Fachhochschule Kiel
University of Applied Sciences

Tools and Programming Languages for Data Science
NumPy

Study Program Data Science

Prof. Dr. Tillmann Schwörer

Python foundations

Data types

Operators

Functions

Control flow and iterators

Programming concepts & paradigms

See also Precourse Programming

Tooling

Installation

Visual Studio Code

Jupyter Notebooks

Packages

Virtual Environments

Git and Github

Python

Data Science Workflow

NumPy pandas matplotlib

Our course agenda

- **Introduction and overview**
- **NumPy**: Basic data handling with Numpy arrays
- **Pandas**
  - Exploratory data analysis
  - Data consolidation
  - Data cleaning
- **Data visualization using Matplotlib and Seaborn**
- **Interacting with APIs**
- **Interacting with SQL databases**
- **Version Control with Git and GitHub**
- **Advanced Python**

Numpy: Numeric Python

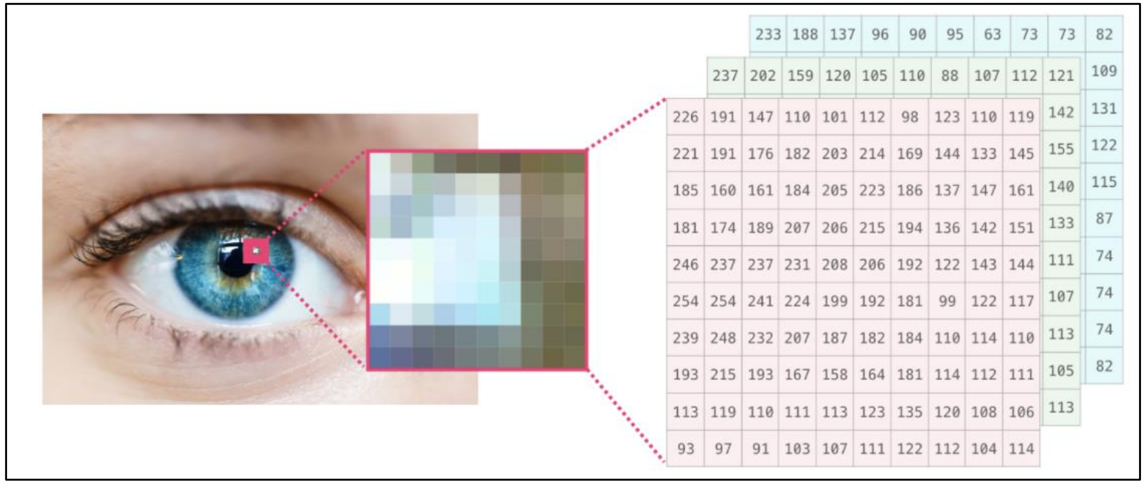
Data science is all about numbers

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

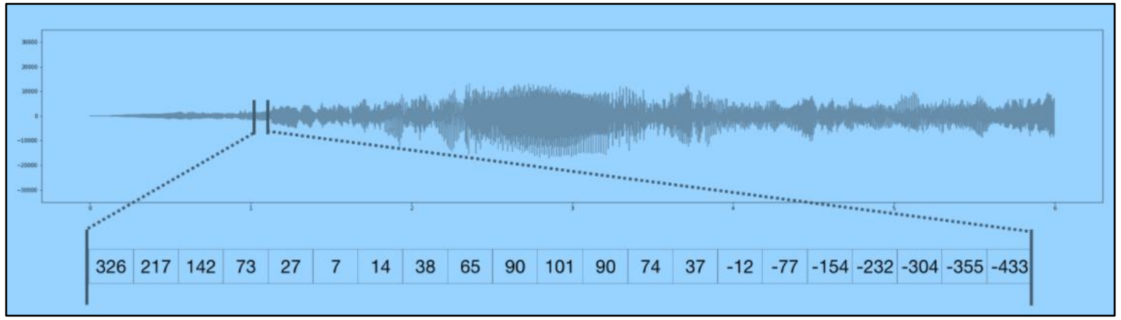
One-Hot Encoded Data

Team_A	Team_B	Team_C	Points
1	0	0	25
1	0	0	12
0	1	0	15
0	1	0	14
0	1	0	19
0	1	0	23
0	0	1	25
0	0	1	29



	text
0	The sky is blue and beautiful
1	Love this blue and beautifl sky
2	The quick brown fox jumps over the lazy dog.
3	A King's breakfast has sausages, ham, bacon, e...
4	I love green eggs, ham, sausages and bacon!
5	The brown fox is quick and the blue dog is lazy!
6	The sky is very blue and the sky is very beaut...
7	The dog is lazy but the brown fox is quick

	bacon	beans	beautifl	beautiful	blue	breakfast
0	0	0	0	1	1	0
1	0	0	1	0	1	0
2	0	0	0	0	0	0
3	1	1	0	0	0	1
4	1	0	0	0	0	0
5	0	0	0	0	1	0
6	0	0	0	1	1	0
7	0	0	0	0	0	0



http://jalammar.github.io/visual-numpy/

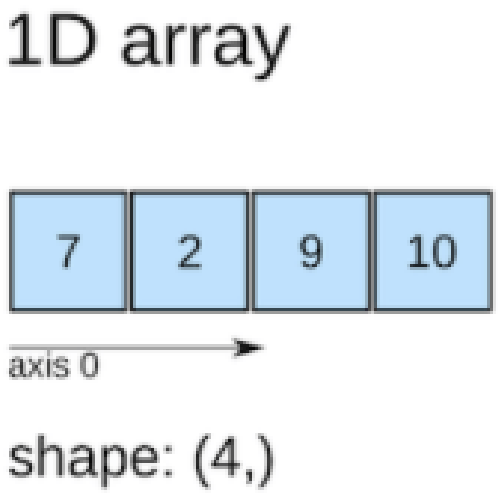
Characteristics of NumPy arrays

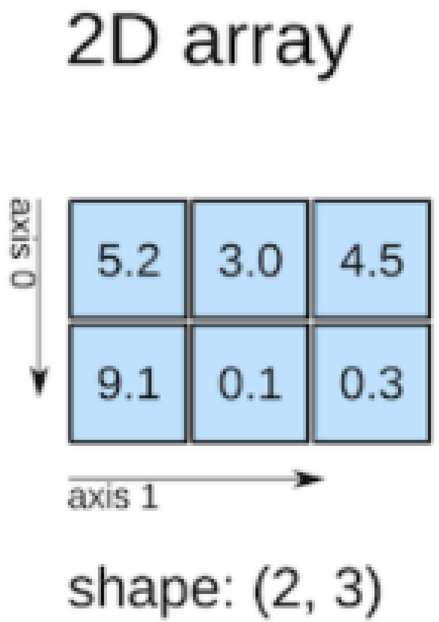
- **Multidimensional:**
  - Vectors (1 dimension), matrices (2), tensors (3 or more)
  - We say „dimensions“ or „axes“
- **Homogeneous data type**
  - All elements must be of the same data type: e.g. only integers
  - Enables efficient storage and fast computations because there is no need for type checking
- **Array attributes**
  - <u>ndim</u>: number of dimensions (e.g. 1, 2, 3, …)
  - <u>shape</u>: length along each dimension (e.g. (50, 5) for data with 50 rows and 5 cols)
  - <u>size</u>: total number of elements (e.g. 250 for data with 50 rows and 5 cols)
  - <u>dtype:</u> data type (e.g. int32, float64, …)

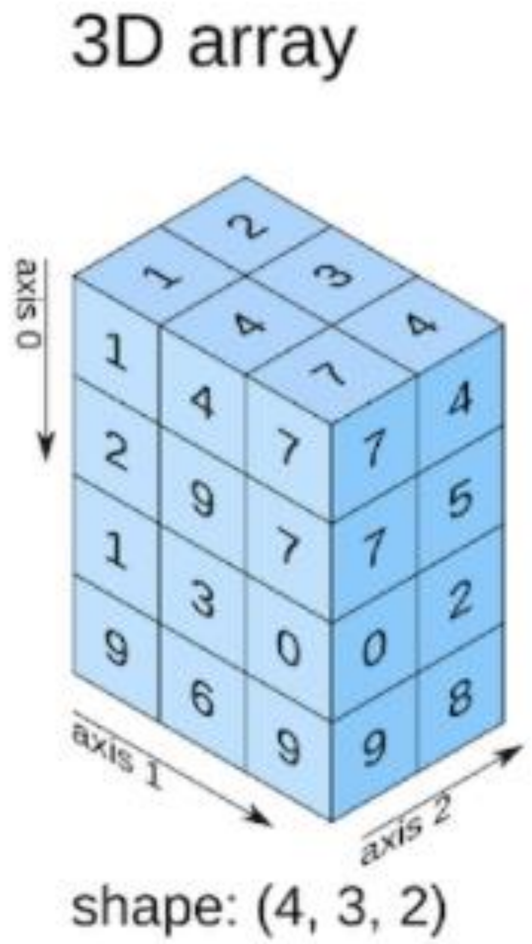
NumPy Key Features

- **Multidimensional arrays**
  - 1, 2, or higher-dimensional data
- **Efficient indexing, slicing and manipulation of array elements**
- **Mathematical/statistical functions** operating on arrays:
  - mean
  - max
  - log
  - …
- **Linear Algebra**
  - dot product
  - solving system of equations
  - …

N-dimensional array








```
np.array([7, 2, 9, 10])
```

```
np.array([(5.2, 3.0, 4.5),
          (9.1, 0.1, 0.3)])
```

Advantages of NumPy

- **Highly performant**
- **Memory efficient**
- Broad range of **operations relevant for data analysis and manipulation**
- **Linked to or interoperable with many other Python packages**
  - Pandas Series and DataFrames are built on numpy
  - Machine learning with scikit-learn expects numpy arrays as input,
  - Arrays can be visualized with matplotlib
  - Pytorch tensors are interoperable with numpy arrays
  - …

Characteristics of NumPy arrays

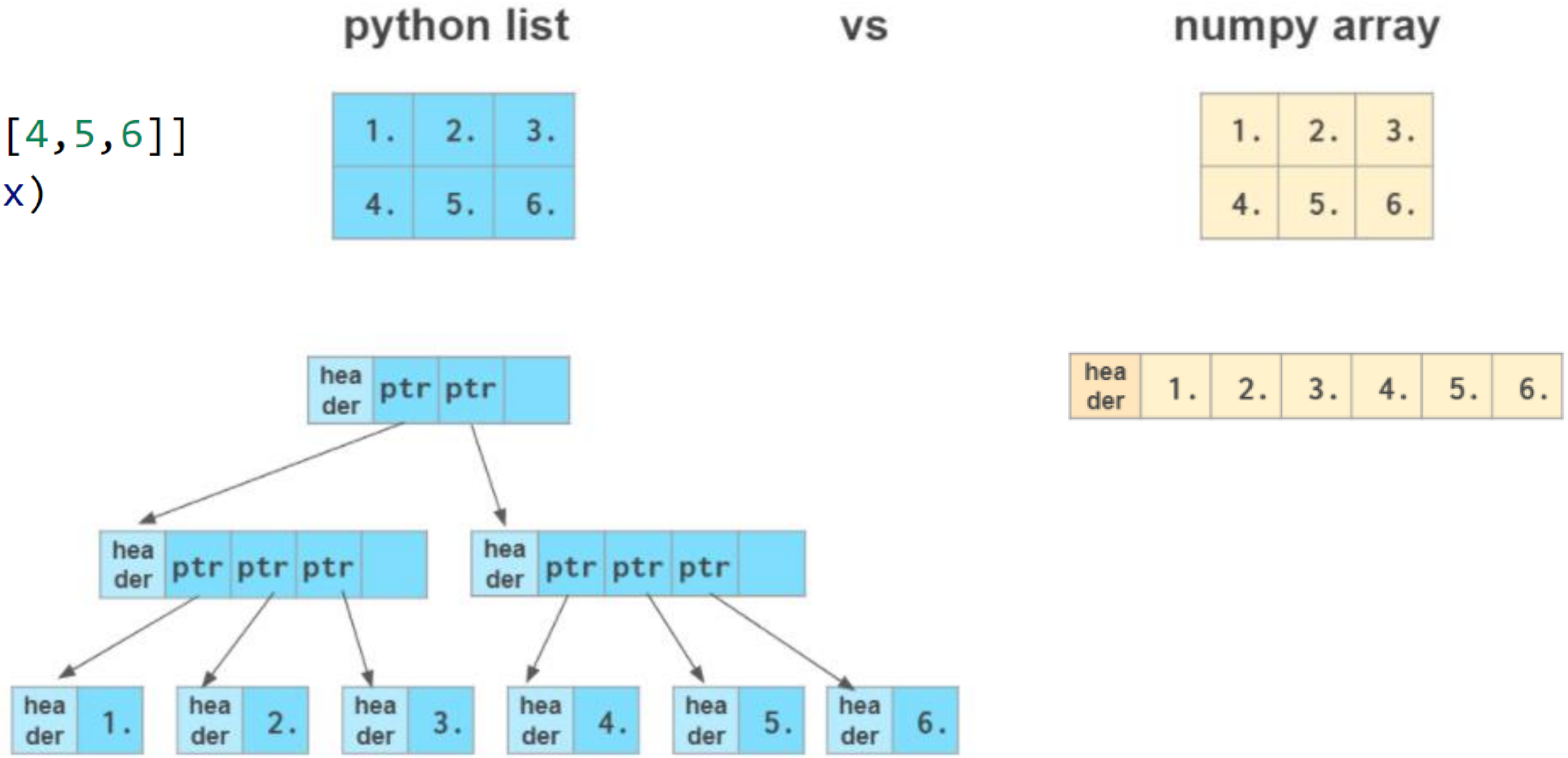
- **„Size-immutable“ / „fixed-size“:**
  - Once the numpy array is created its array size cannot be changed → the number of elements cannot be changed
  - Append, insert, remove leads to creation of a new array
- **“Element-mutable”**
  - Elements within the array can be modified in place → no new array is created
  - Modification of individual elements, slices, or more complex selections (e.g. based on some boolean criterium)

List vs NumPy array

Lists	Numpy Arrays
• Built-in • 1-dimensional • Heterogeneous items • Flexible resizing: adding or deleting elements can often occur in place • Not optimized for Math	• Not built-in • Multi-dimensional • Homogeneous items • Fixed size: adding or deleting items creates a new array • High performance math

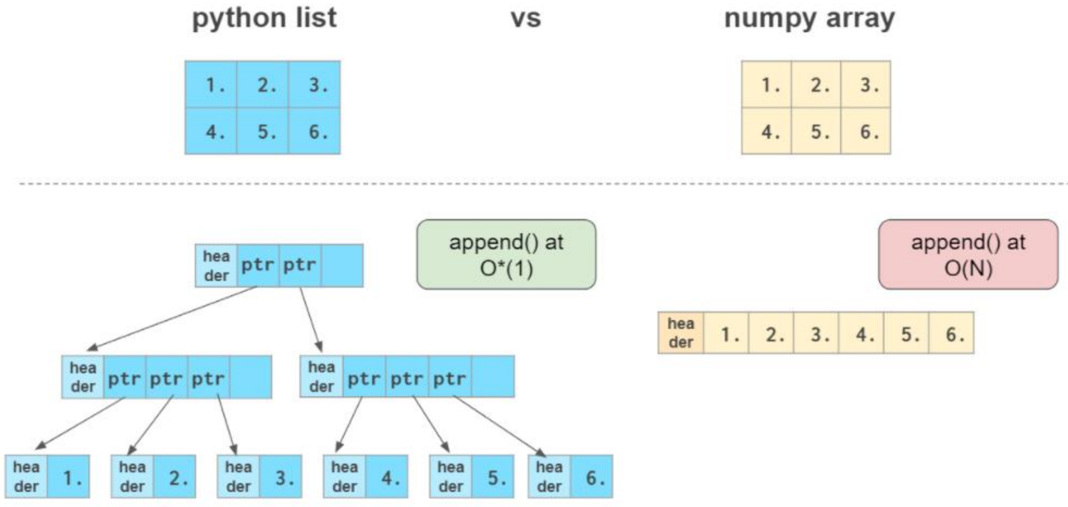
Why are Numpy arrays (usually) faster than lists?

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



Why are Numpy arrays (usually) faster than lists?

- Use **contiguous memory**
- Each element needs **less memory**
- **No type checking** needed when iterating through object
- **Vectorized operations**: For typical operations on arrays, we can avoid Python loops (slow) → Numpy implicitly delegates execution of loops to underlying C code



But: Slower if the size of the object changes (insert, append, delete)

Element-wise operations

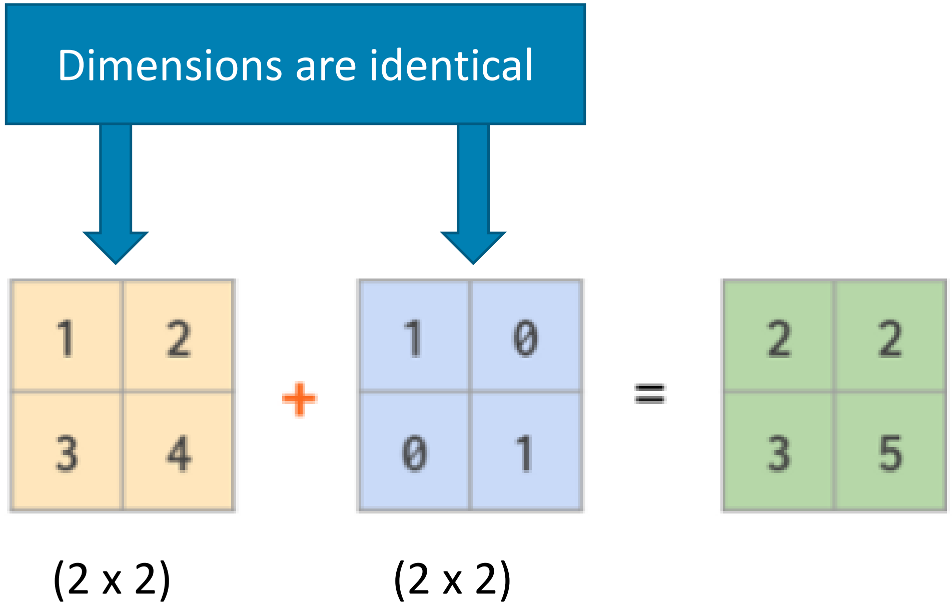
Vectorized operation: no Python loop needed

```python
squared = []
for number in [2, 3]:
    squared.append(number**2)
```

```python
np.array([2, 3])**2
```

$$a^2 \quad = \quad \begin{bmatrix} 2 & 3 \end{bmatrix} \; ** \; 2 \quad = \quad \begin{bmatrix} 4 & 9 \end{bmatrix}$$

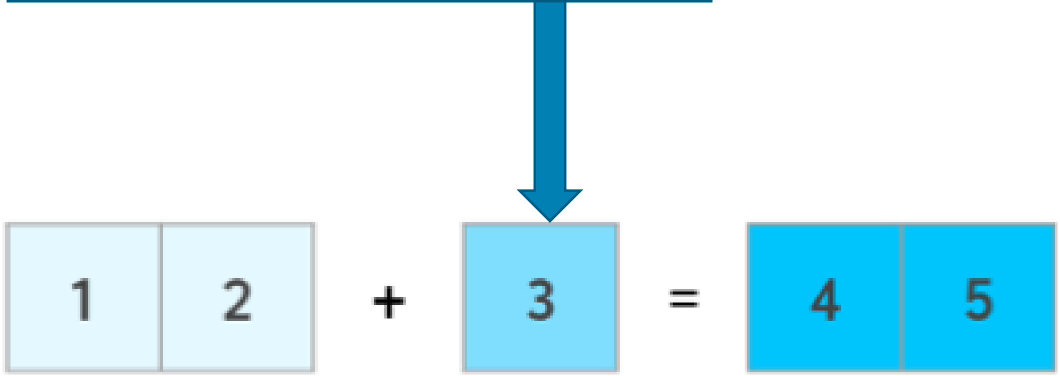
Element-wise operations



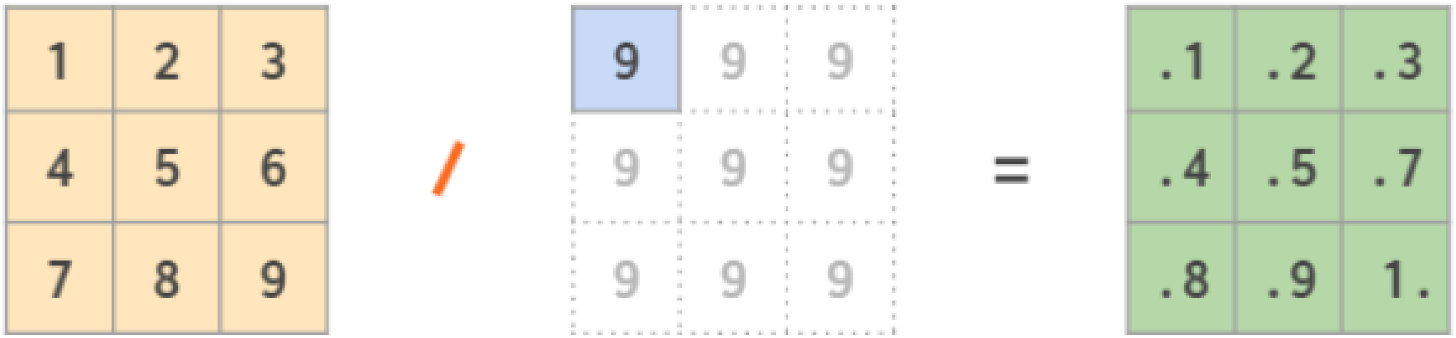

```
x = np.array([[1,2],[3,4]])
y = np.array([[1,0],[0,1]])
x + y
```

Broadcasting

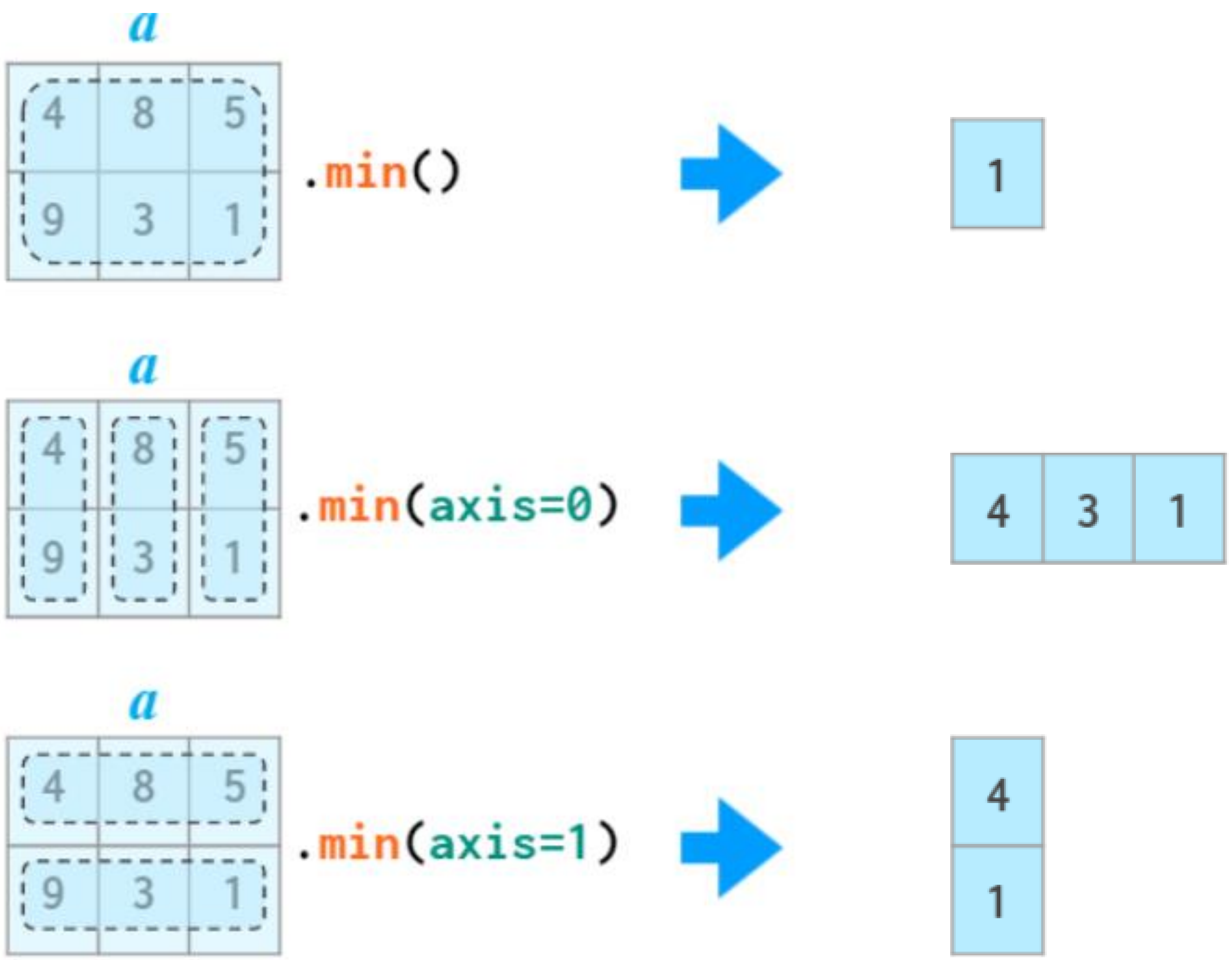
Dimensions of the smaller array are **broadcasted** to fit the dimensions of the larger array




```
np.array([1, 2]) + 3
```



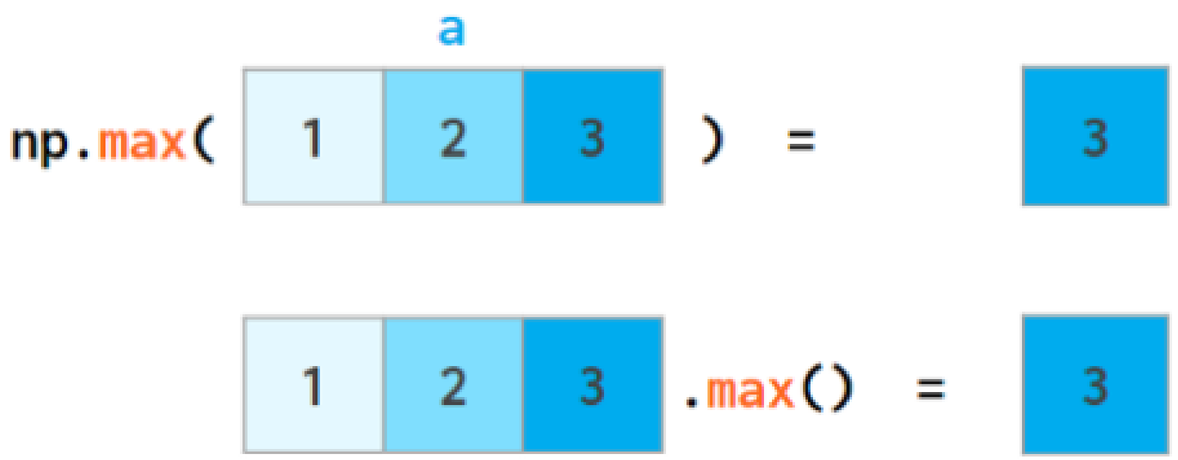
Statistics and the axis Argument



Other statistics / aggregation functions

- min
- max
- mean
- median
- sum
- std
- var
- …

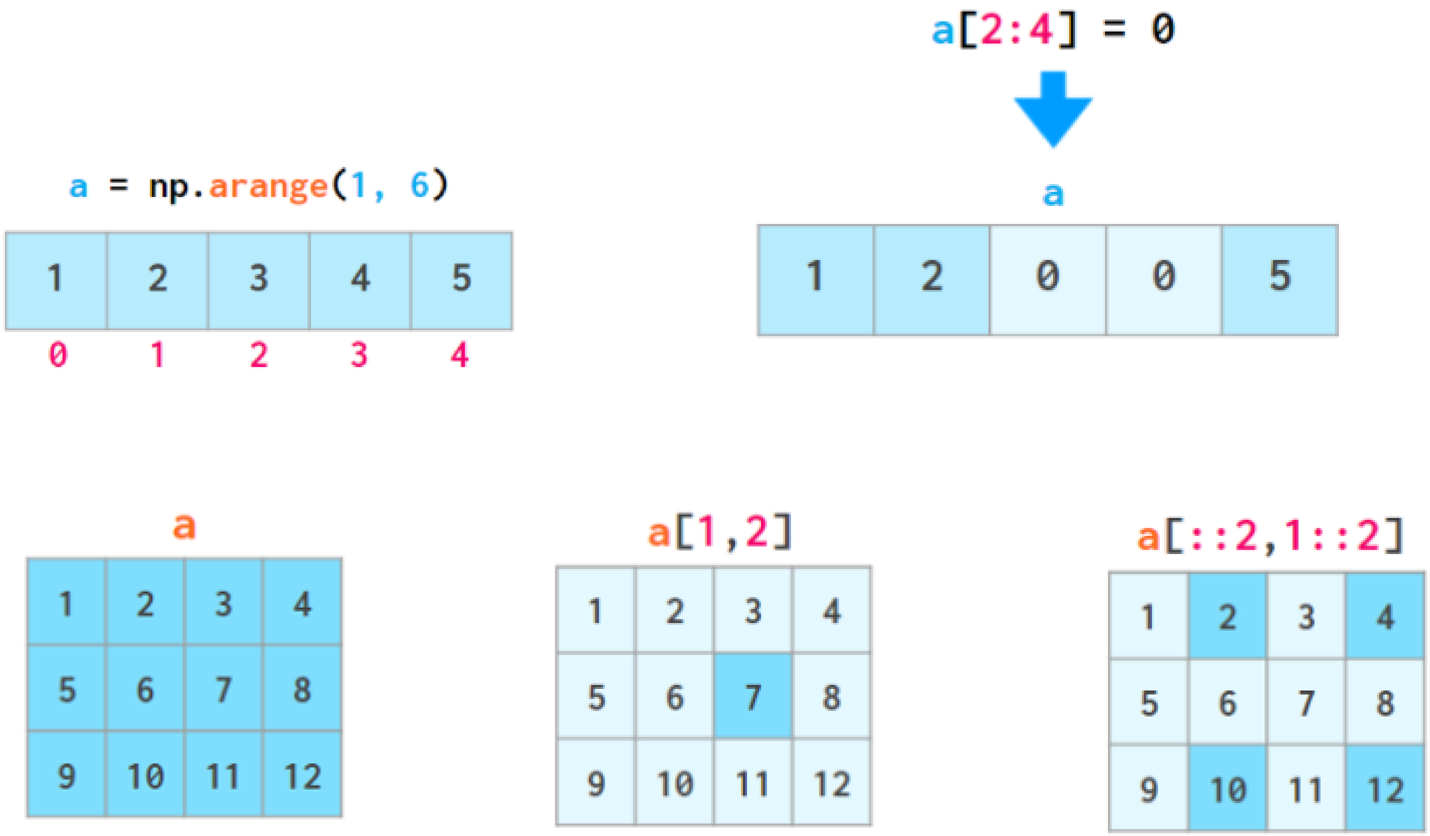
Two alternatives



Top-level function of the numpy module

Method of numpy arrays

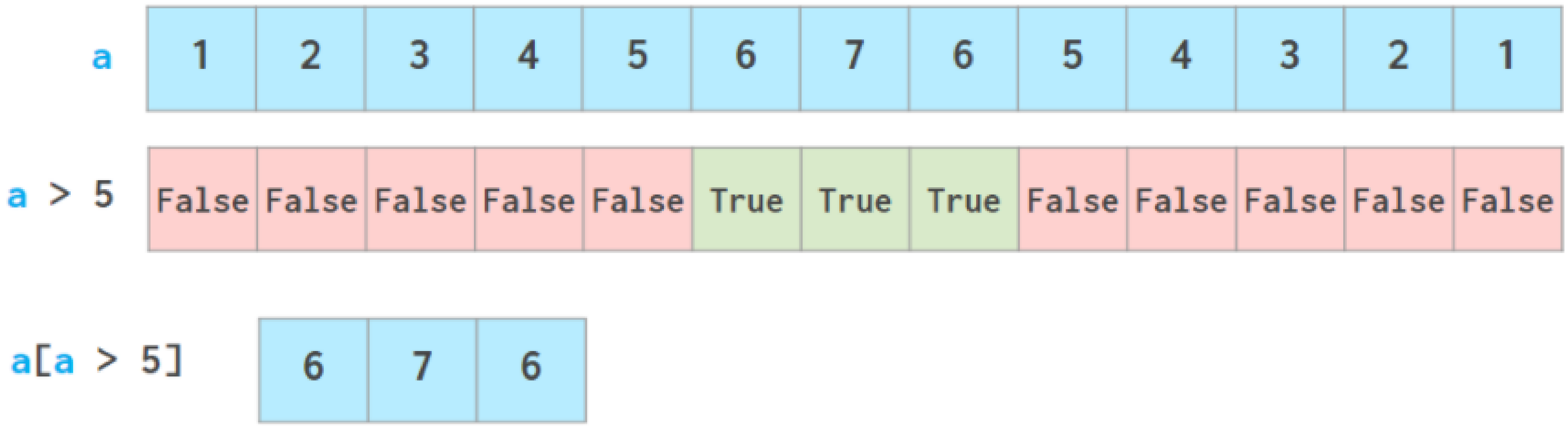
Indexing and Slicing



Similar to lists:

- Same indexing rules
- Applicable to any dimension
- Mutability: elements of an array can be changed in-place → potential side effects between multiple names pointing to the same numpy array

Boolean masking



Linear algebra

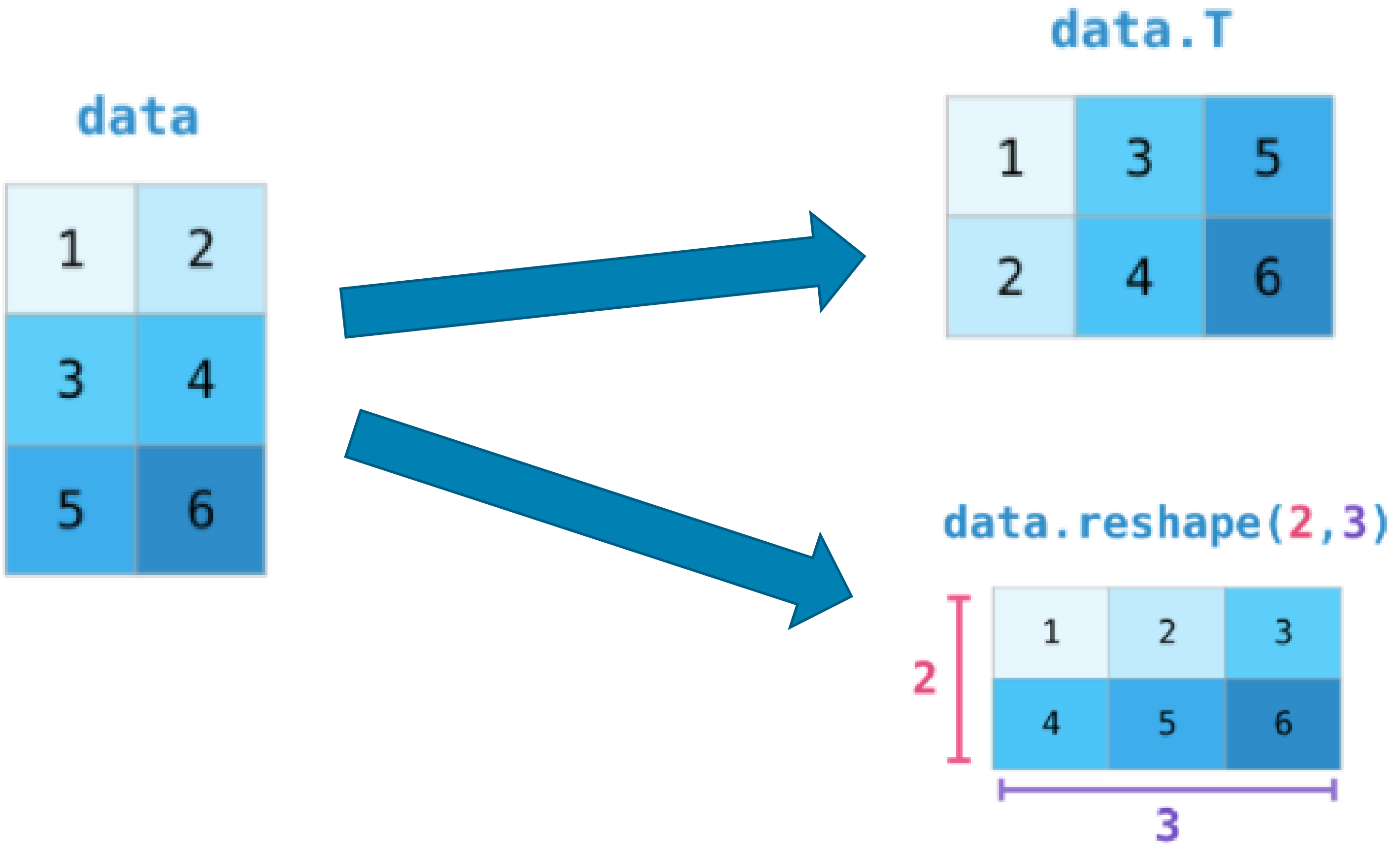
```
x = np.array([[1,2],[5,6],[7,8]])
y = np.array([3,4])

x.dot(y)
x @ y
```

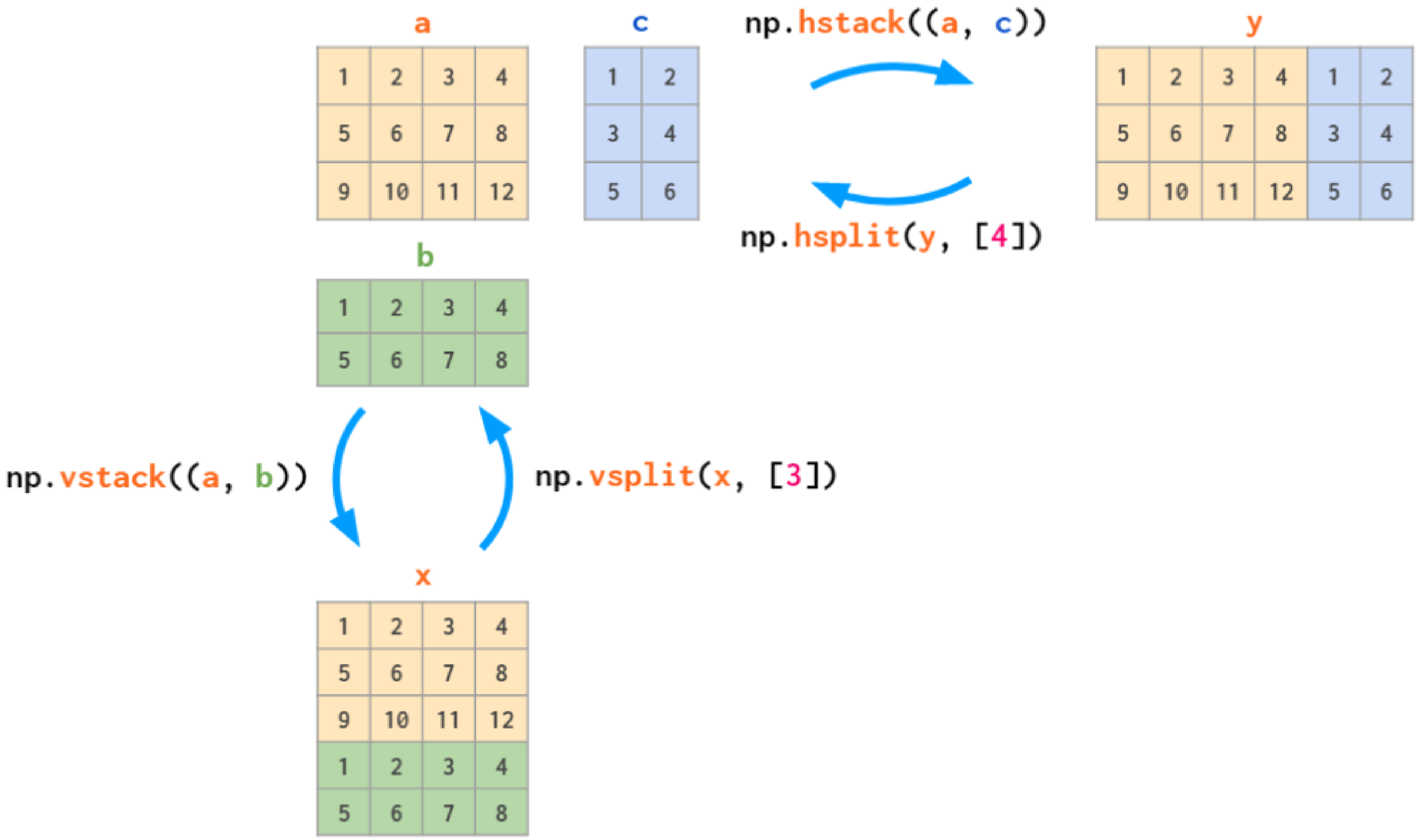
$$\begin{bmatrix}1&2\\5&6\\7&8\end{bmatrix}\cdot\begin{bmatrix}3\\4\end{bmatrix} \longrightarrow \begin{array}{c}\begin{bmatrix}3&4\end{bmatrix}\\ \begin{bmatrix}1&2\\5&6\\7&8\end{bmatrix}\end{array} = \begin{bmatrix}1\cdot3+2\cdot4\\5\cdot3+6\cdot4\\7\cdot3+8\cdot4\end{bmatrix} = \begin{bmatrix}11\\39\\53\end{bmatrix}$$

(3 x 2) (2 x 1) (3 x 1)

Transpose and reshape



Stacking and splitting



References

Visual guides to Numpy:

- https://betterprogramming.pub/numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d
- http://jalammar.github.io/visual-numpy/