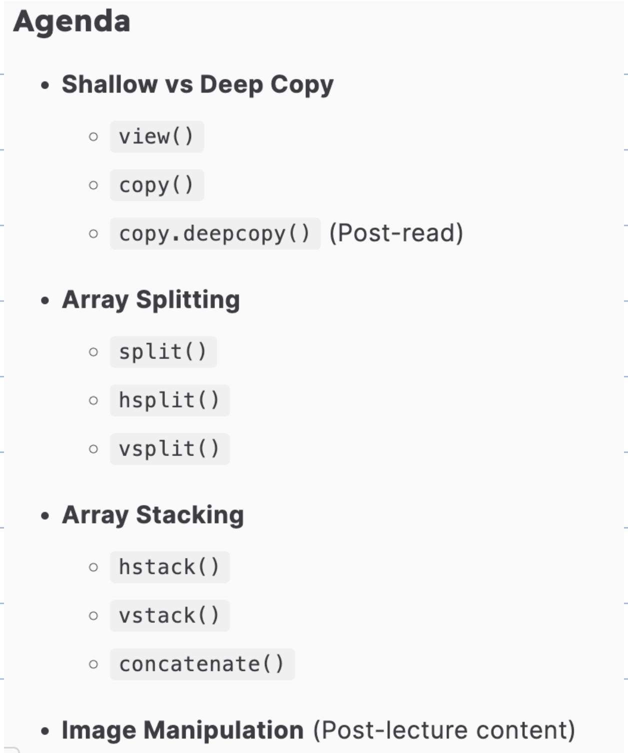
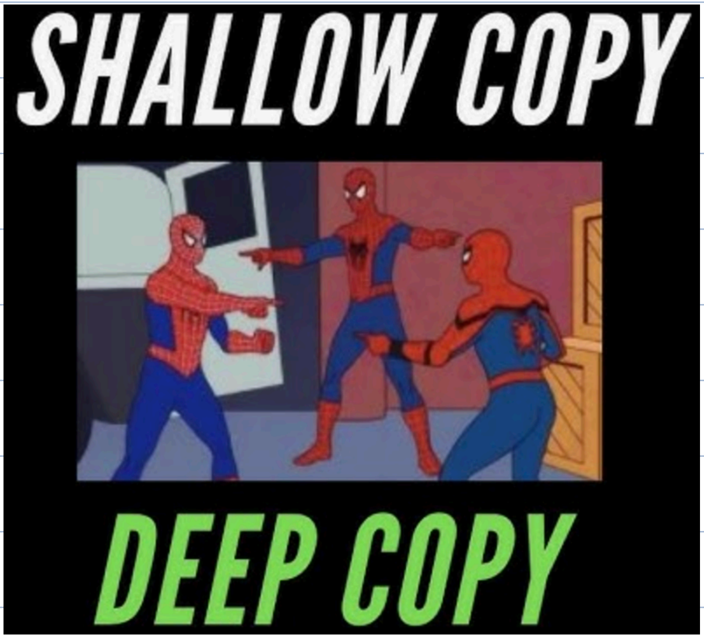
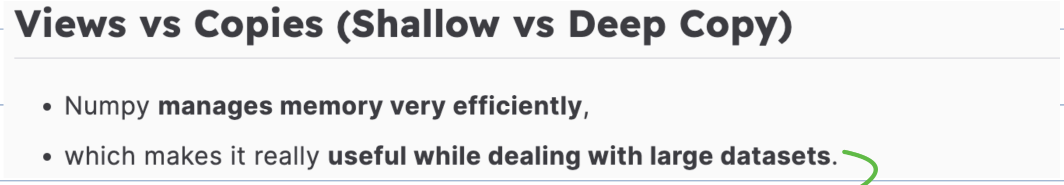


Agenda

- **Shallow vs Deep Copy**
  - `view()`
  - `copy()`
  - `copy.deepcopy()` (Post-read)
- **Array Splitting**
  - `split()`
  - `hsplit()`
  - `vsplit()`
- **Array Stacking**
  - `hstack()`
  - `vstack()`
  - `concatenate()`
- **Image Manipulation** (Post-lecture content)





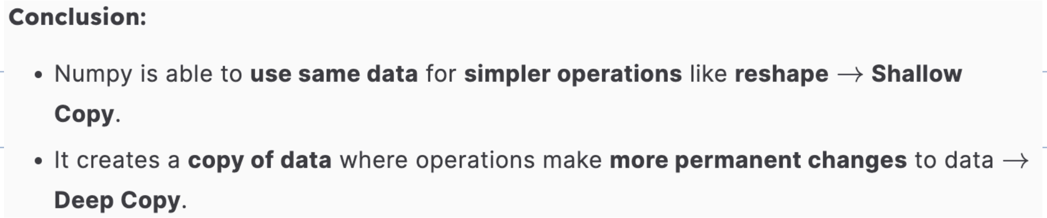
Views vs Copies (Shallow vs Deep Copy)

- Numpy **manages memory very efficiently**,
- which makes it really **useful while dealing with large datasets**.

Shallow Copy → Copy element refers to the same memory as original.

Deep Copy → copy element refers to a new memory.

np.shares_memory(a, b) → checks if a & b share the same memory.

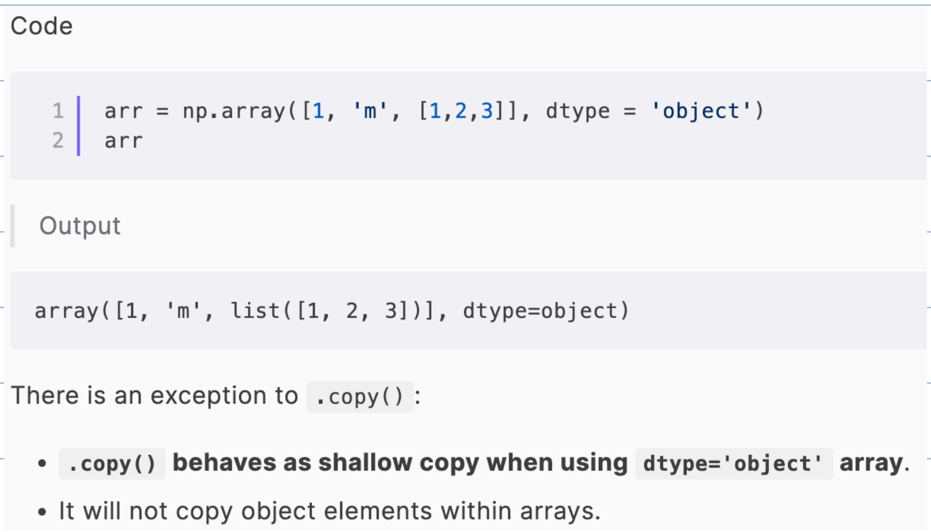


Conclusion:

- Numpy is able to **use same data** for **simpler operations** like **reshape** → **Shallow Copy**.
- It creates a **copy of data** where operations make **more permanent changes** to data → **Deep Copy**.

What are object arrays?

- Object arrays are basically array of any Python datatype.



Code

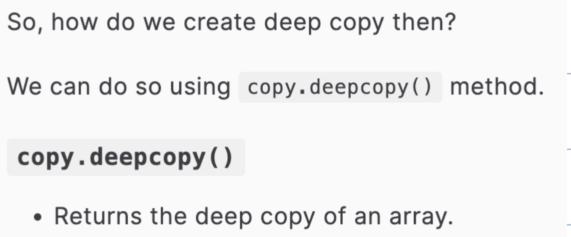
```
arr = np.array([1, 'm', [1,2,3]], dtype = 'object')
arr
```

Output

```
array([1, 'm', list([1, 2, 3])], dtype=object)
```

There is an exception to `.copy()`:

- **`.copy()` behaves as shallow copy when using `dtype='object'` array**.
- It will not copy object elements within arrays.

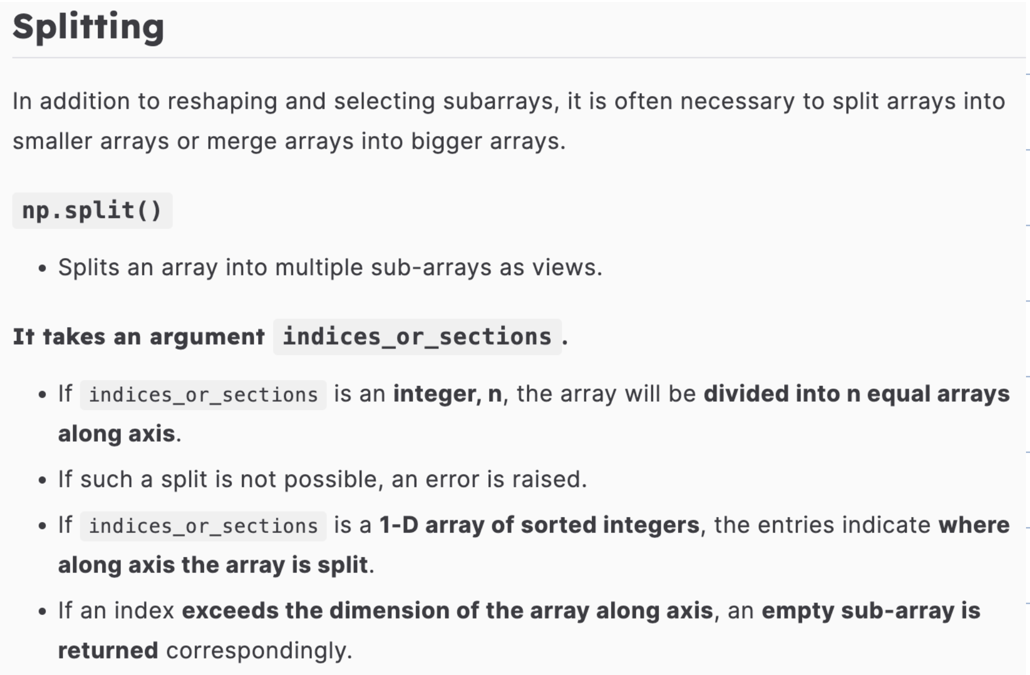


So, how do we create deep copy then?

We can do so using `copy.deepcopy()` method.

`copy.deepcopy()`

- Returns the deep copy of an array.



Splitting

In addition to reshaping and selecting subarrays, it is often necessary to split arrays into smaller arrays or merge arrays into bigger arrays.

`np.split()`

- Splits an array into multiple sub-arrays as views.

It takes an argument `indices_or_sections`.

- If `indices_or_sections` is an **integer, n**, the array will be **divided into n equal arrays along axis**.
- If such a split is not possible, an error is raised.
- If `indices_or_sections` is a **1-D array of sorted integers**, the entries indicate **where along axis the array is split**.
- If an index **exceeds the dimension of the array along axis**, an **empty sub-array is returned** correspondingly.

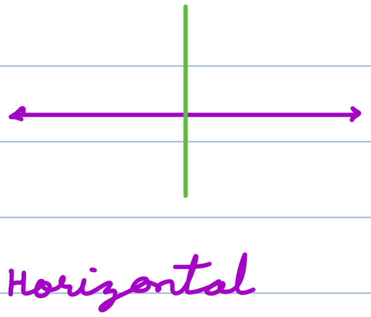
- There are 2 axis to a 2D array
  1. **1st axis - Vertical axis**
  2. **2nd axis - Horizontal axis**

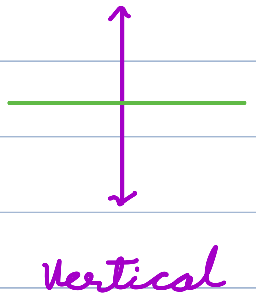
Along which axis are we splitting the array?

- The split we want happens across the **2nd axis (Horizontal axis)**
- That is why we use `hsplit()`

So, try to think in terms of "whether the operation is happening along vertical axis or horizontal axis".

- The split we want happens across the **1st axis (Vertical axis)**
- That is why we use `vsplit()`





Stacking

`np.vstack()`

- Stacks a list of arrays **vertically (along axis 0 or 1st axis)**.
- For **example**, **given a list of row vectors, appends the rows to form a matrix**.

`np.hstack()`

- Stacks a list of arrays **horizontally (along axis 1 or 2nd axis)**.

`np.concatenate()`

- Can perform both `vstack` and `hstack`
- Creates a new array by appending arrays after each other, along a given axis.

Provides similar functionality, but it takes a **keyword argument** `axis` that specifies the **axis along which the arrays are to be concatenated**.

The input array to `concatenate()` needs to be of dimensions atleast equal to the dimensions of output array.