NUMPY

What are axes?

For (7,) 7 columns
For (2,3) 2 rows and 7 columns
For (2,3,4) 2 depth 3 rows and 4 columns

Note the outermost square brackets refer to the first dimension and so on

If for a thing like np.sum(axis = 0) i do then it sums along that axis ie that axis is deleted from the array
So (3,4) becomes (3,) if axis = 1
And (3,4) becomes (4,) if axis = 0
Note if you want it to instead of deleting make the dimension to 1 then use keepdims = True

Broadcasting?
Trick to understand broadcasting
(N,1,2) - (K,2) = (N,K,2)
It kinda takes lcm of the two corresponding digits basically to give final size.

Imp functions

a = np.array([1, 2, 3])
b = a.copy() #independent copy
c = a[1:] #not independent copy so any changes in c will reflect in a

You can use .base to figure out
b.base is None while c.base is a

np.arange(start, stop, step) step tells spes between each number
np.linspace(start, stop, num) num is number of elements

Slicing:-

start : stop : step for each dimnesion (:step is optional)
Use just : to denote all elements of a dimension
Can use … to denote all the other dimensions
Reshaping:-

```
arr = np.array([[1, 2], [3, 4]])
flat = arr.flatten() # gives [1,2,3,4] return copy
rav = arr.ravel() # same but returns view so changes in rav affect arr

transposed = arr.T
np.transpose(arr, axes=(1, 0, 2)) # as transposing is just exchanging axes

a = np.array([[[1], [2], [3]]])   # shape: (1, 3, 1)
squeezed = np.squeeze(a)         # shape: (3,)

a = np.array([1, 2, 3])     # shape: (3,)
b = np.expand_dims(a, axis=0)  # shape: (1, 3)
b = np.expand_dims(a, axis=1)  # shape: (3, 1)

arr = np.array([1, 2, 3, 4, 5, 6])
reshaped = arr.reshape(2, 3) #gives [[1 2 3] [4 5 6]]

arr = np.array([1, 2, 3, 4, 5])
resized = np.resize(arr, (3, 3))
# Output:
# [[1 2 3]
#  [4 5 1]
#  [2 3 4]]
```

Misc

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = np.concatenate((a, b)) # [1 2 3 4 5 6]

a2 = np.array([[1, 2], [3, 4]])
b2 = np.array([[5, 6], [7, 8]])
result_2d = np.concatenate((a2, b2), axis=1)
# [[1 2 5 6]
#  [3 4 7 8]]
```

The way it works is that it will add up the dimension number along that axis SO if we have (2,a,4) and (2,b,4) and add along axis 1 then we get (2,a+b,4)

Note: You cannot concatenate arrays with different shapes unless they are aligned along the specified axis. So remain dimensions have to be same

stack creates a new dimension at a index and stacks along that
Eg stacked_col = np.stack((a, b), axis=1)

Split does even spliting
split_result = np.split(a, 3)
print(split_result)
# [array([1, 2]), array([3, 4]), array([5, 6])]

Array_split is similar to original BUT similar to `split()`, but allows **uneven splits** when the array cannot be divided exactly. Remaining elements are distributed as evenly as possible.

WHERE imp
np.where(condition, x, y)
At position where condition is true you get x and where wrong you get y

sort()
np.sort(arr, axis)

Boolean indexing is indexing via a boolean array of same shape of array