examples (Failure)

Example 1:

```
A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

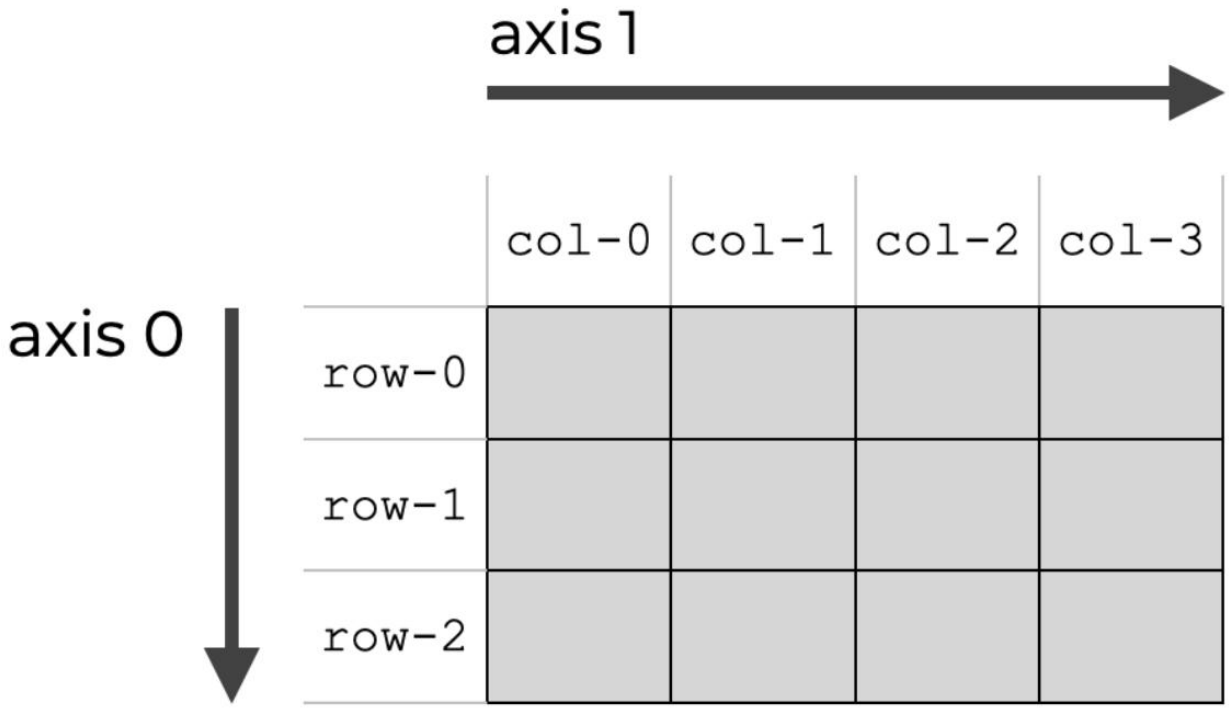
Example 2:



Mathematical Functions

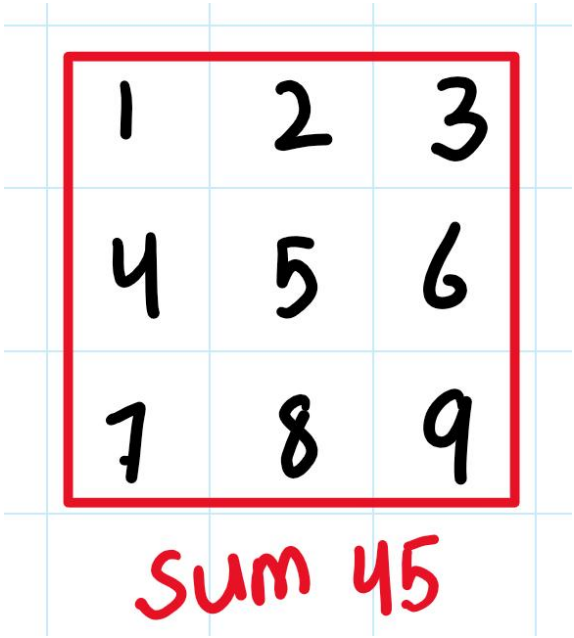
- np.sum()
- np.mean()
- np.log()
- np.exp() # raise to power
- np.min()
- np.max()

Axes in Numpy

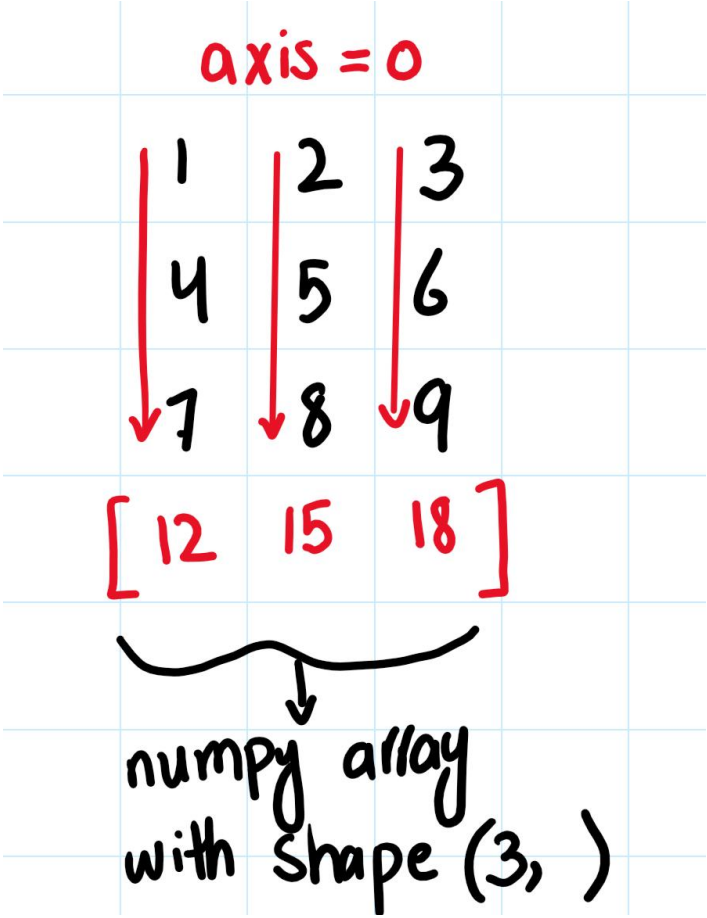


np.sum() with axes

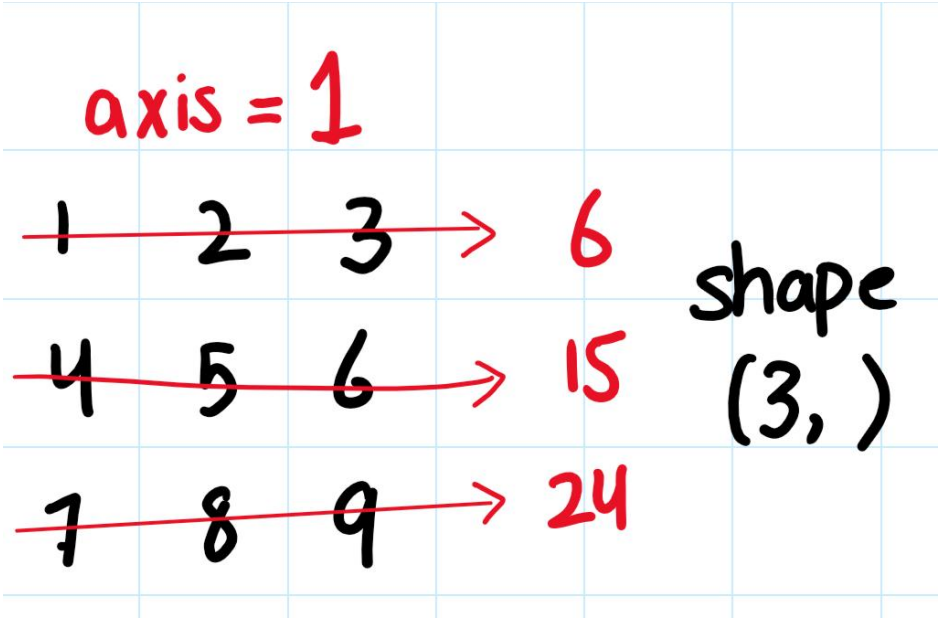
np.sum(my_matrix)



np.sum(mat , axis = 0)



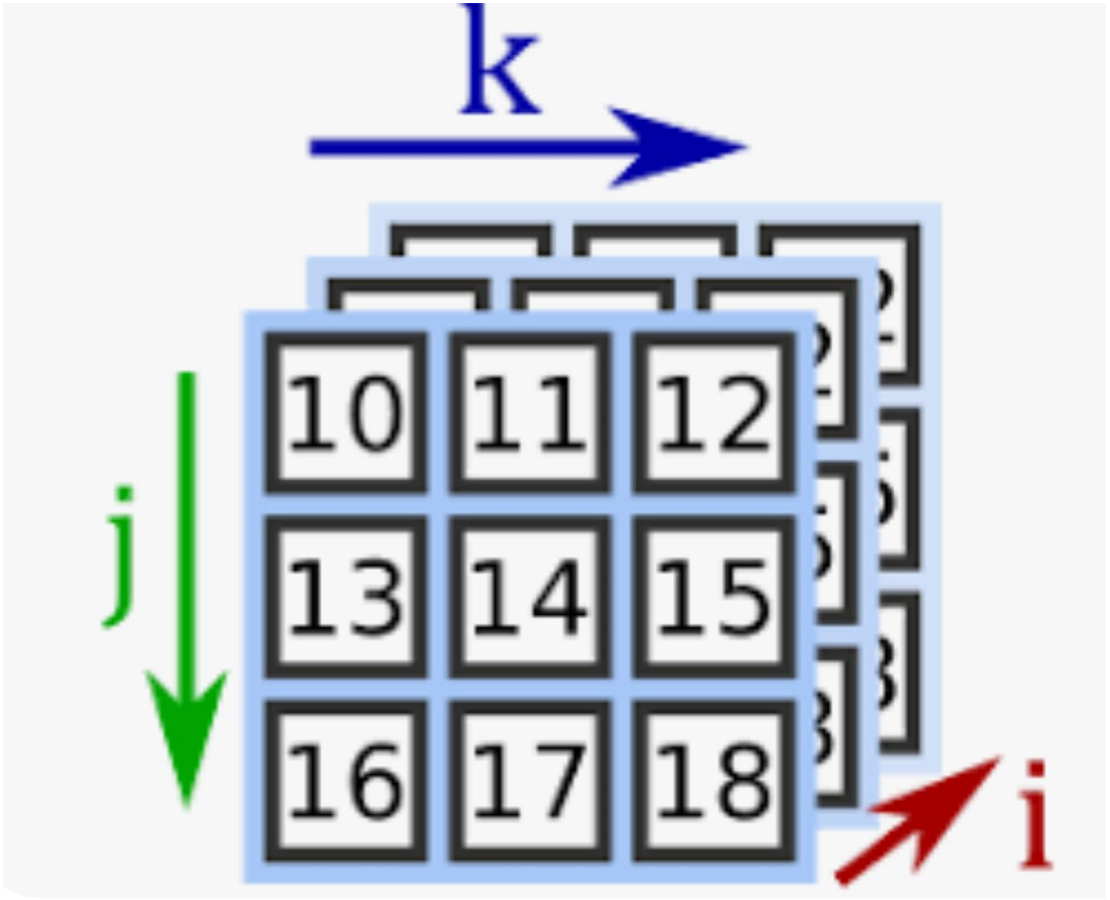
np.sum(mat , axis = 1)

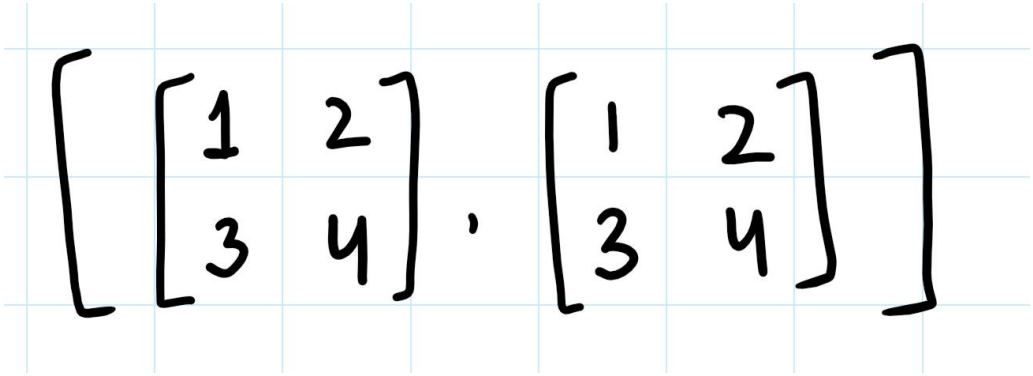


Similary , you can use :

np.mean(mat , axis=?)
np.min(mat , axis=?)
etc....

Axis in 3D





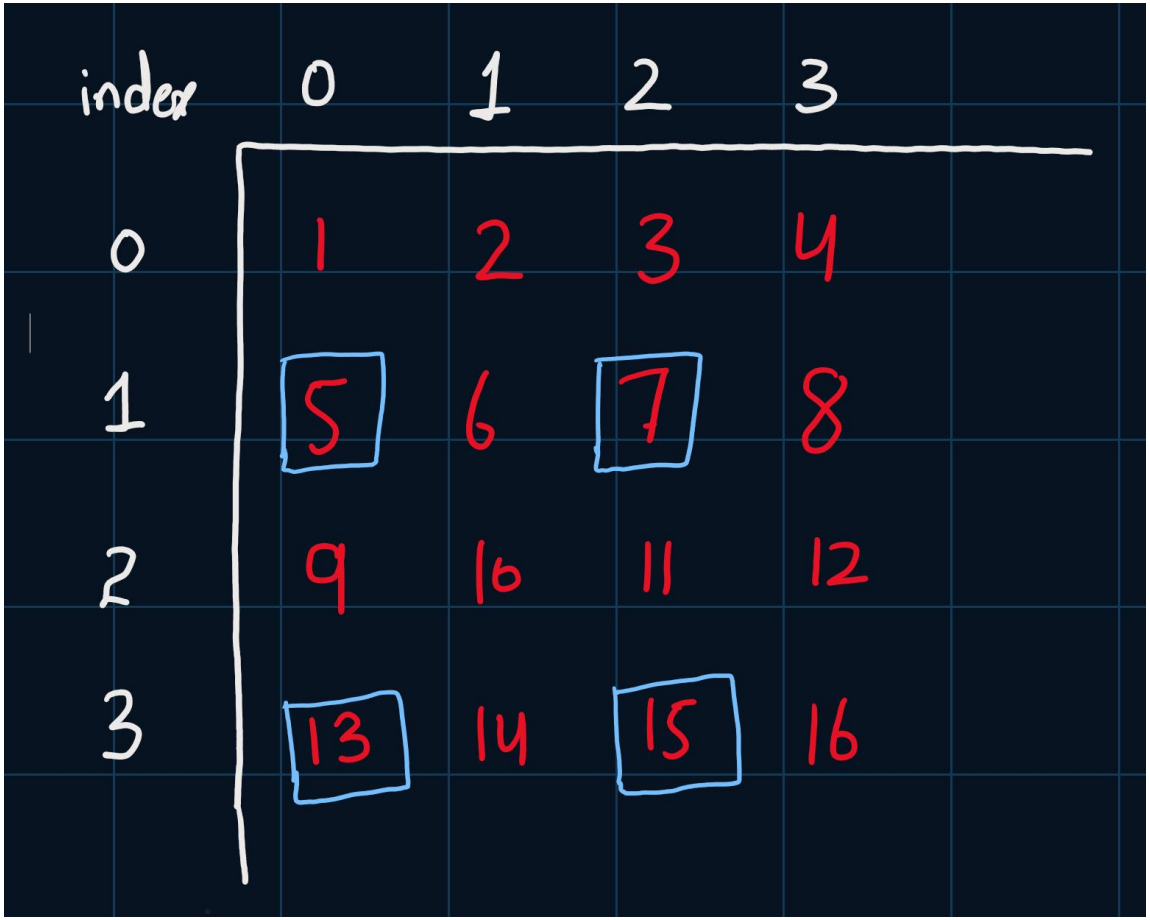
What would be the result of following :

```
np.sum(cube , axis=0)
np.sum(cube , axis=1)
np.sum(cube , axis=2)
```

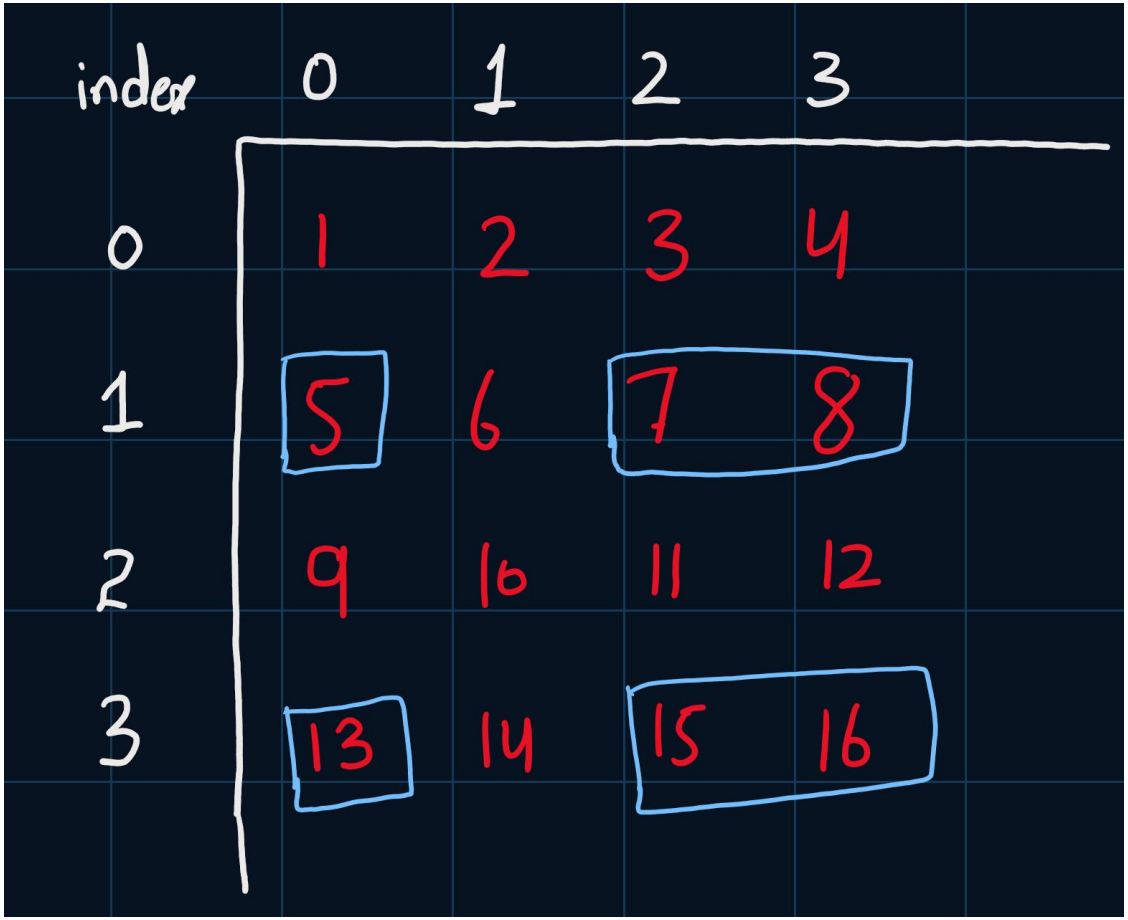
Quiz !!!

Take out a sheet

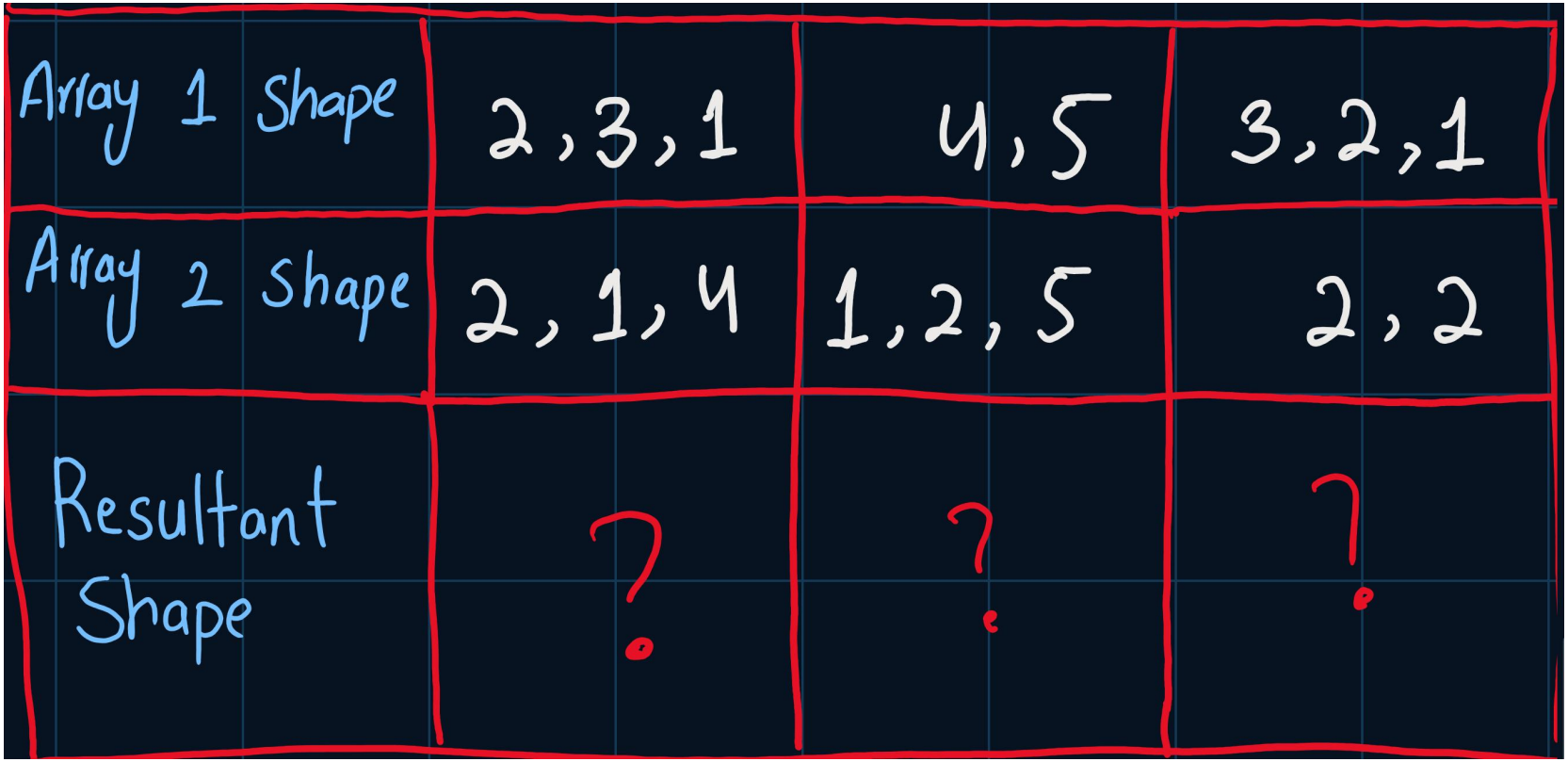
Question 1 : How can we do slicing to get these elements ?



Question 2



Question 3: **BroadCasting Result ? .**



Question 4 : Sum across an axis ?