DATA SOCIETY:

Data wrangling in Python - Data wrangling with NumPy - 3

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Filter NumPy arrays	
Reshape NumPy arrays	

Recap: arrays

- NumPy is widely used in machine learning and scientific computing due to its basic core data structure: array
- Arrays are useful variables that can hold more than one value at a time
- We can also **filter** arrays to find elements that meet certain criteria
- Before continuing in the code notebook, make sure you have NumPy imported into your environment

```
# Import numpy as 'np' sets 'np' as the shortcut/alias.
import numpy as np
```

Accessing array values

- Just like with lists, we can grab individual elements or a range of elements from an array using square bracket notation

```
nums = np.arange(20, 30, .5)

print(nums[1])  #<- get the second element
```

```
20.5
```

```
print(nums[0:3]) #<- get the first three elements
```

```
[20.  20.5 21. ]
```

Logical filtering

- You can't filter lists by a logical condition, but you can filter arrays by a **logical condition**
- If the corresponding condition is met, then it retains the value from the array; otherwise it excludes it

```
print(nums)
```

```
[20.  20.5 21.  21.5 22.  22.5 23.  23.5 24.  24.5 25.  25.5 26.  26.5
 27.  27.5 28.  28.5 29.  29.5]
```

```
large_nums = nums[nums > 26]
print(large_nums)
```

```
[26.5 27.  27.5 28.  28.5 29.  29.5]
```

Logical filtering (cont'd)

```
print(nums)
```

```
[20.   20.5 21.   21.5 22.   22.5 23.   23.5 24.   24.5 25.   25.5 26.   26.5
 27.   27.5 28.   28.5 29.   29.5]
```

```
large_nums = nums[nums > 26]
print(large_nums)
```

```
[26.5 27.  27.5 28.  28.5 29.  29.5]
```

- It is important to remember that there are a few steps happening here:
  - The expression within the brackets produces a so-called **Boolean mask**: an array of `True/False` values
  - The logical statement `> 26` is applied to each value of `nums`, so the result is an array of `True`/`False` values
  - Our `nums` array and the mask array are then lined up, and the values out of `nums` are filtered based on the corresponding mask value

Two-dimensional arrays

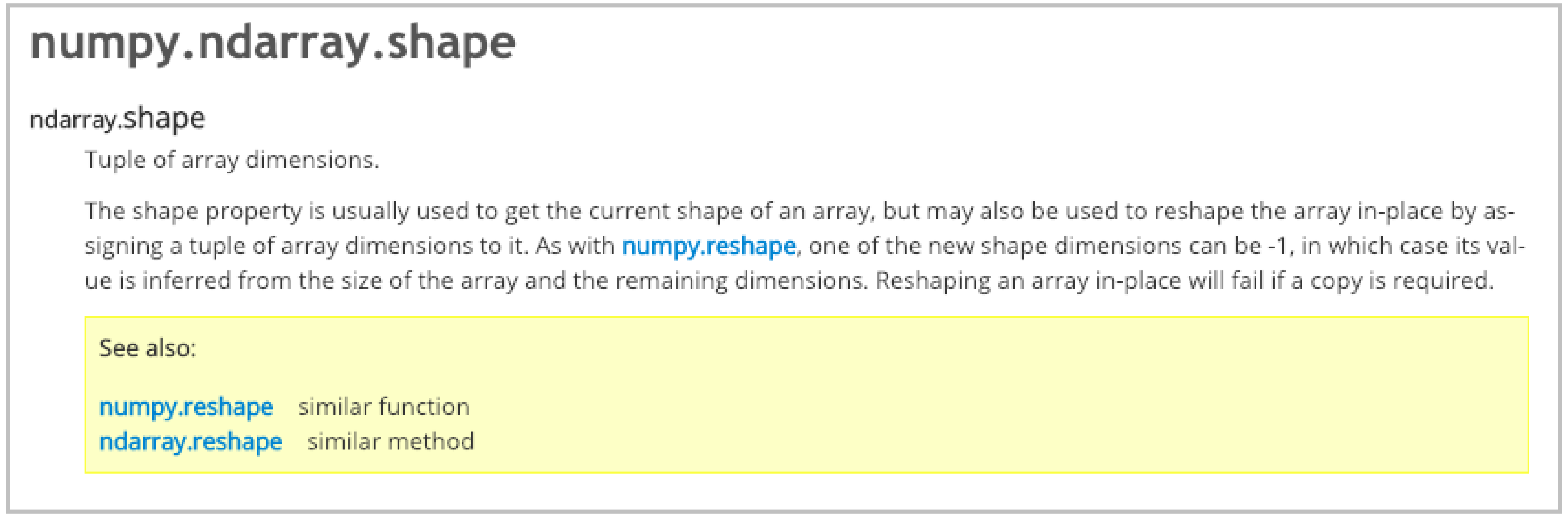
- As the name suggests, `ndarray` (i.e. n-dimensional array) can have more than one dimension
- Multiple dimensions are created by nesting lists within each other
- To create a 2D array (a **matrix**), we can write the following:

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

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

Two-dimensional arrays - shape

- The `shape` property of an array tells us the size of each of its dimensions



numpy.ndarray.shape

ndarray.shape

Tuple of array dimensions.

The shape property is usually used to get the current shape of an array, but may also be used to reshape the array in-place by assigning a tuple of array dimensions to it. As with numpy.reshape, one of the new shape dimensions can be -1, in which case its value is inferred from the size of the array and the remaining dimensions. Reshaping an array in-place will fail if a copy is required.

See also:

numpy.reshape	similar function
ndarray.reshape	similar method

Two-dimensional arrays - shape (cont'd)

```
print(mat.shape) #<- 3 rows and 4 columns -- returned as a tuple
```

```
(3, 4)
```

```
nrows, ncols = mat.shape
print(nrows)
```

```
3
```

Two-dimensional arrays - extracting elements

- To extract a value from the matrix, we use two-dimensional bracket notation:
  - The **1st** number is the **row** position
  - The **2nd** number is the **column** position

```
print(mat[1, 3]) #<- 2nd row 4th column - remember that indexing starts at 0!
```

```
2
```

Two-dimensional arrays - rows

- To extract an entire row of a matrix, replace the column ID with a colon
- The colon indicates that you would like to include all of the columns
- Alternatively, you can specify a range of column positions, which uses normal Python list slicing notation

```
print(mat[0, :]) #<- first row
```

```
[8 2 6 8]
```

```
print(mat[0, 0:2]) #<- first row and just first 2 columns
```

```
[8 2]
```

Two-dimensional arrays - columns

- Similarly, to extract a single column, replace the row argument with a colon or leave it blank

```
print(mat[:, 2]) #<- 3rd column
```

```
[6 7 7]
```

```
print(mat[1:3, 2]) #<- 3rd column but skipping over the first row
```

```
[7 7]
```

```
print(mat[1:3, 2:3]) #<- same as previous, but maintains the vertical structure of the column
```

```
[[7]
 [7]]
```

Module completion checklist

Objective	Complete
Filter NumPy arrays	✔
Reshape NumPy arrays	

Reshaping arrays

- Sometimes we may need to reshape an array according to our needs
- We can do so by calling `.reshape()` function on an array and passing the **new shape** as an argument

```
arr = np.arange(1,13)
print(arr)
```

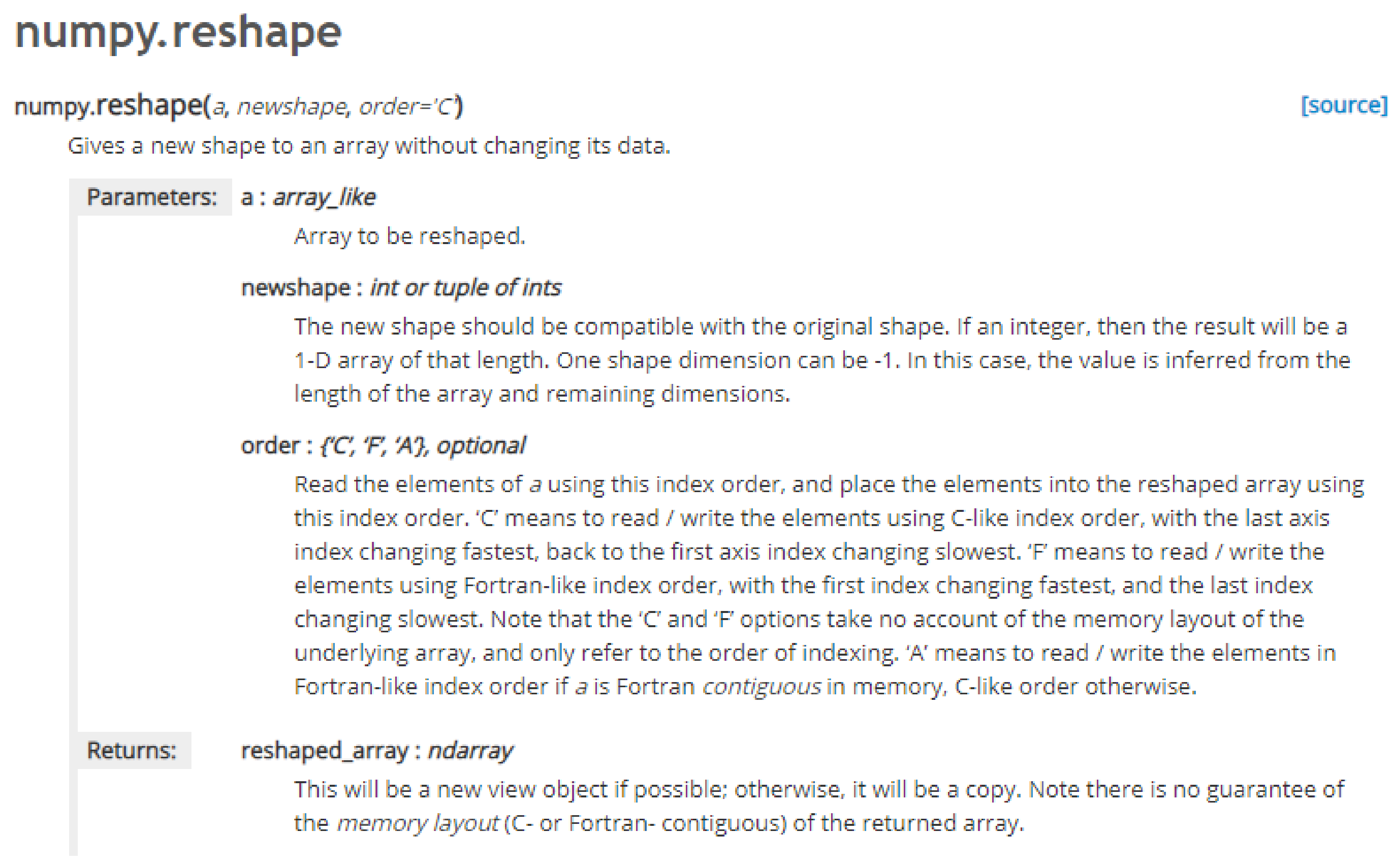
```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
print(arr.reshape(3, 4))
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Reshaping arrays (cont'd)

- Here is some more information about reshaping



numpy.reshape

numpy.reshape(*a, newshape, order='C'*) [source]

Gives a new shape to an array without changing its data.

Parameters: **a** : *array_like*

Array to be reshaped.

newshape : *int or tuple of ints*

The new shape should be compatible with the original shape. If an integer, then the result will be a 1-D array of that length. One shape dimension can be -1. In this case, the value is inferred from the length of the array and remaining dimensions.

order : *{'C', 'F', 'A'}, optional*

Read the elements of *a* using this index order, and place the elements into the reshaped array using this index order. 'C' means to read / write the elements using C-like index order, with the last axis index changing fastest, back to the first axis index changing slowest. 'F' means to read / write the elements using Fortran-like index order, with the first index changing fastest, and the last index changing slowest. Note that the 'C' and 'F' options take no account of the memory layout of the underlying array, and only refer to the order of indexing. 'A' means to read / write the elements in Fortran-like index order if *a* is Fortran *contiguous* in memory, C-like order otherwise.

Returns: **reshaped_array** : *ndarray*

This will be a new view object if possible; otherwise, it will be a copy. Note there is no guarantee of the *memory layout* (C- or Fortran- contiguous) of the returned array.

Reshaping arrays (cont'd)

- We can also specify one of the new dimensions and let Python infer the other dimension, given the number of elements in the array if possible

```
print(arr.reshape(2,        #<- specify number of rows=2
                 -1))       #<- number of columns=-1 lets Python infer it
```

- If the array cannot be reshaped to the given dimensions, Python throws an error

```
print(arr.reshape(2,        #<- specify number of rows=2
                  2))       #<- number of columns=2 throws error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-c2ef230382ec> in <module>
----> 1 arr.reshape(5,-1)

ValueError: cannot reshape array of size 12 into shape (5,newaxis)
```

Knowledge check

Link: ***https://forms.gle/n4hs4tacPNYTuWAN8***

Exercise

You are now ready to try Tasks 7-8 in the Exercise for this topic

Module completion checklist

Objective	Complete
Filter NumPy arrays	✔
Reshape NumPy arrays	✔

Data Wrangling with NumPy: Topic summary

In this part of the course, we have covered:

- NumPy use cases and object types
- NumPy array manipulation

Congratulations on completing this module!