Open-Source packages for DA: NumPy (Numerical Python)

Nazgul Rakhimzhanova

IITU 2021

OUTLINE

- **Previously**
- **Problems** for DS
- **Notebooks,** examples
- **Datasets:** formats
- **NumPy package** basics
- **Readings**

PREVIUOSLY

- Discussed about **DS and ML**
- Did overview of the course **syllabus**
- Got familiar with **IDE** (Jupyter Notebook)

What do you remember about each activity?

SYLLABUS

week	Mid Term (weeks 01-07)	End Term (weeks 08-14)	week
01	Intro: Data Science Area and open-source tools for Data Science	Statistics: Distribution – Lognormal, Exponential	08
02	NumPy package for data science	Sampling and Estimation	09
03	Pandas package for data science	Correlation and Covariance	10
04	Visualization with matplotlib	Hypothesis testing	11
05	Statistics: Distribution – Normal	Decision Tree	12
06	Exploratory Data Analysis (EDA)	Linear Regression	13
07	**Summary for 6 weeks QA session**	**Summary for 6 weeks QA session**	**14**
15	**Course summary**		

PREVIUOSLY

- **Data science** - Data science is a multi-disciplinary field that uses scientific methods, processes, algorithms and systems **to extract knowledge and insights from structured and unstructured data.**

How do you understand this?

PREVIUOSLY

- We take some data; we perform some manipulations over them and then we can share more information about the object(s)/situation that data represent or hide.

PREVIUOSLY

- **Lemonade stand**

This is Alan, and he started his business this summer.

Let's see how we can help Alan with DS.

PREVIUOSLY

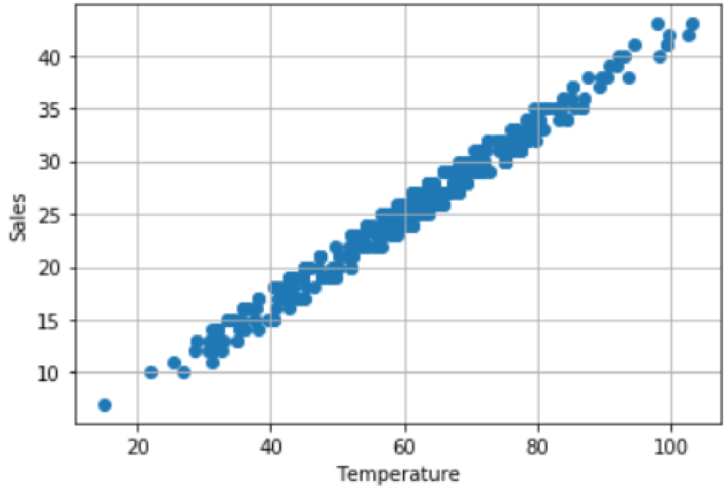
- Lemonade stand data

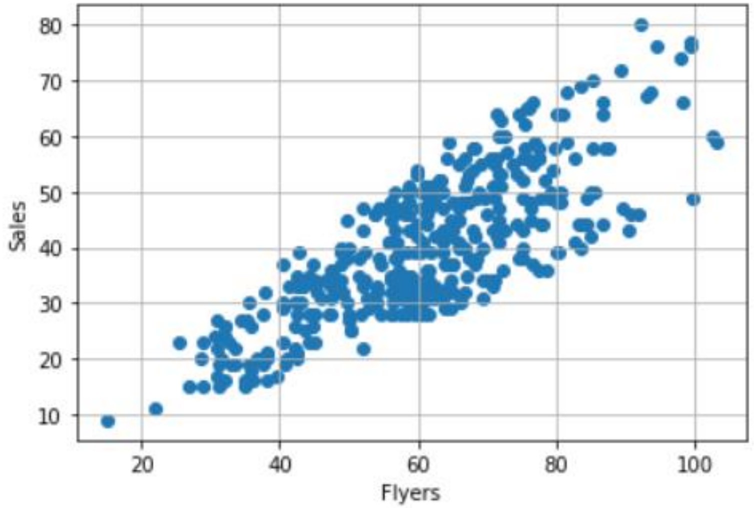
	B	C	D	E	F	G	H
	Date	**Day**	**Temperature**	**Rainfall**	**Flyers**	**Price**	**Sales**
	01.01.2017	Sunday	27	2,00	15	0,3	10
	02.01.2017	Monday	28,9	1,33	15	0,3	13
	03.01.2017	Tuesday	34,5	1,33	27	0,3	15
	04.01.2017	Wednesday	44,1	1,05	28	0,3	17
	05.01.2017	Thursday	42,4	1,00	33	0,3	18
	06.01.2017	Friday	25,3	1,54	23	0,3	11
	07.01.2017	Saturday	32,9	1,54	19	0,3	13
	08.01.2017	Sunday	37,5	1,18	28	0,3	15
	09.01.2017	Monday	38,1	1,18	20	0,3	17

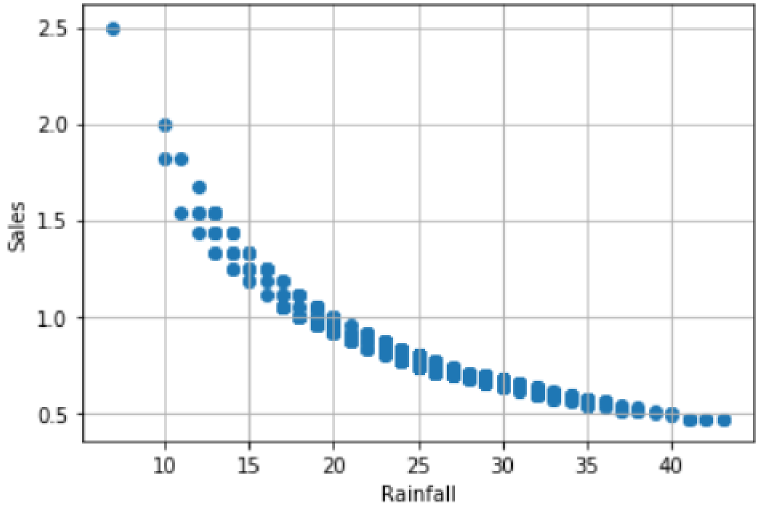
	Temperature	Rainfall	Flyers	Price	Sales
count	365.000000	365.000000	365.000000	365.000000	365.000000
mean	60.731233	0.826603	40.284932	0.333973	25.323288
std	16.196266	0.273171	13.178651	0.075206	6.893589
min	15.100000	0.470000	9.000000	0.300000	7.000000
25%	49.700000	0.650000	31.000000	0.300000	20.000000
50%	61.100000	0.740000	39.000000	0.300000	25.000000
75%	71.300000	0.910000	49.000000	0.300000	30.000000
max	102.900000	2.500000	80.000000	0.500000	43.000000

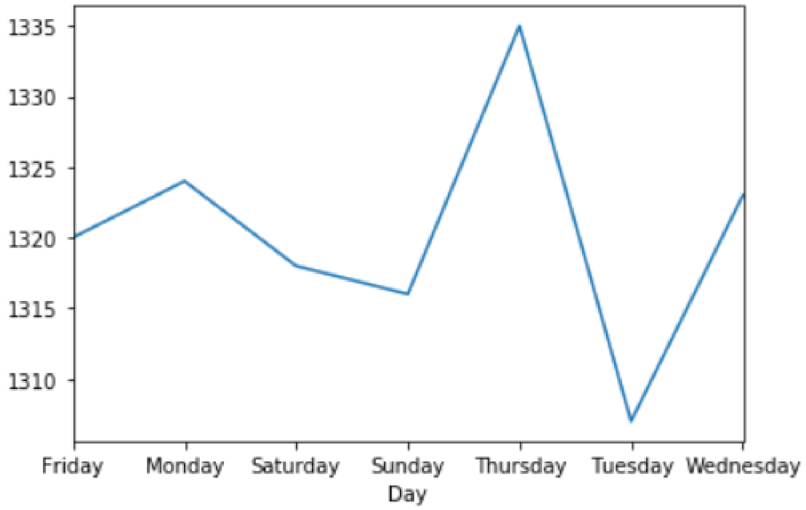
PREVIUOSLY

- Lemonade stand – **descriptive analysis**









Problems DS can solve

- **Prediction:** traffic, flood, disease, earthquakes, election outcome, sales etc.
- **Detection:** fraud, illegal immigrants, suspicious individuals etc.

Notebooks, examples

- https://anaconda.org/dask/dask-dataframe-hdfs/notebook

Datasets

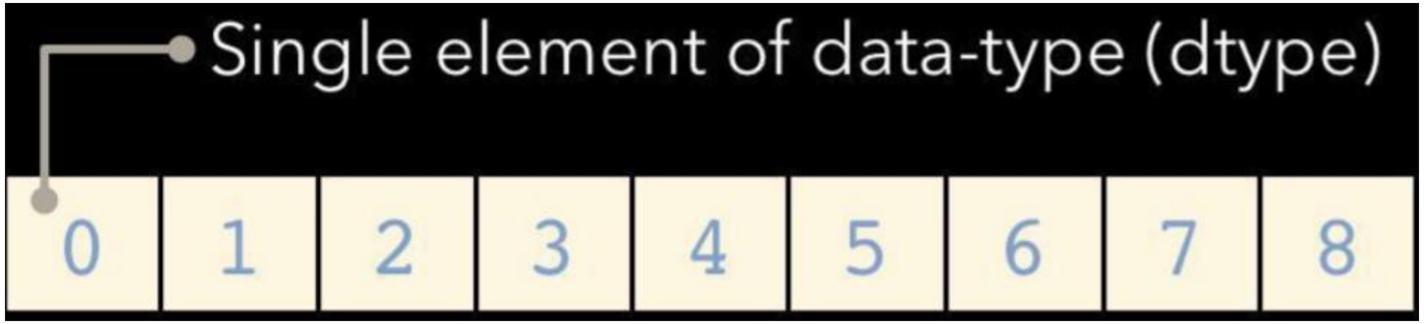
Here are some examples of what can qualify as a dataset:

- A table or a CSV file with some data
- An organized collection of tables
- A file in a proprietary format that contains data
- A collection of files that together constitute some meaningful dataset
- A structured object with data in some other format that you might want to load into a special tool for processing
- Images capturing data
- Files relating to machine learning, such as trained parameters or neural network structure definitions
- Anything that looks like a dataset to you

NUMPY

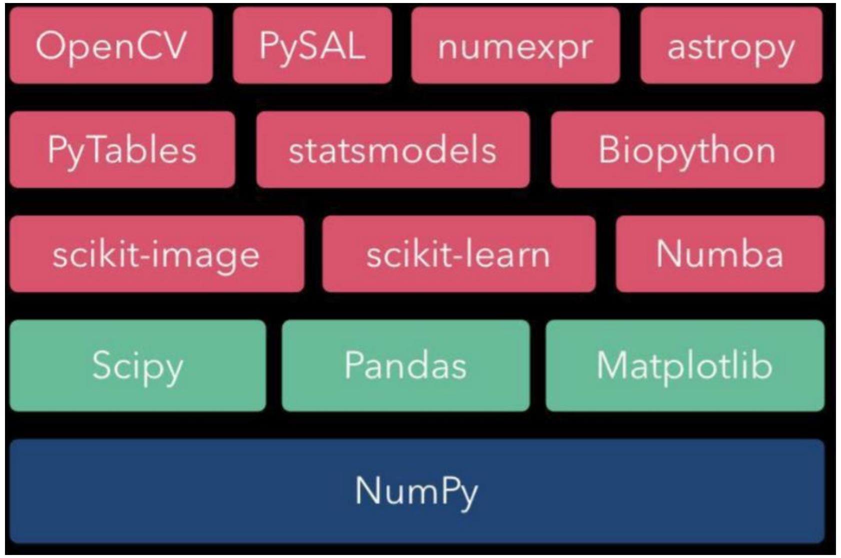
NumPy

- NumPy is a Python C extension library for **array-oriented computing**
  - Efficient
  - In-memory
  - Contiguous (or Stridden)
  - Homogeneous (but types can be algebraic)



NumPy

- NumPy is suited for many applications
  - Image processing
  - Signal processing
  - Linear algebra etc.



NumPy

- Before you will use any package in Python you need to import it.

```
import numpy as np
```

NumPy

Arrays

numpy arrays are dense, continuous, uniformly sized blocks of identically typed data values

In [73]:
```
import numpy as np
L = [[0,1],[2,3]]
A = np.array(L)
```

In [74]:
```
print("L:",L)
print("A:\n",A)
```

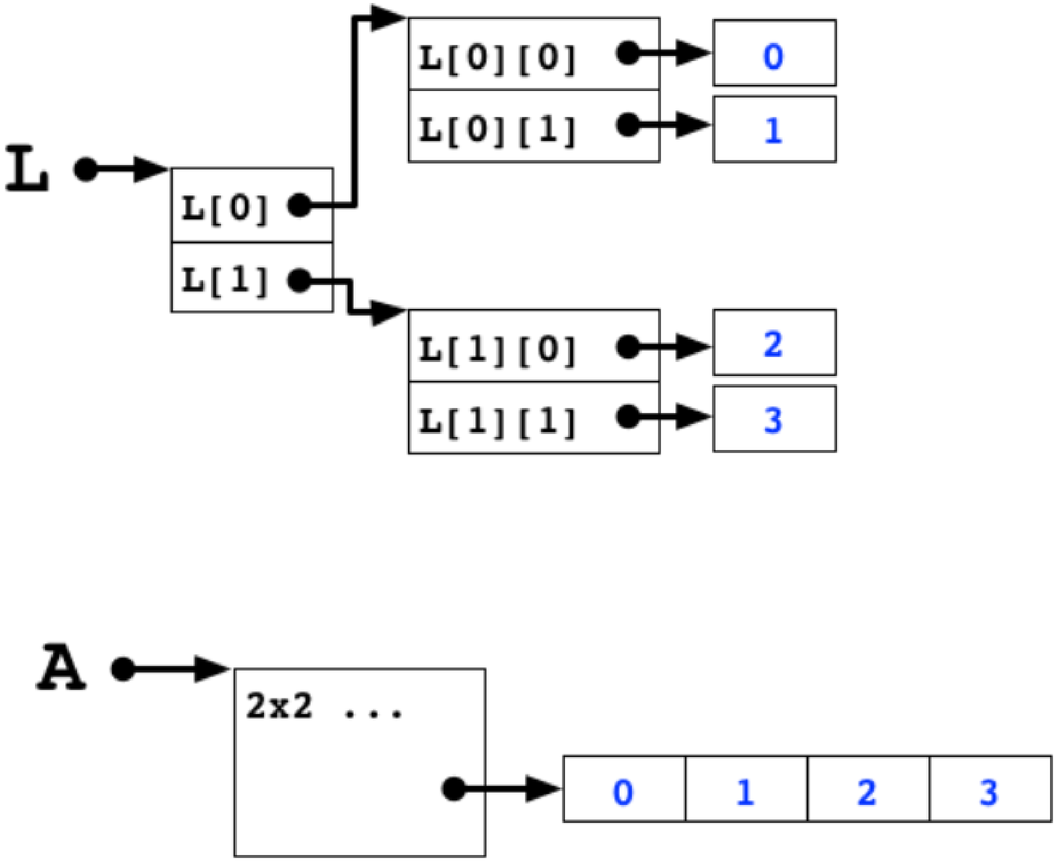
```
L: [[0, 1], [2, 3]]
A:
 [[0 1]
 [2 3]]
```

In [75]:
```
print(type(L),type(A))
```

```
<class 'list'> <class 'numpy.ndarray'>
```

NumPy

Array Memory Layout



NumPy

Why does this matter?

Keeping data close together results in faster access times.

- It's easier to figure out the location of the data
- The data is more likely to fit in the processor's *cache*

If you have a *block* of *dense* numerical data, store it in a numpy array

NumPy

Creating numpy Arrays

Note that `np.ndarray` and `np.array` are the same thing.

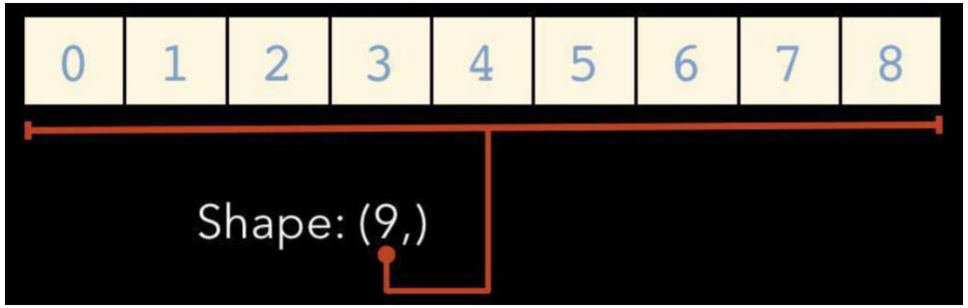
In [79]:
```
A = np.array([1,2,3,4])
A.dtype #type of what is stored in the array - NOT python types!
```
Out[79]:
```
dtype('int64')
```

In [80]:
```
A.ndim #number of dimensions (axes in numpy speak)
```
Out[80]:
```
1
```

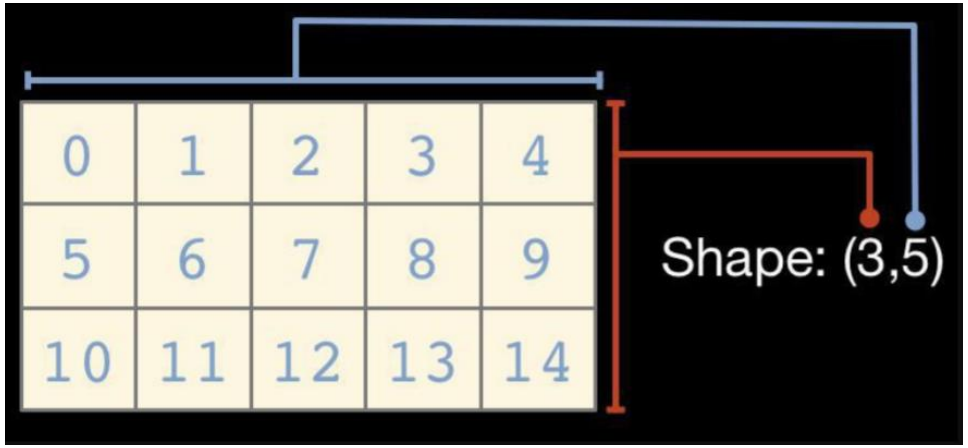
In [81]:
```
A.shape #size of the dimensions as a tuple
```
Out[81]:
```
(4,)
```

In [82]:
```
A.reshape((4,1)).shape #a column vector
```
Out[82]:
```
(4, 1)
```

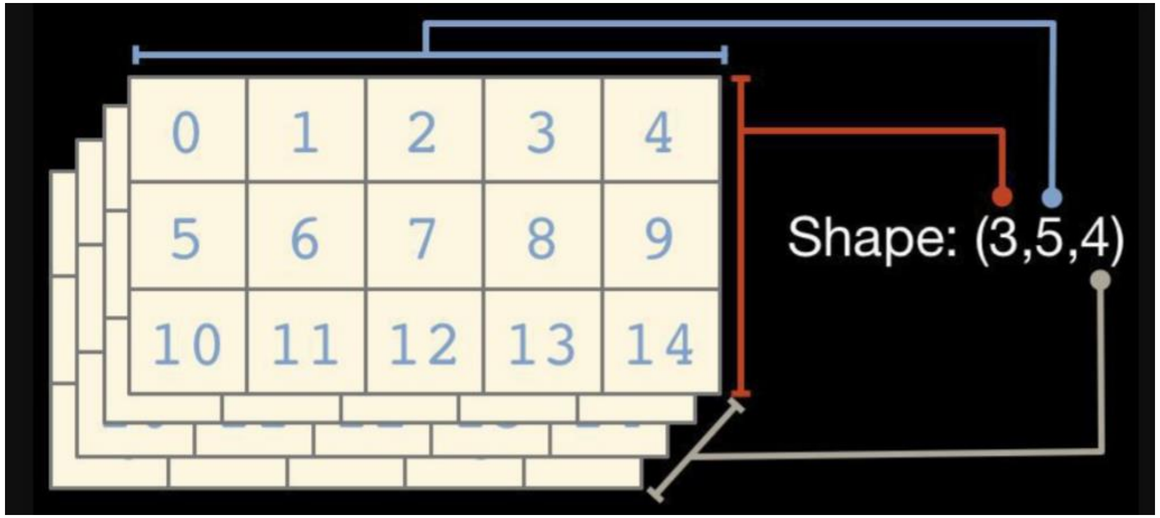
NumPy



0 1 2 3 4 5 6 7 8
Shape: (9,)



0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
Shape: (3,5)



0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
Shape: (3,5,4)

NumPy

Initializing numpy Arrays

In [87]:
```python
#can initialize an array with a list, or list of lists (or list of lists of lists, etc)
M = np.array([[1,2,3],[4,5,6.0]])
print(M.dtype,M.shape)
```

```
float64 (2, 3)
```

In [88]:
```python
#if know the size, but not the data, can initialize to zeros:
Z = np.zeros((10,10))
#or ones
O = np.ones((5,10))
#or identity
I = np.identity(3) #this makes a 3x3 square identity matrix
```

In [89]:
```python
print(Z.dtype) #note, default type is floating point
```

```
float64
```

In [90]:
```python
Z = np.zeros((10,10),np.int64) #can change
print(Z.dtype)
```

```
int64
```

Indexing and Slicing

numpy arrays can be indexed and sliced a lot like python lists, but take **tuples** of values to reference each dimension.

In [91]:
```
M = np.array([[0,1,2],[3,4,5]])
M
```

Out[91]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [92]:
```
print(M[1,1]) #indexing
print(M[0,-1]) #last item of first row
```

```
4
2
```

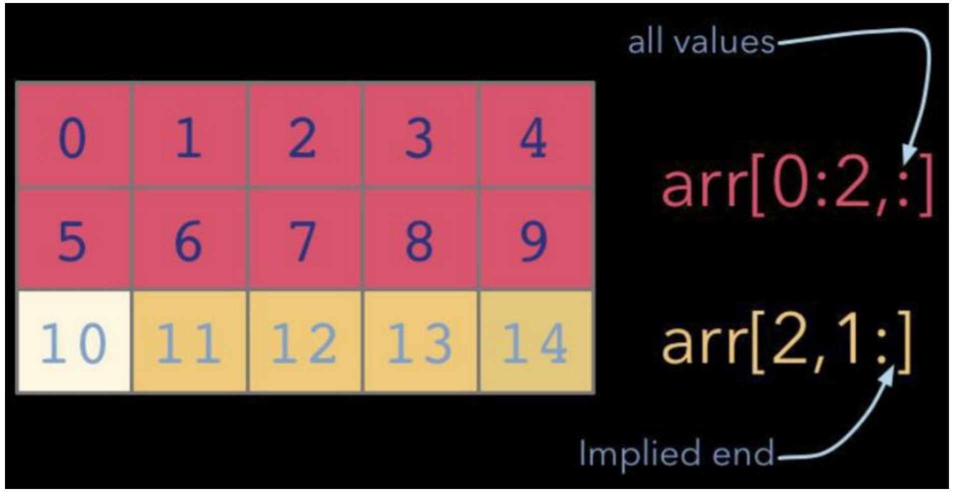
In [93]:
```
print(M[0,1:]) #can have slices - all but first column of first row
```

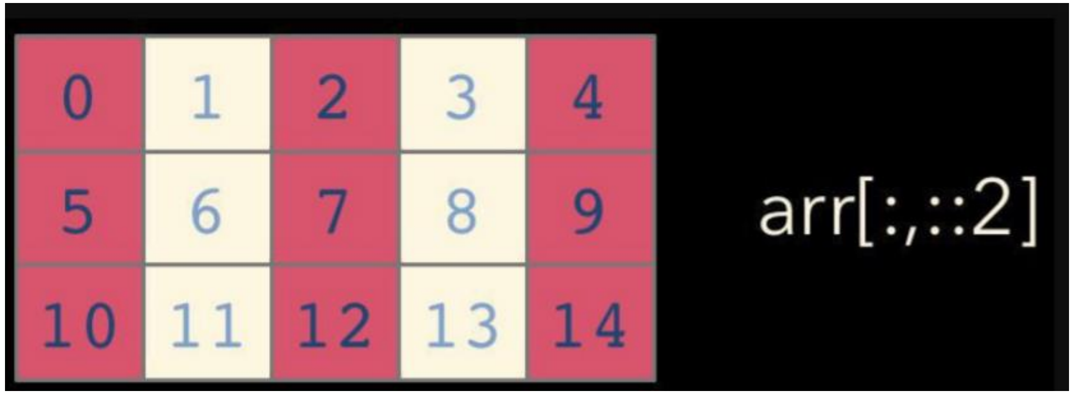
```
[1 2]
```

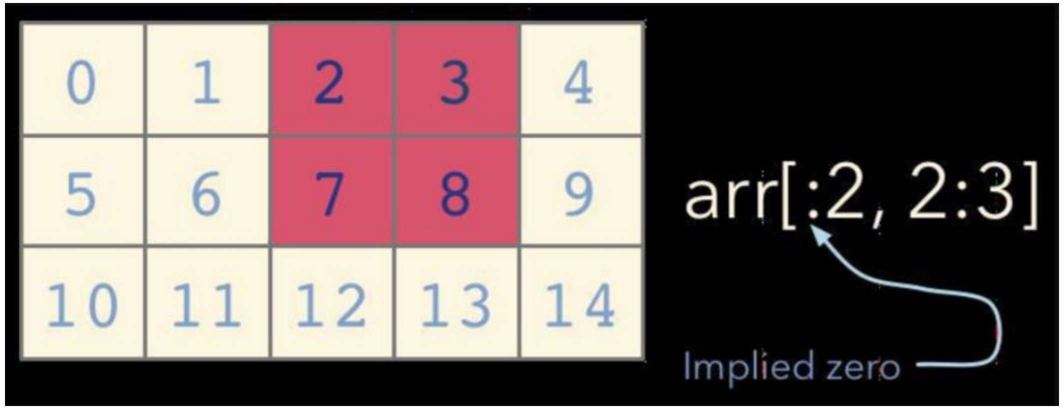
In [94]:
```
print(M[1],M[1,:]) #missing indices are treated as complete slices
```

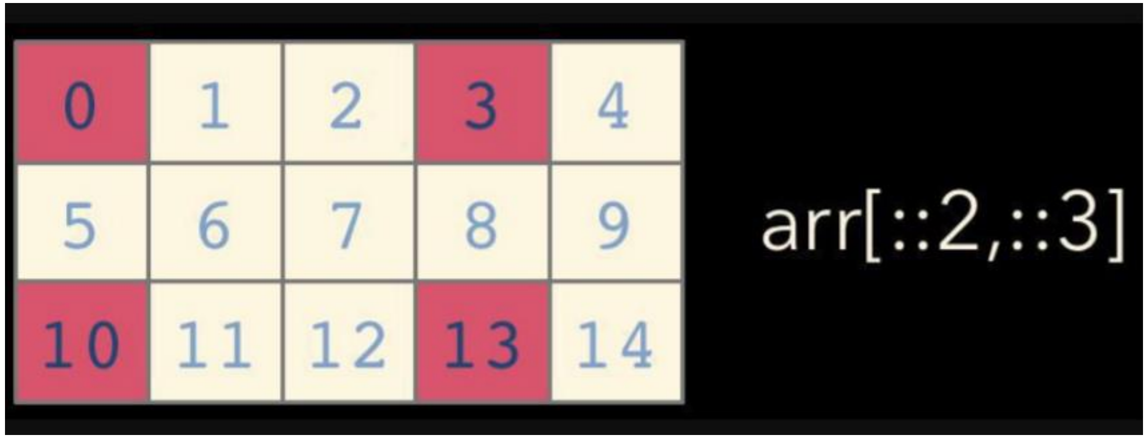
```
[3 4 5] [3 4 5]
```

NumPy: Slicing









Advanced Slicing: Boolean

Indexing by **Boolean** *numpy arrays* can be used to select elements

In [99]:
```
b = A > 4
print(b)
```

```
[False False False  True  True  True]
```

In [100]:
```
print(A[b])
```

```
[ 9 16 25]
```

NumPy

Slicing Assignment

In [101]:
```
print("b =",b)
A[b] = 0
```

```
b = [False False False  True  True  True]
```

In [102]:
```
print(A)
```

```
[0 1 4 0 0 0]
```

NumPy

Array Views vs. Copies

- A numpy array object has a pointer to a dense block of memory that stores the data of the array.
- *Basic* slices are just *views* of this data - they are **not** a new copy.
- Binding the same object to different variables will **not** create a copy.
- *Advanced* slices will create a copy if bound to a new variable - these are cases where the result may contain elements that are not contiguous in the original array

```
In [105]: A = np.array([[0,1,2],[3,4,5],[6,7,8]])

In [106]: B = A #A and B reference the _same_ object
          A is B

Out[106]: True

In [107]: B[0,0] = 1000
          A

Out[107]: array([[1000,    1,    2],
                 [   3,    4,    5],
                 [   6,    7,    8]])
```

```
In [110]: newMat = A.copy() #this will actually copy the data
          newMat[0,0] = 0
          A

Out[110]: array([[1000,    1,    2],
                 [   3,    4, 5000],
                 [   6,    7,    8]])

In [111]: newMat

Out[111]: array([[   0,    1,    2],
                 [   3,    4, 5000],
                 [   6,    7,    8]])
```

NumPy

Advanced Slices Copy

In [112]:
```
A = np.array([[0,1,2],[3,4,5],[6,7,8]])
B = A[A > 4]
B
```

Out[112]:
```
array([5, 6, 7, 8])
```

In [113]:
```
B[:] = -1
B
```

Out[113]:
```
array([-1, -1, -1, -1])
```

In [114]:
```
A
```

Out[114]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

NumPy: Delete

Deletion

```
In [18]: arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

In [19]: arr

Out[19]: array([[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12]])

In [20]: np.delete(arr, 1, 0)

Out[20]: array([[ 1,  2,  3,  4],
                [ 9, 10, 11, 12]])

In [35]: np.delete(arr, np.s_[::2], 1)

Out[35]: array([[ 2,  4],
                [ 6,  8],
                [10, 12]])

In [36]: np.delete(arr, [1,3,5], None)

Out[36]: array([ 1,  3,  5,  7,  8,  9, 10, 11, 12])
```

NumPy: Delete

By using boolean mask

```
In [32]: arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
         arr

Out[32]: array([[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12]])

In [27]: mask = np.ones(len(arr), dtype=bool)

In [28]: mask[[0, 1]] = False

In [29]: res = arr[mask,...]
         res

Out[29]: array([[ 9, 10, 11, 12]])
```

NumPy: Functions

Functions on Arrays

numpy includes a number of standard functions that will work on arrays

In [118]:
```
A = [1,2,3,4]
np.mean(A)
```
Out[118]:
```
2.5
```
In [119]:
```
np.sum(A)
```
Out[119]:
```
10
```
In [120]:
```
np.sin(A)
```
Out[120]:
```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

NumPy: Axis

Axis

Most aggregation operations take an `axis` parameter that limits the operation to a specific direction in the array

- axis 0: across rows (apply operation to individual columns)
- axis 1: across columns (apply operation to individual rows)

In [121]:
```
b = np.arange(12).reshape(3,4); b
```

Out[121]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

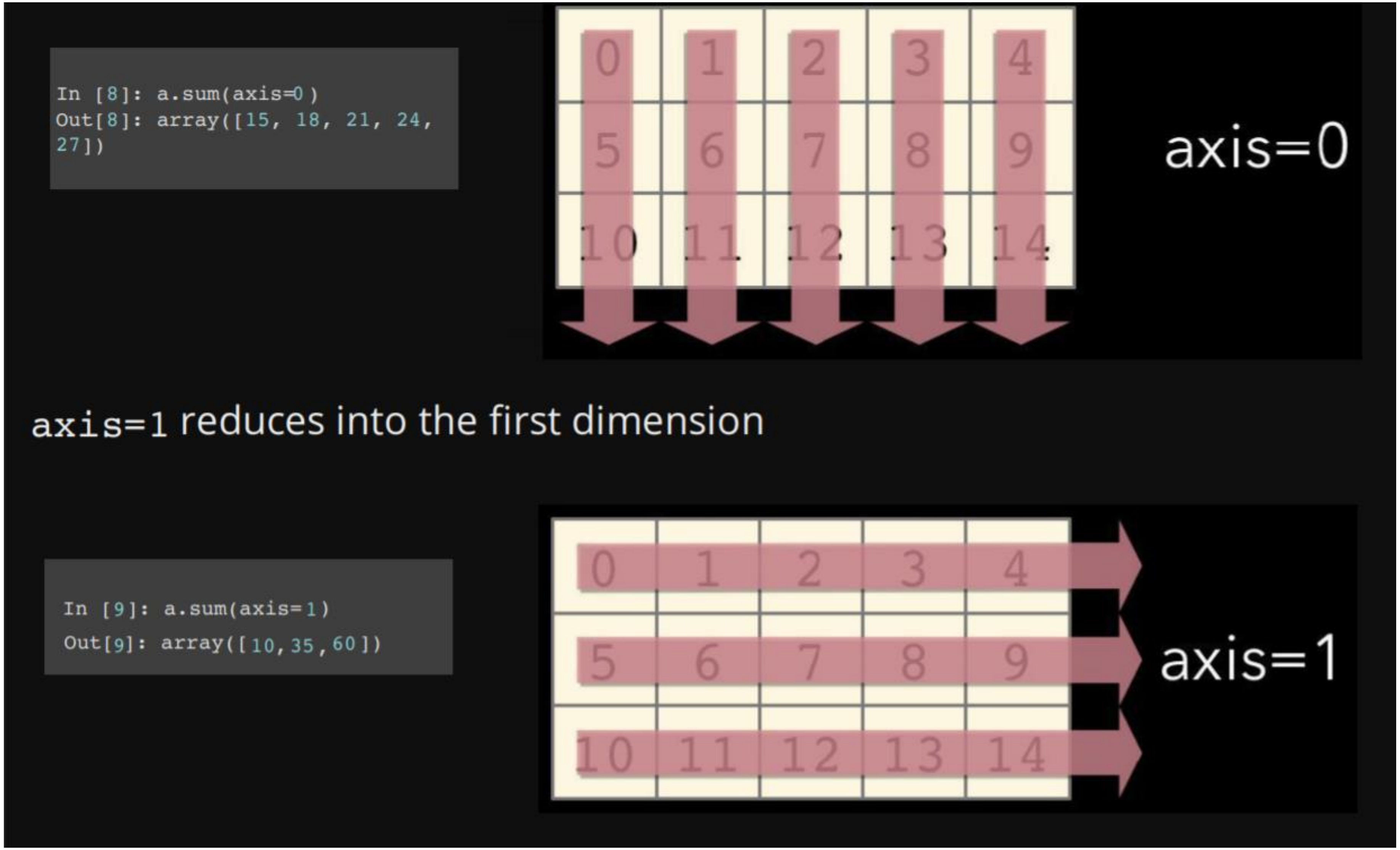
In [122]:
```
np.sum(b)
```

Out[122]:
```
66
```

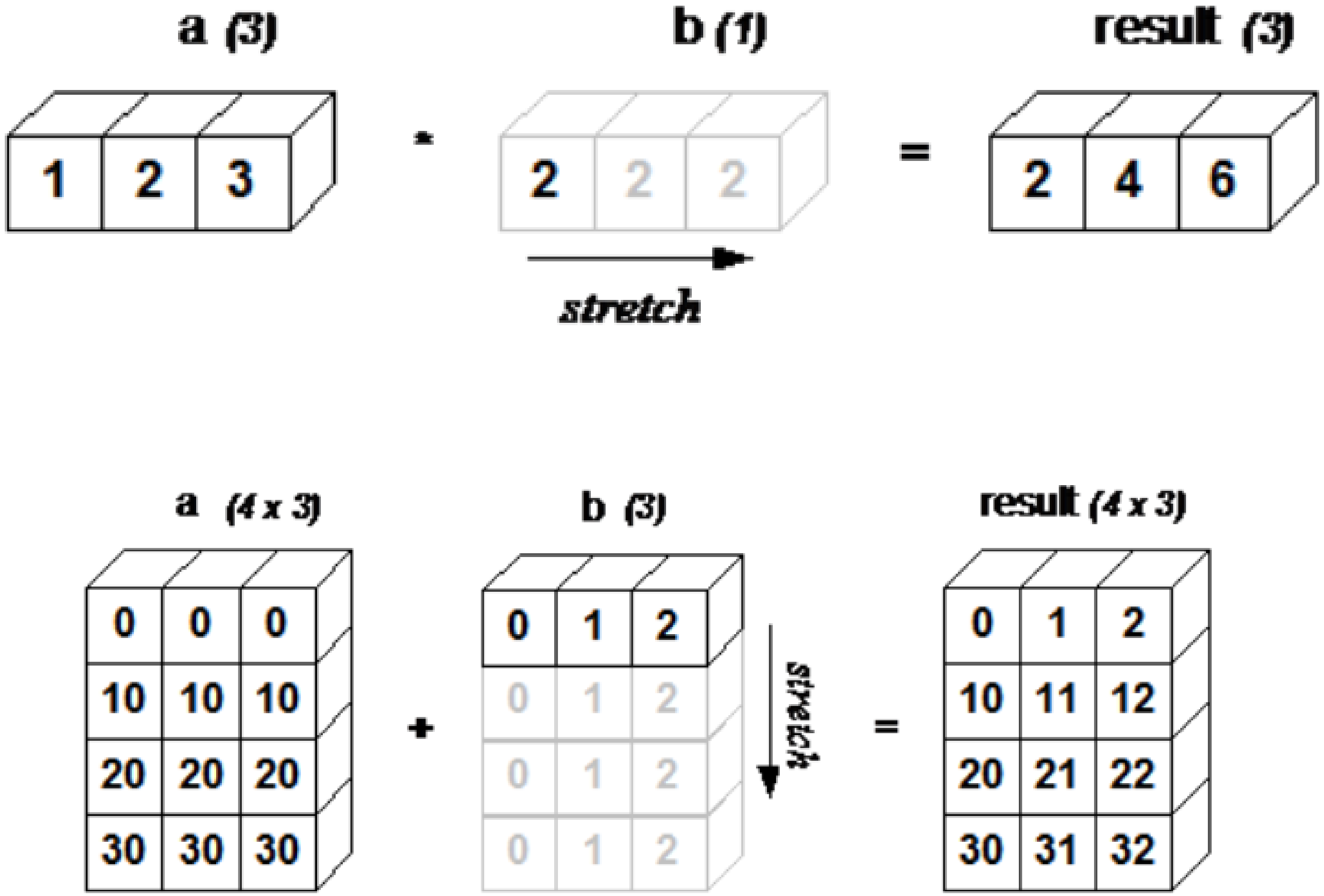
In [123]:
```
np.sum(b,axis=0)
```

Out[123]:
```
array([12, 15, 18, 21])
```

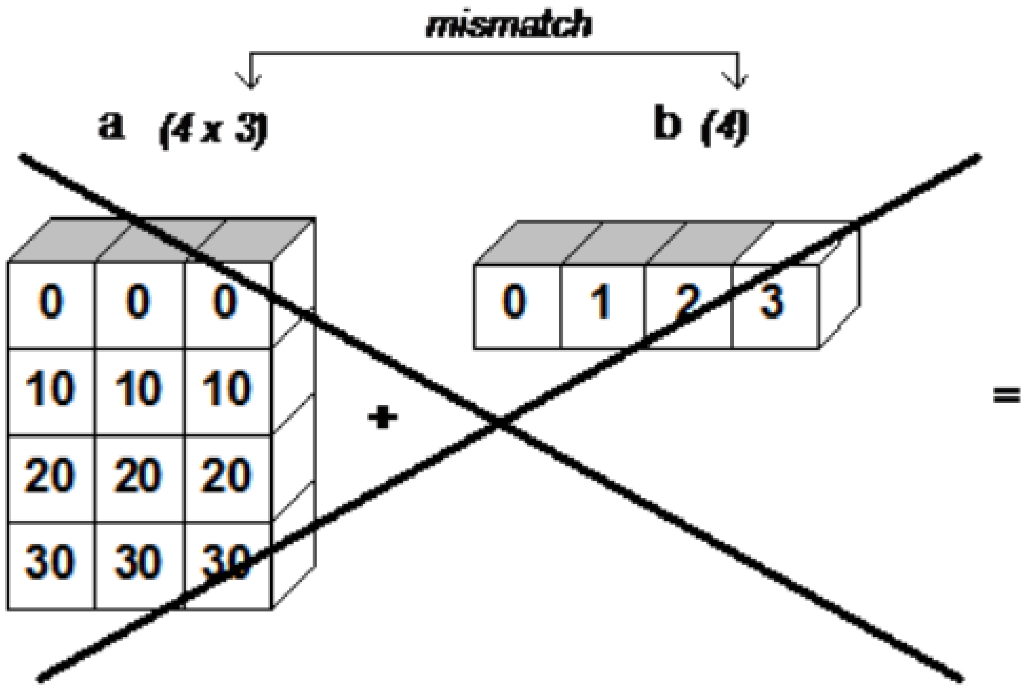
NumPy: Axis


In [8]: a.sum(axis=0)
Out[8]: array([15, 18, 21, 24, 27])
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
axis=0
axis=1 reduces into the first dimension
In [9]: a.sum(axis=1)
Out[9]: array([10, 35, 60])
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
axis=1

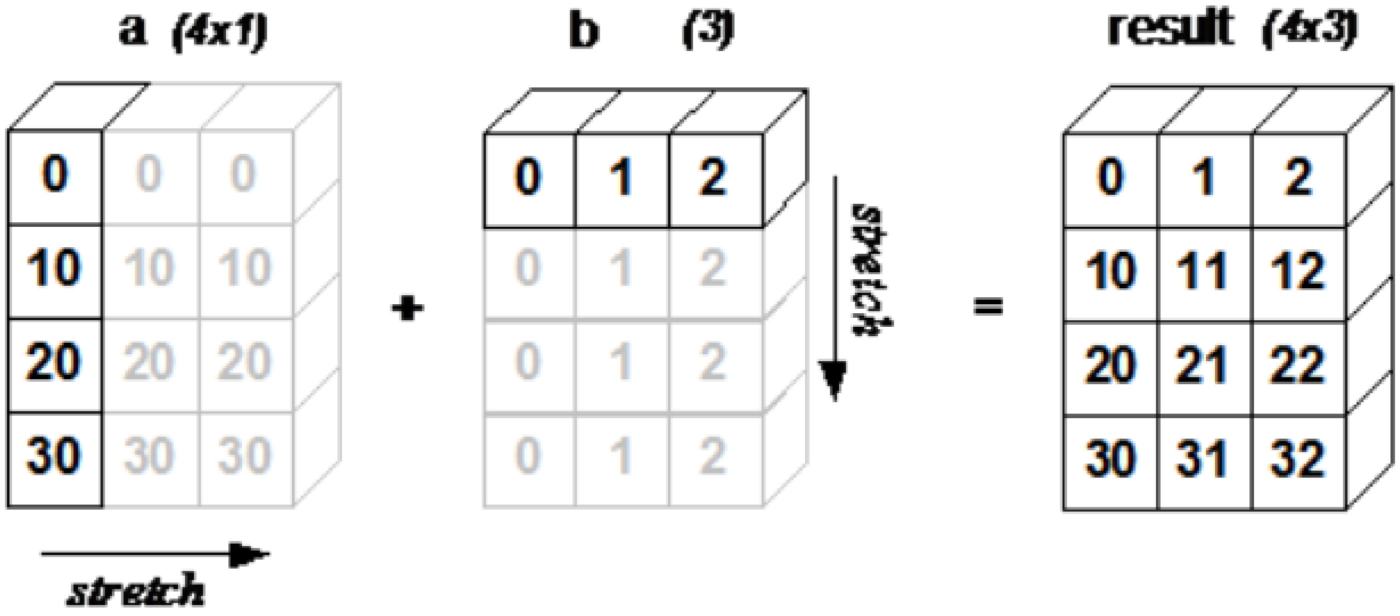
NumPy: Broadcasting



NumPy: Broadcasting



NumPy: Broadcasting



Labs

Laboratory01

Readings

- https://numpy.org/devdocs/user/theory.broadcasting.html
- https://numpy.org/devdocs/user/absolute_beginners.html
- **Data Science from Scratch**, Book by Joel Grus

Additional resources

- Khan academy