

Introducing arrays

INTRODUCTION TO NUMPY



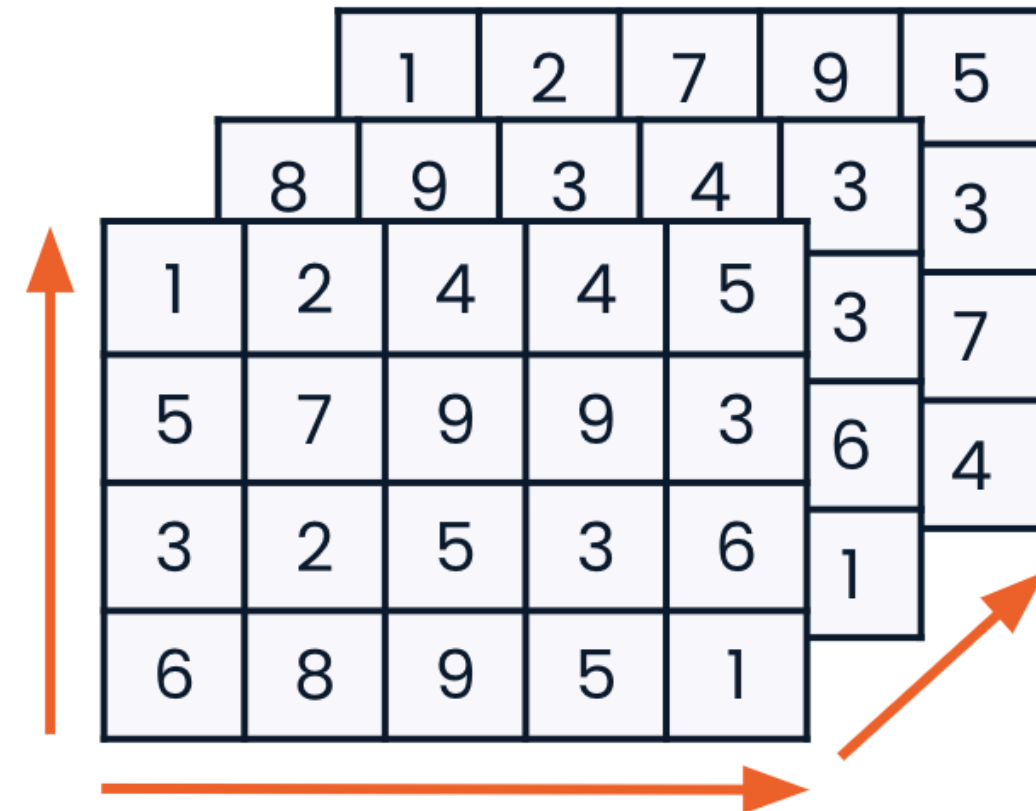
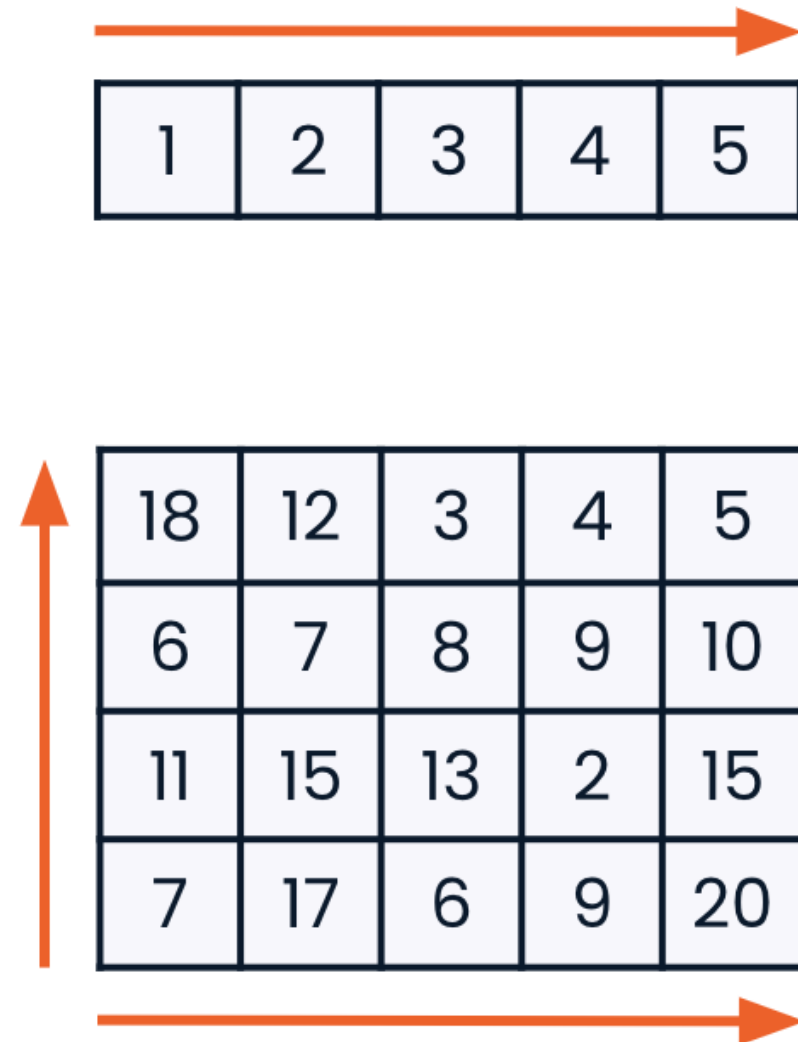
Izzy Weber

Core Curriculum Manager, DataCamp

NumPy and the Python ecosystem



NumPy arrays



Importing NumPy

```
import numpy as np
```

Creating 1D arrays from lists

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

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

```
type(array)
```

```
numpy.ndarray
```

Creating 2D arrays from lists

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

```
array([[3, 2, 5],  
       [9, 7, 1],  
       [4, 3, 6]])
```

Python lists

- Can contain many different data types

```
python_list = ["beep", False, 56, .945, [3, 2, 5]]
```

NumPy arrays

- Can contain only a single data type
- Use less space in memory

```
numpy_boolean_array = [[True, False], [True, True], [False, True]]
```

```
numpy_float_array = [1.9, 5.4, 8.8, 3.6, 3.2]
```

Creating arrays from scratch

There are many NumPy functions used to create arrays from scratch, including:

- `np.zeros()`
- `np.random.random()`
- `np.arange()`

Creating arrays: np.zeros()

```
np.zeros((5, 3))
```

```
array([[0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.]])
```

Creating arrays: `np.random.random()`

```
np.random.random((2, 4))
```

```
array([[0.88524516, 0.85641352, 0.33463107, 0.53337117],  
       [0.69933362, 0.09295327, 0.93616428, 0.03601592]])
```

`np.random.random()`

NumPy module

Function name

Creating arrays with np.arange()

```
np.arange(-3, 4)
```

```
array([-3, -2, -1,  0,  1,  2,  3])
```

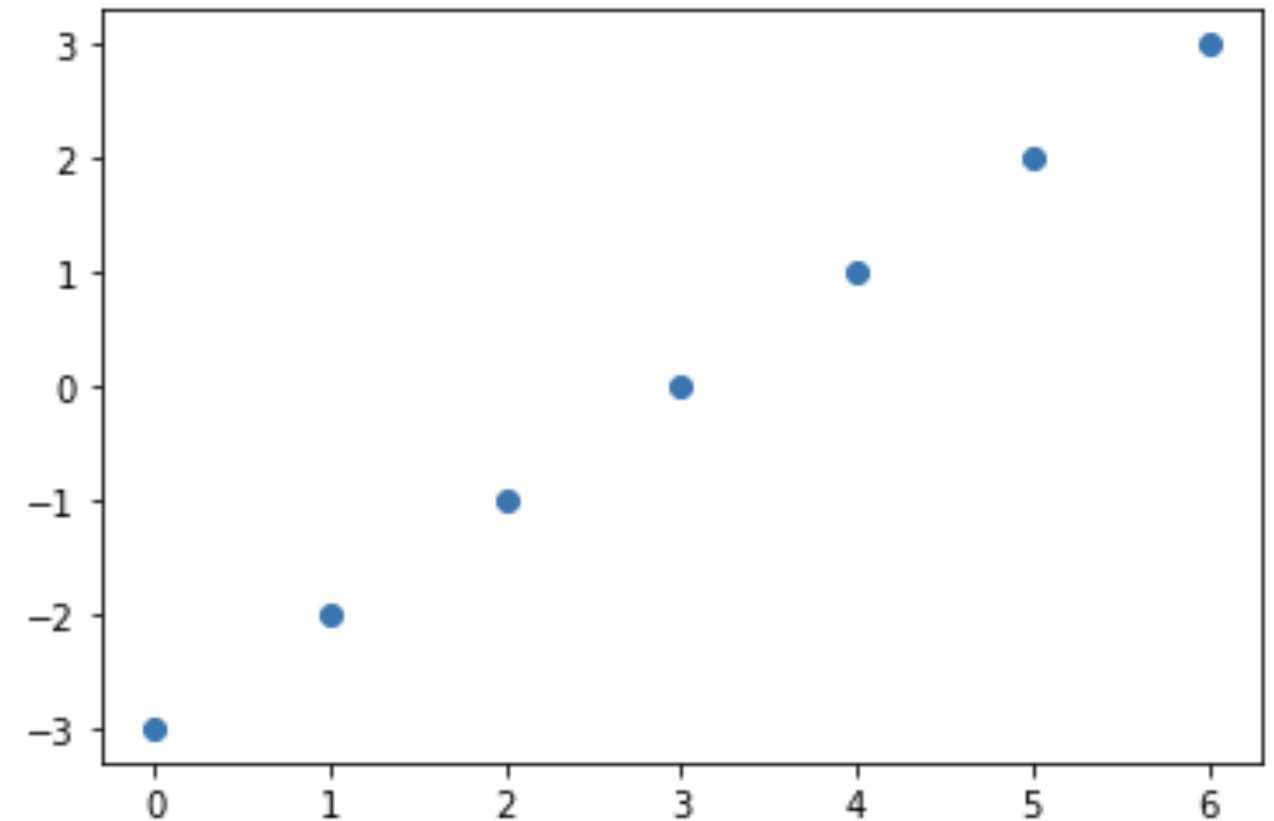
```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

```
np.arange(-3, 4, 3)
```

```
array([-3,  0,  3])
```

```
from matplotlib import pyplot as plt  
plt.scatter(np.arange(0, 7),  
            np.arange(-3, 4))  
plt.show()
```



Let's practice!

INTRODUCTION TO NUMPY

Array dimensionality

INTRODUCTION TO NUMPY

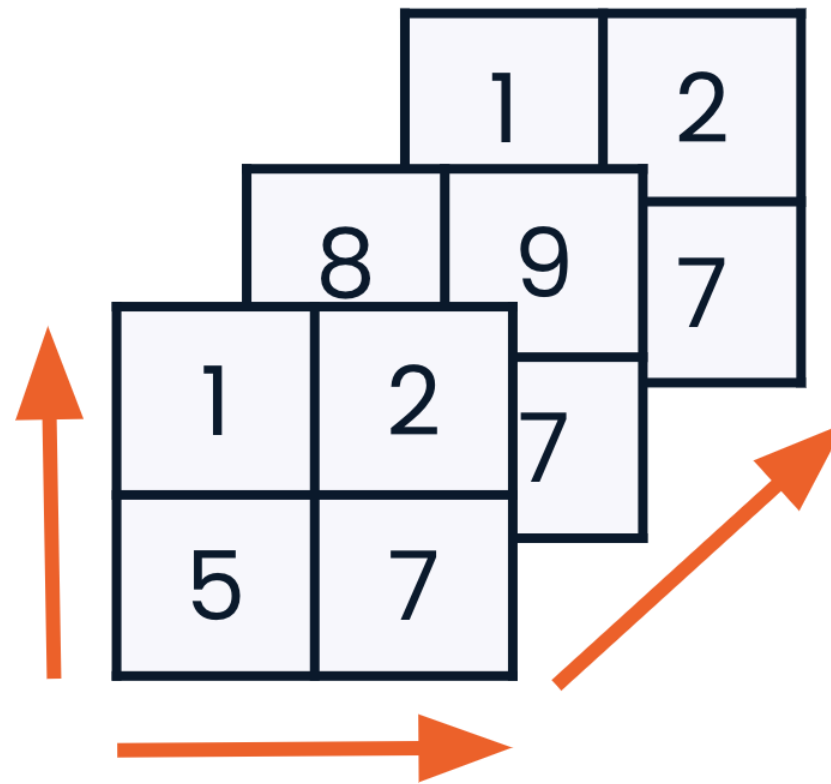


Izzy Weber

Core Curriculum Manager, DataCamp

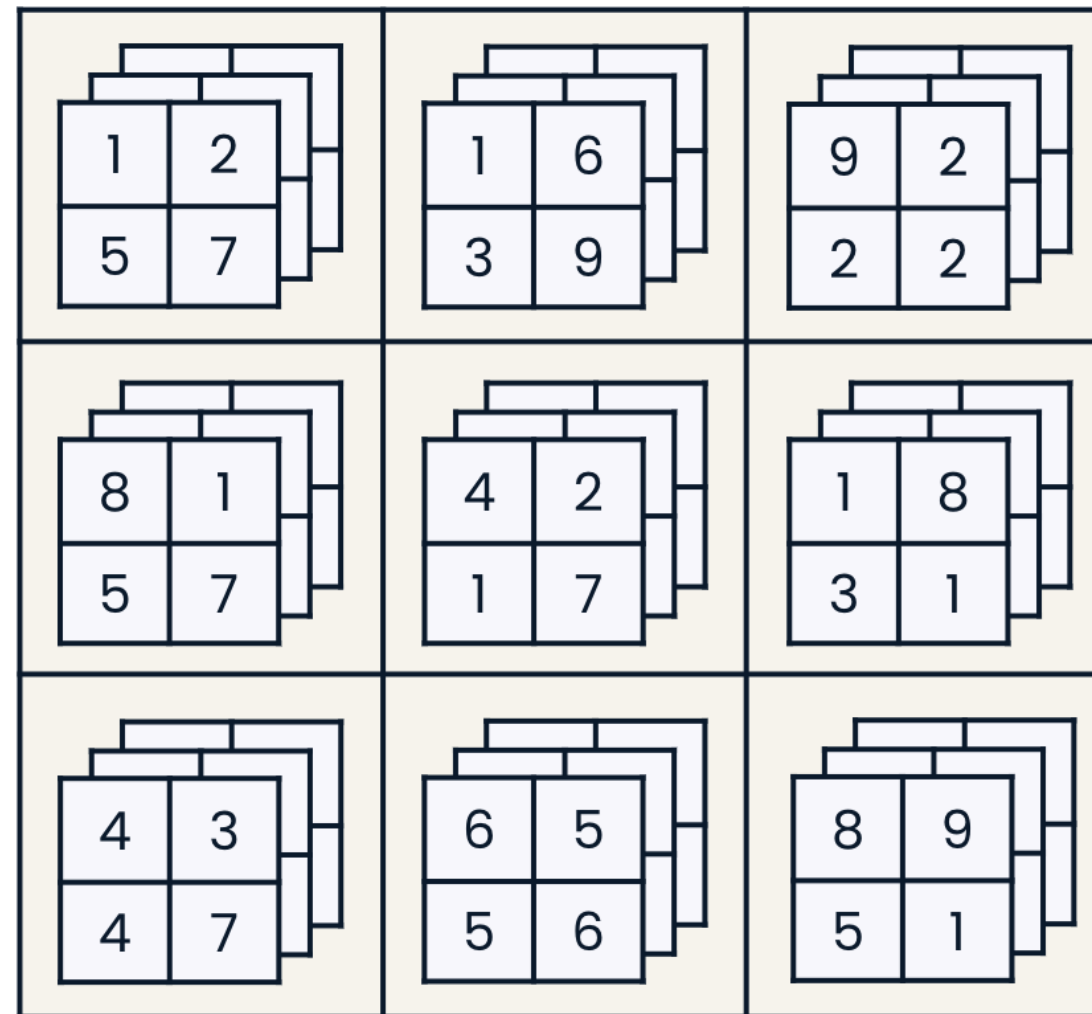
3D arrays

```
array_1_2D = np.array([[1, 2], [5, 7]])  
array_2_2D = np.array([[8, 9], [5, 7]])  
array_3_2D = np.array([[1, 2], [5, 7]])  
array_3D = np.array([array_1_2D, array_2_2D, array_3_2D])
```

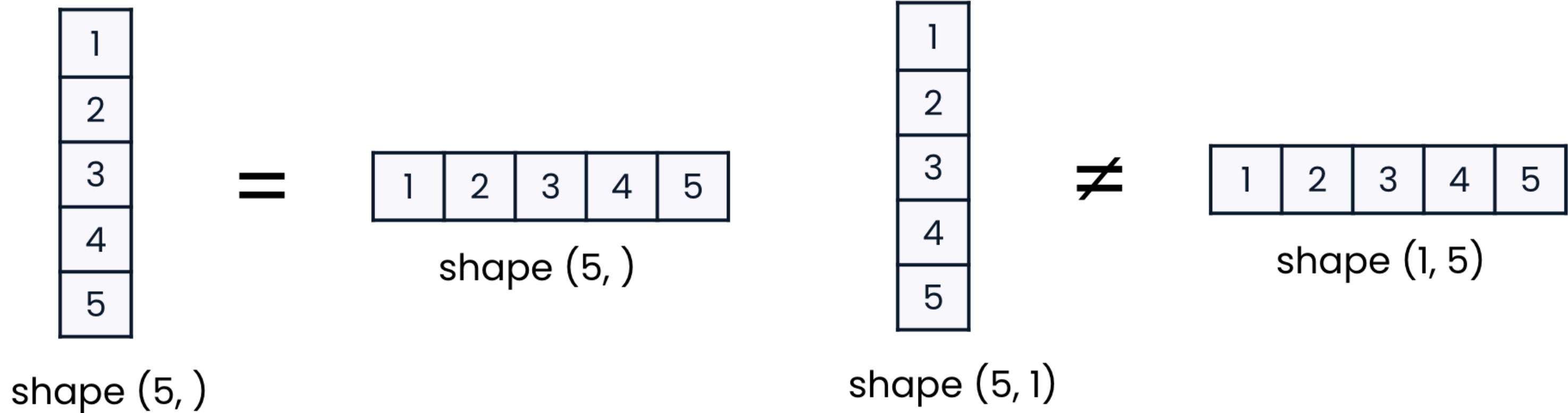


4D arrays

```
array_4D = [array_A_3D, array_B_3D, array_C_3D, array_D_3D, array_E_3D,  
            array_F_3D, array_G_3D, array_H_3D, array_I_3D]
```



Vector arrays



Matrix and tensor arrays

- A matrix has two dimensions

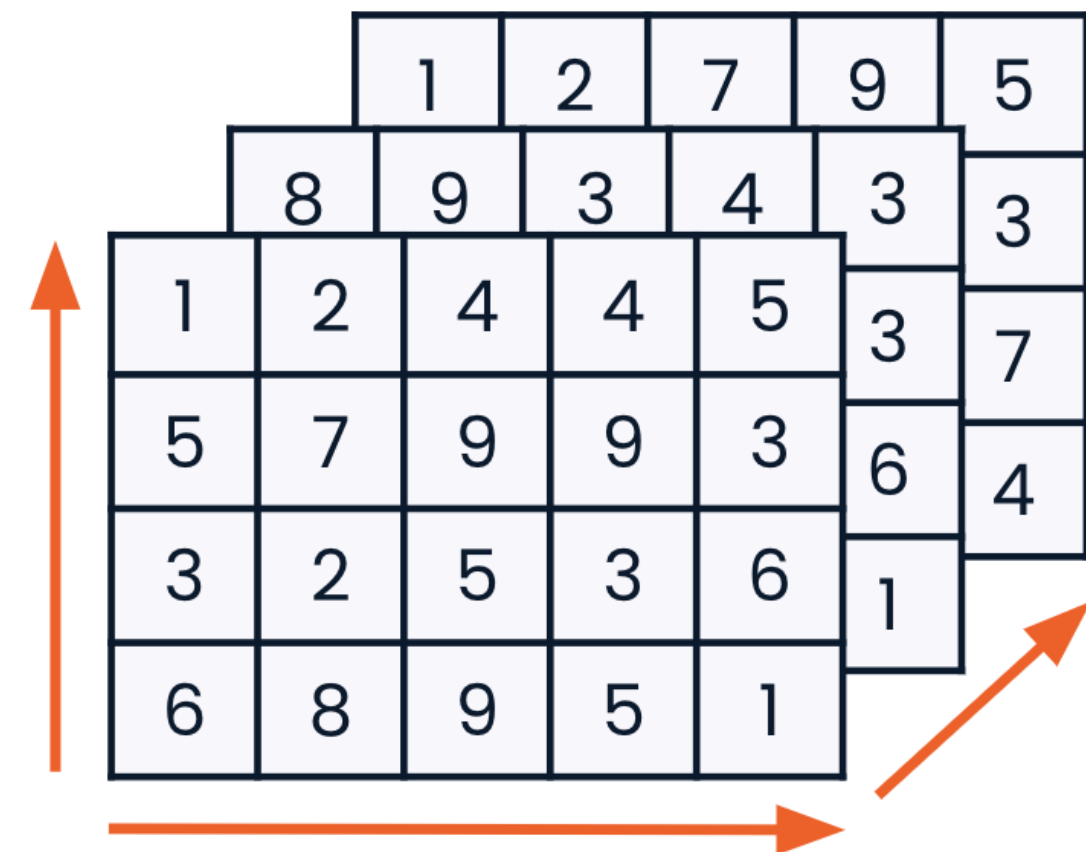
matrix



18	12	3	4	5
6	7	8	9	10
11	15	13	2	15
7	17	6	9	20

- A tensor has three or more dimensions

tensor



		1	2	7	9	5
	8	9	3	4	3	3
1	2	4	4	5	3	7
5	7	9	9	3	6	4
3	2	5	3	6	1	
6	8	9	5	1		

Shapeshifting

Array attribute:

- `.shape`

Array methods:

- `.flatten()`
- `.reshape()`

Finding an array's shape

```
array = np.zeros((3, 5))  
print(array)
```

```
array([[0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0.]])
```

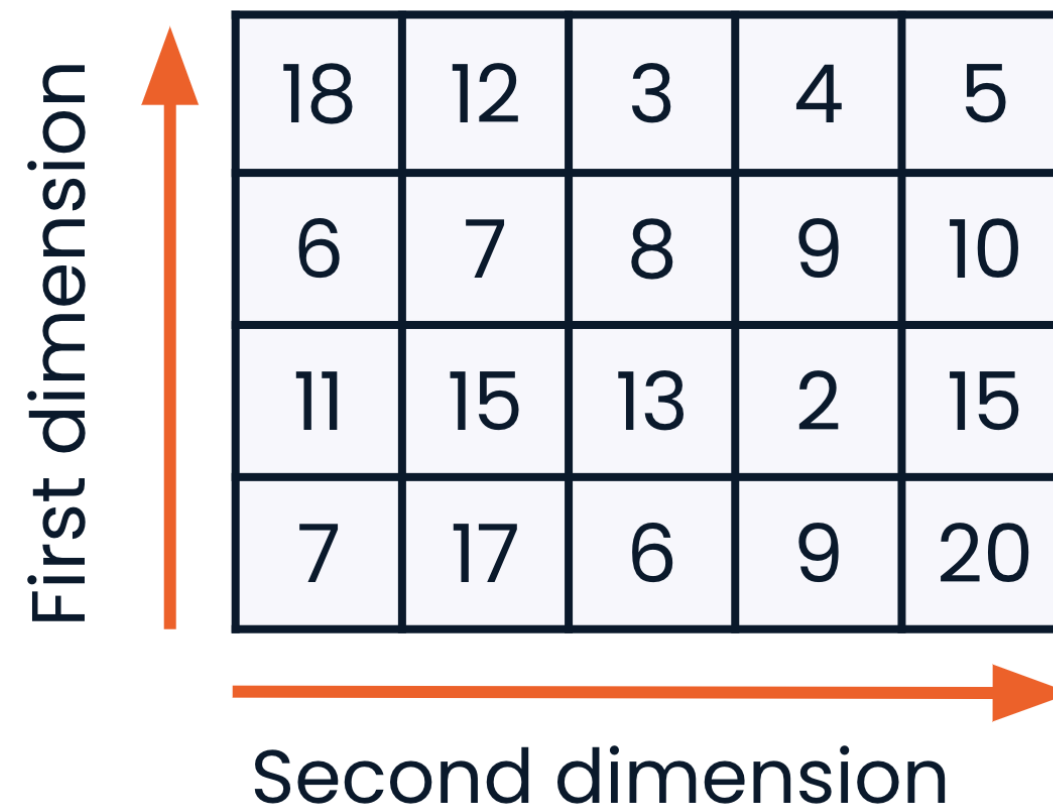
```
array.shape
```

```
(3, 5)
```

Rows and columns

In 2D arrays...

- Rows are the first dimension
- Columns are the second dimension



A 4x5 grid of numbers representing a 2D array. To the left of the grid is a vertical orange arrow pointing upwards, labeled 'First dimension'. Below the grid is a horizontal orange arrow pointing to the right, labeled 'Second dimension'.

18	12	3	4	5
6	7	8	9	10
11	15	13	2	15
7	17	6	9	20

Flattening an array

```
array = np.array([[1, 2], [5, 7], [6, 6]])  
array.flatten()
```

```
array([1, 2, 5, 7, 6, 6])
```

Reshaping an array

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

```
array([[1, 2, 5],  
       [7, 6, 6]])
```

```
array.reshape((3, 3))
```

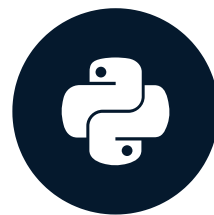
```
ValueError: cannot reshape array of size 6 into shape (3,3)
```

Let's practice!

INTRODUCTION TO NUMPY

NumPy data types

INTRODUCTION TO NUMPY



Izzy Weber

Core Curriculum Manager, DataCamp

NumPy vs. Python data types

Sample Python data types:

- `int`
- `float`

Sample NumPy data types:

- `np.int64`
- `np.int32`
- `np.float64`
- `np.float32`

Bits and bytes

The number 10436 represented in binary is:

0010100011000100



The binary string 0010100011000100 is shown with two brackets underneath. The first bracket covers the first 8 bits (00101000) and is labeled '8 bits = 1 byte'. The second bracket covers the next 8 bits (11000100) and is also labeled '8 bits = 1 byte'.

`np.int32` can store 4,294,967,296 integers:



A horizontal double-headed arrow spans the width of the equation below it, indicating a range or span.

$$2^{32} = 4,294,967,296$$

Bits and bytes

The number 10436 represented in binary is:

0010100011000100

8 bits = 1 byte

8 bits = 1 byte

`np.int32` can store 4,294,967,296 integers:

-2,147,483,648

2,147,483,647

$$2^{32} = 4,294,967,296$$

The .dtype attribute

```
np.array([1.32, 5.78, 175.55]).dtype
```

```
dtype('float64')
```

Default data types

```
int_array = np.array([[1, 2, 3], [4, 5, 6]])  
int_array.dtype
```

```
dtype('int64')
```

String data

```
np.array(["Introduction", "to", "NumPy"]).dtype
```

```
dtype('<U12')
```

dtype as an argument

```
float32_array = np.array([1.32, 5.78, 175.55], dtype=np.float32)  
float32_array.dtype
```

```
dtype('float32')
```

Type conversion

```
boolean_array = np.array([[True, False], [False, False]], dtype=np.bool_)  
boolean_array.astype(np.int32)
```

```
array([[1, 0],  
       [0, 0]], dtype=int32)
```


Type coercion

```
np.array([True, "Boop", 42, 42.42])
```

```
array(['True', 'Boop', '42', '42.42'], dtype='<U5')
```

Type coercion hierarchy

Adding a float to an array of integers will change all integers into floats:

```
np.array([0, 42, 42.42]).dtype
```

```
dtype('float64')
```

Adding an integer to an array of booleans will change all booleans in to integers:

```
np.array([True, False, 42]).dtype
```

```
dtype('int64')
```

Let's practice!

INTRODUCTION TO NUMPY