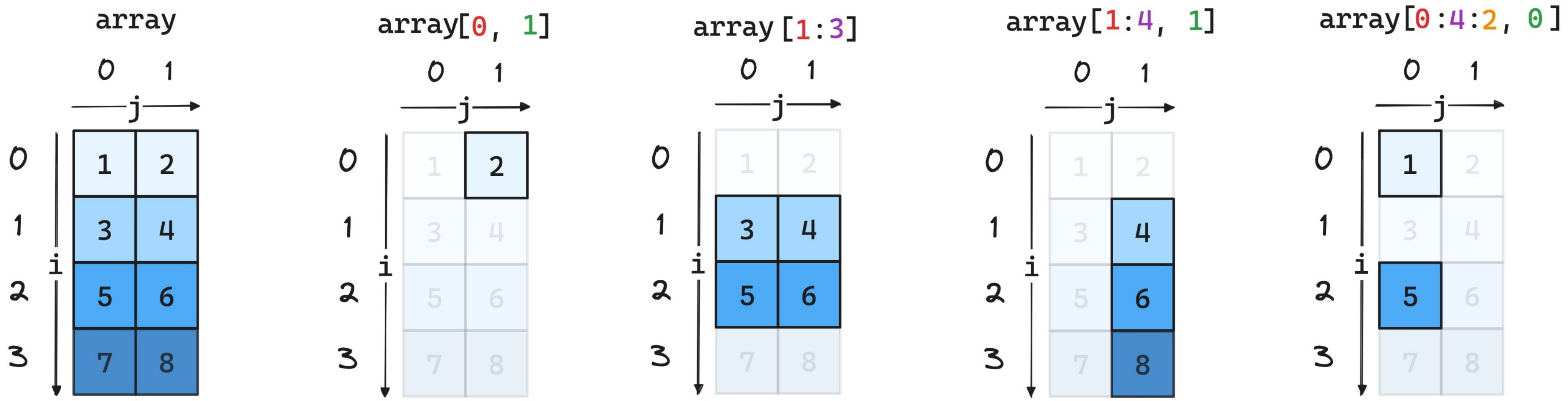
Understanding Indexing, slicing & striding in NumPy!🚀



Step by step explanation with code!

@akshay_pachaar

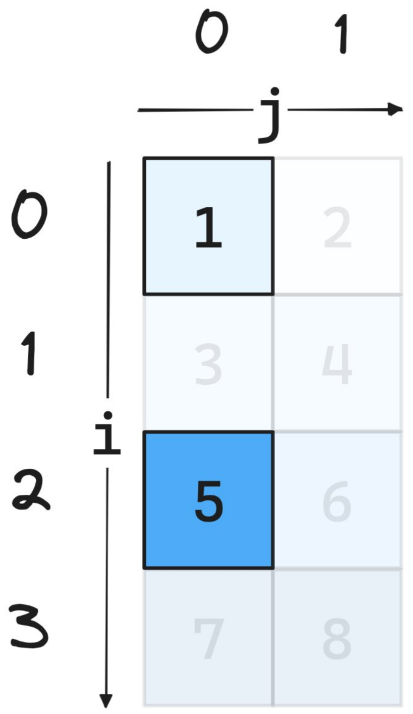
Before we jump into code, let's understand the basic syntax.

Here's an explanation

Swipe 👉

Understanding the Syntax!🚀

```
array[0:4:2, 0]
```



```
array[0:4:2, 0]
```

0: Start index along the first axis (i)

4: Stop index along first axis, The element at this index is not included in the slice

2: This is the step size. It determines the stride between each element selected in the slice

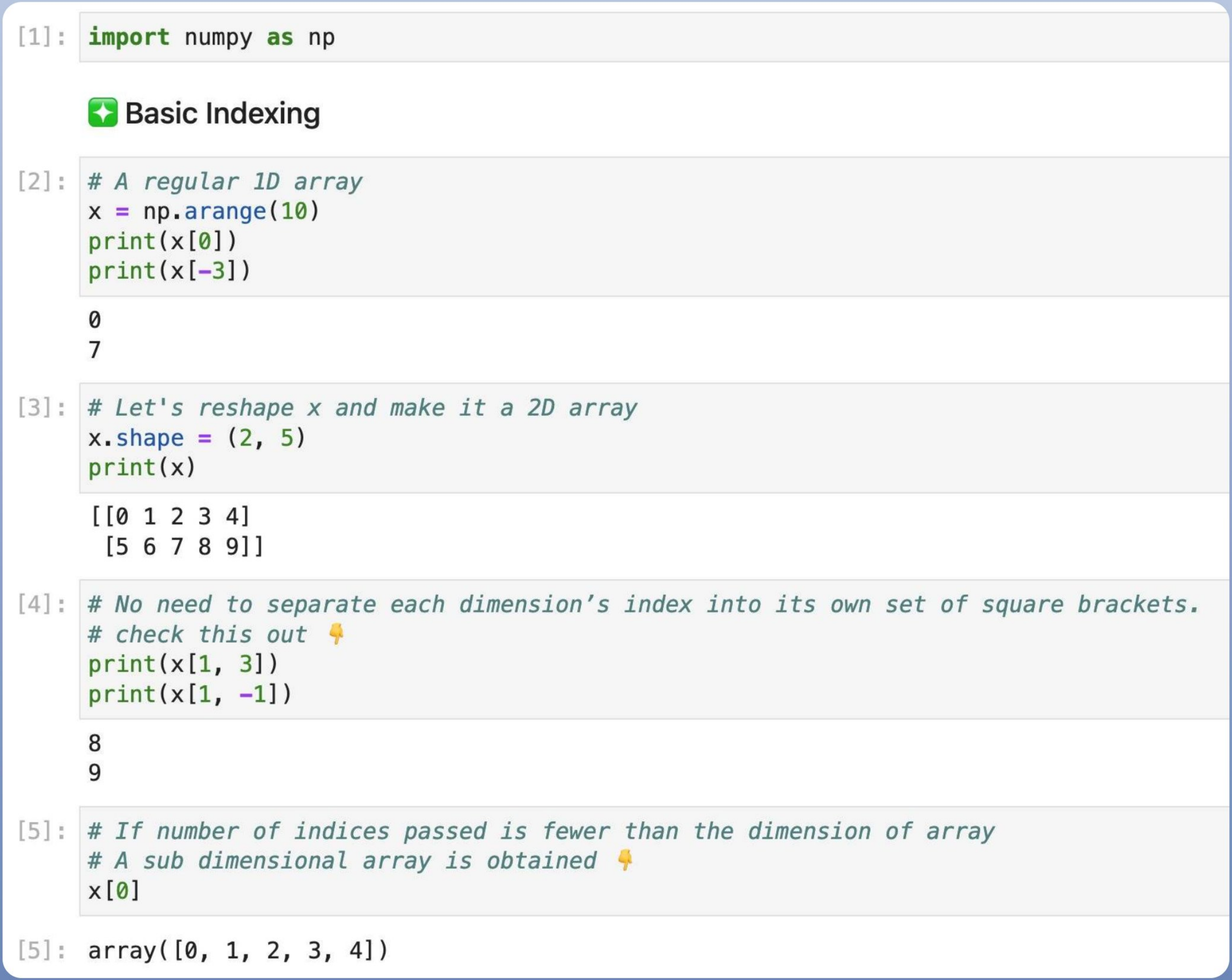
0: This refers to the index along the second axis (j).

@akshay_pachaar

1️⃣ Basic Indexing

Swipe 👉

@akshay_pachaar



```
[1]: import numpy as np
```

❇️ Basic Indexing

```
[2]: # A regular 1D array
     x = np.arange(10)
     print(x[0])
     print(x[-3])
```

```
0
7
```

```
[3]: # Let's reshape x and make it a 2D array
     x.shape = (2, 5)
     print(x)
```

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

```
[4]: # No need to separate each dimension’s index into its own set of square brackets.
     # check this out 👇
     print(x[1, 3])
     print(x[1, -1])
```

```
8
9
```

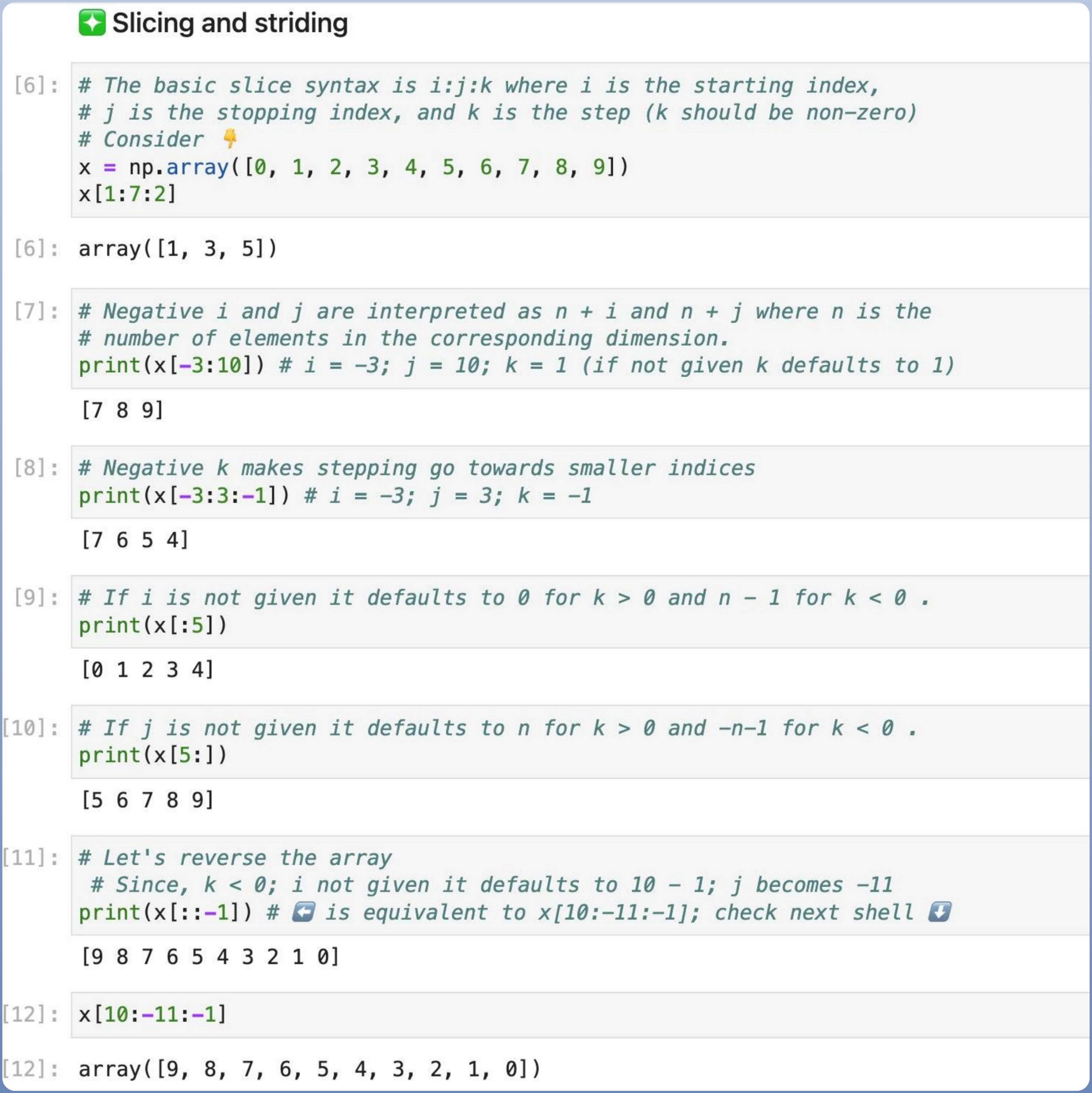
```
[5]: # If number of indices passed is fewer than the dimension of array
     # A sub dimensional array is obtained 👇
     x[0]
```

```
[5]: array([0, 1, 2, 3, 4])
```

2️⃣ Slicing and Striding

Swipe 👉

@akshay_pachaar



❇️ Slicing and striding

```
[6]: # The basic slice syntax is i:j:k where i is the starting index,
     # j is the stopping index, and k is the step (k should be non-zero)
     # Consider 👇
     x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
     x[1:7:2]

[6]: array([1, 3, 5])
```

```
[7]: # Negative i and j are interpreted as n + i and n + j where n is the
     # number of elements in the corresponding dimension.
     print(x[-3:10]) # i = -3; j = 10; k = 1 (if not given k defaults to 1)

     [7 8 9]
```

```
[8]: # Negative k makes stepping go towards smaller indices
     print(x[-3:3:-1]) # i = -3; j = 3; k = -1

     [7 6 5 4]
```

```
[9]: # If i is not given it defaults to 0 for k > 0 and n - 1 for k < 0 .
     print(x[:5])

     [0 1 2 3 4]
```

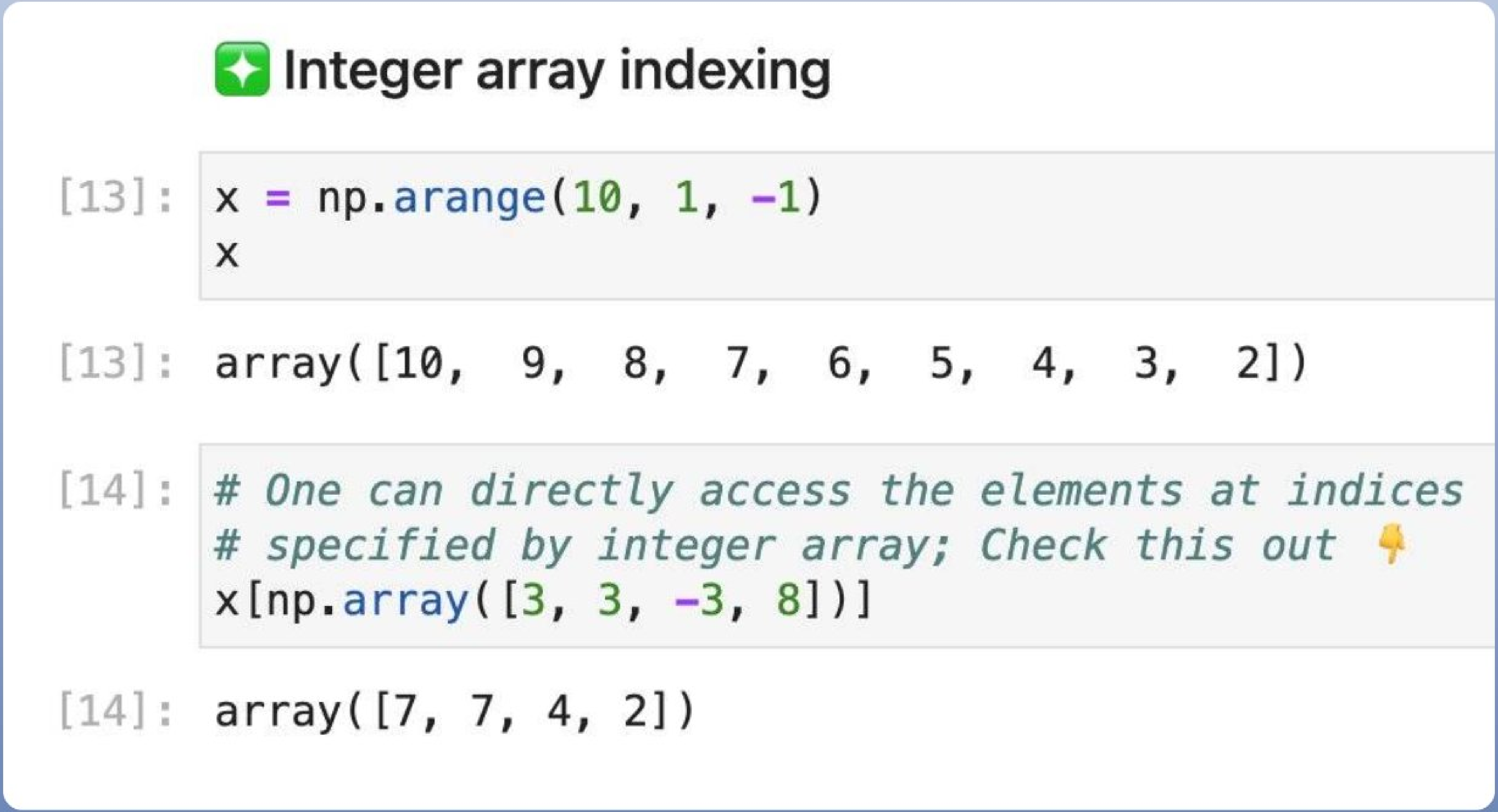
```
[10]: # If j is not given it defaults to n for k > 0 and -n-1 for k < 0 .
      print(x[5:])

      [5 6 7 8 9]
```

```
[11]: # Let's reverse the array
       # Since, k < 0; i not given it defaults to 10 - 1; j becomes -11
      print(x[::-1]) # ⬅️ is equivalent to x[10:-11:-1]; check next shell ⬇️

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

```
[12]: x[10:-11:-1]

[12]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

3️⃣ Integer array Indexing

Swipe 👉



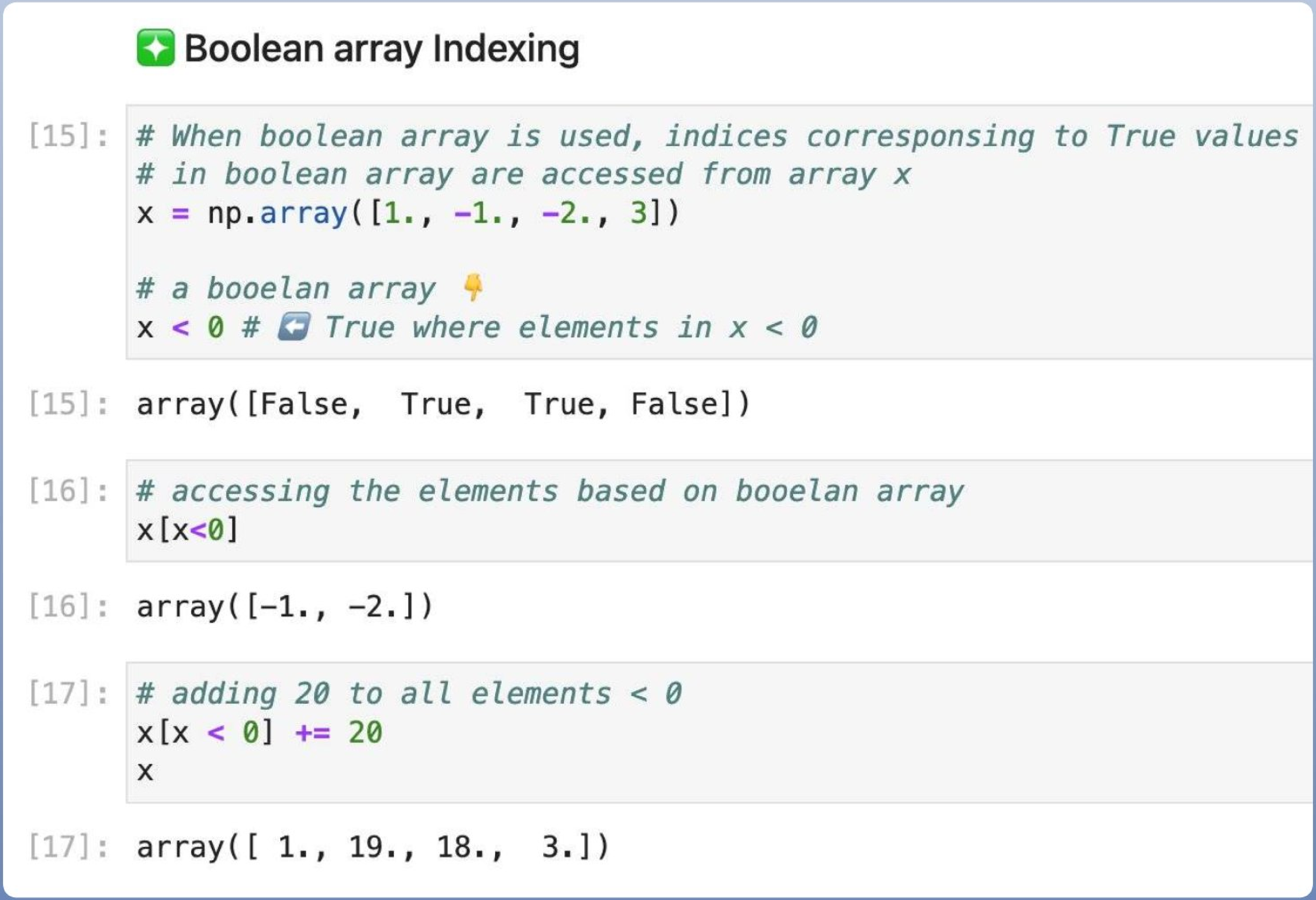
❇️ Integer array indexing

```
[13]: x = np.arange(10, 1, -1)
      x

[13]: array([10,  9,  8,  7,  6,  5,  4,  3,  2])

[14]: # One can directly access the elements at indices
      # specified by integer array; Check this out 👇
      x[np.array([3, 3, -3, 8])]

[14]: array([7, 7, 4, 2])
```

4️⃣ Boolean array indexing

Swipe 👉

@akshay_pachaar



❇️ Boolean array Indexing

```
[15]: # When boolean array is used, indices corresponsing to True values
      # in boolean array are accessed from array x
      x = np.array([1., -1., -2., 3])

      # a booelan array 👇
      x < 0 # ↩️ True where elements in x < 0

[15]: array([False,  True,  True, False])

[16]: # accessing the elements based on booelan array
      x[x<0]

[16]: array([-1., -2.])

[17]: # adding 20 to all elements < 0
      x[x < 0] += 20
      x

[17]: array([ 1., 19., 18.,  3.])
```

That's a wrap!

If you interested in:

- Python 🐍
- Data Science 📈
- Machine Learning 🤖
- MLOps 🛠️
- NLP 🗣️
- Computer Vision 🎥
- LLMs 🧠

Follow me on LinkedIn ✔️
Everyday, I share tutorials on above topics!

Cheers!! 🙂

@akshay_pachaar