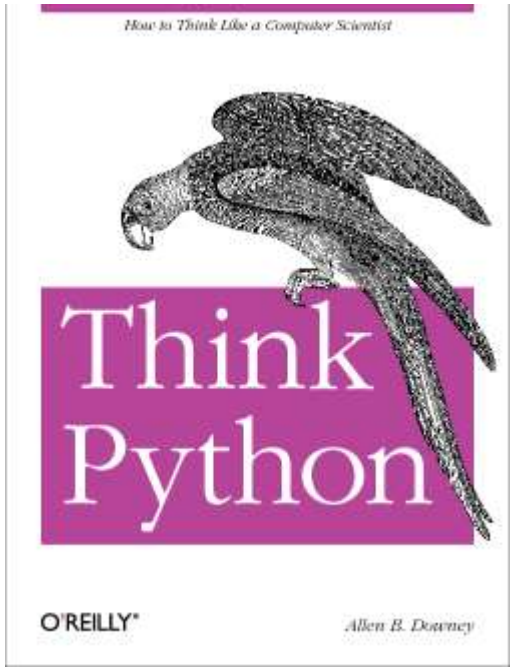
Lecture:
Introduction to Python

Alireza Akhavanpour

PyLab.akhavanpour.ir

Free book



- http://greenteapress.com/thinkpython/thinkpython.pdf
- http://greenteapress.com/thinkpython/html/index.html

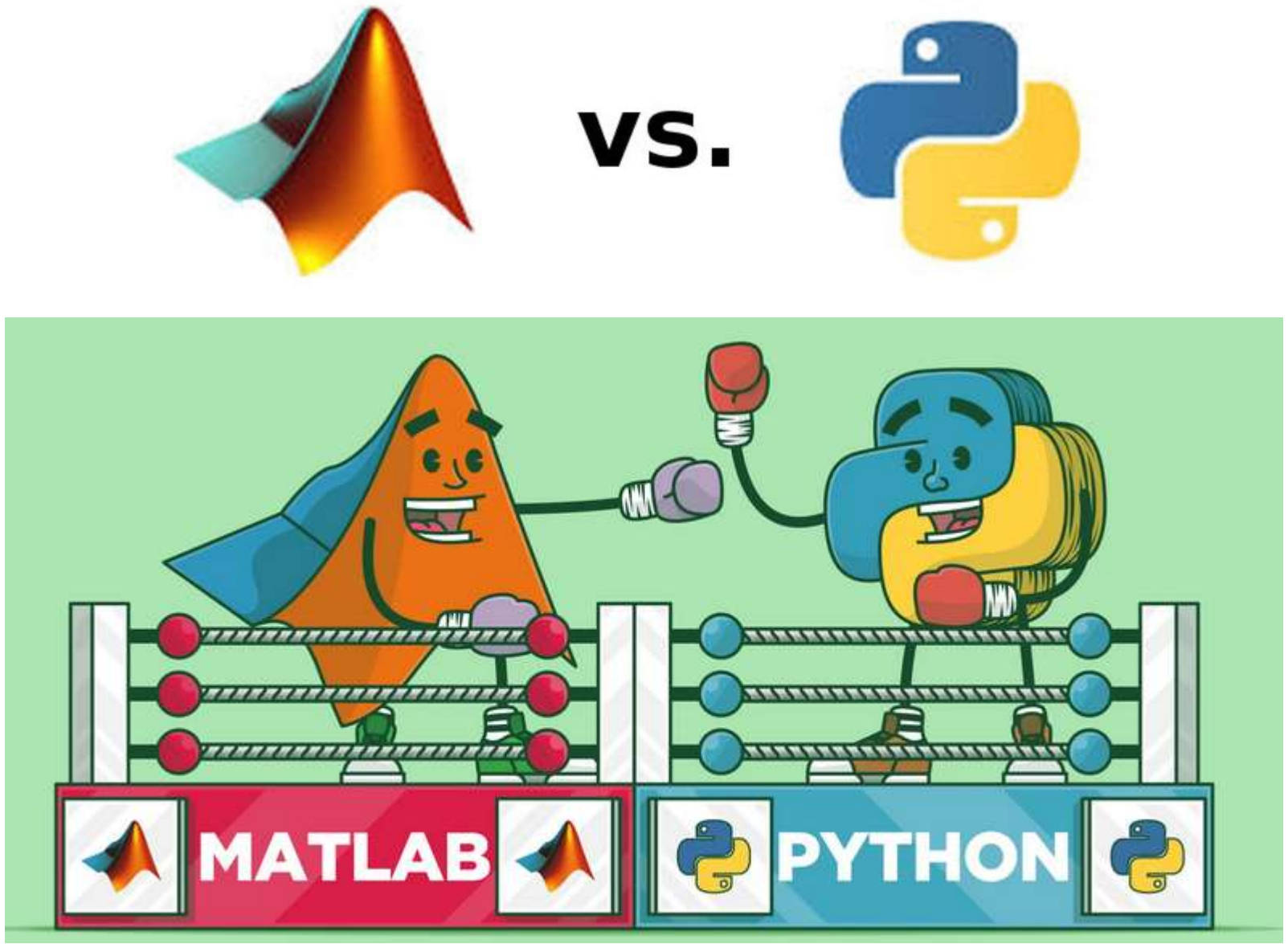
The Python Tutorial

The Python Tutorial

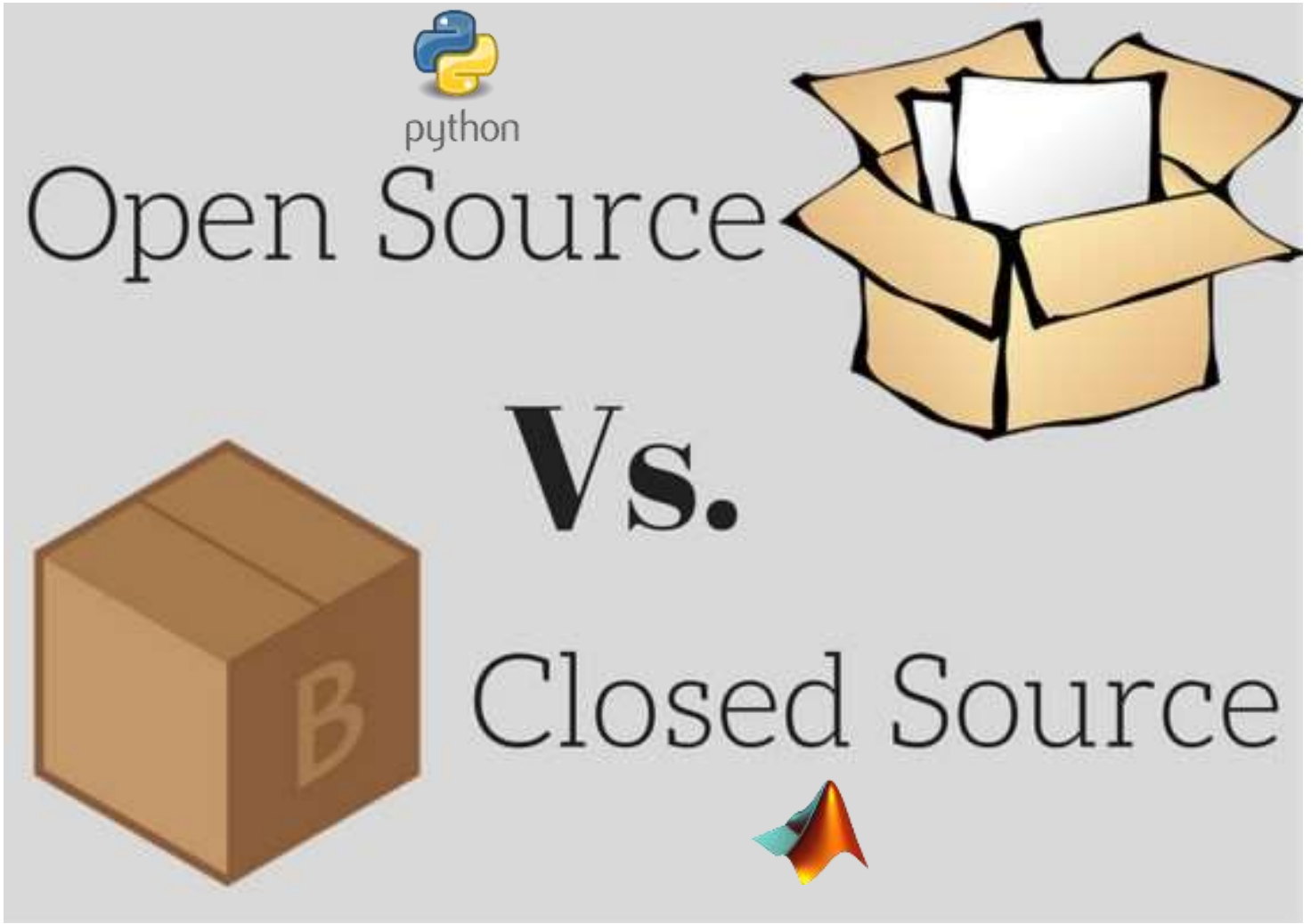
Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

- **https://docs.python.org/3/tutorial/index.html**


vs.
MATLAB
PYTHON

Open source vs close source



Who uses python?

- Reddit
- Dropbox,
- original Google algorithm was written in Python
- the data to create the 2019 photo of a black hole was processed in Python
- Netflix use Python in their data analytics work
- …

python
Search term

matlab
Search term

Interest over time

https://trends.google.com/trends/explore?date=all&q=python,matlab

deep learning

عبارت جستجو

عبارت‌های جستجوی مرتبط

deep learning machine learning	1
deep machine learning	2
machine learning	3
python deep learning	4
deep learning network	5

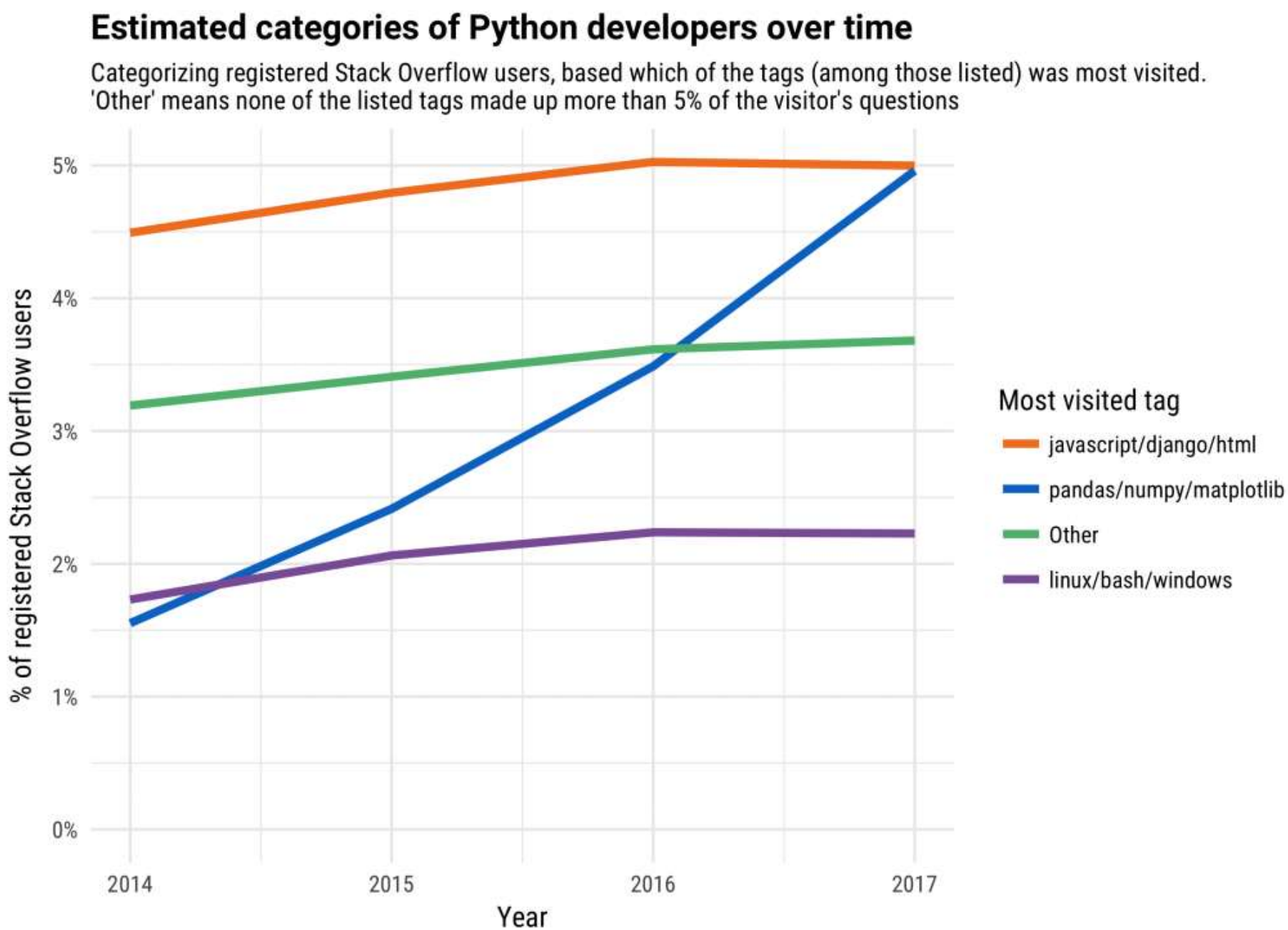
machine learning

عبارت جستجو

عبارت‌های جستجوی مرتبط

machine learning python	1
python	2
what is machine learning	3
machine learning google	4
machine learning pdf	5

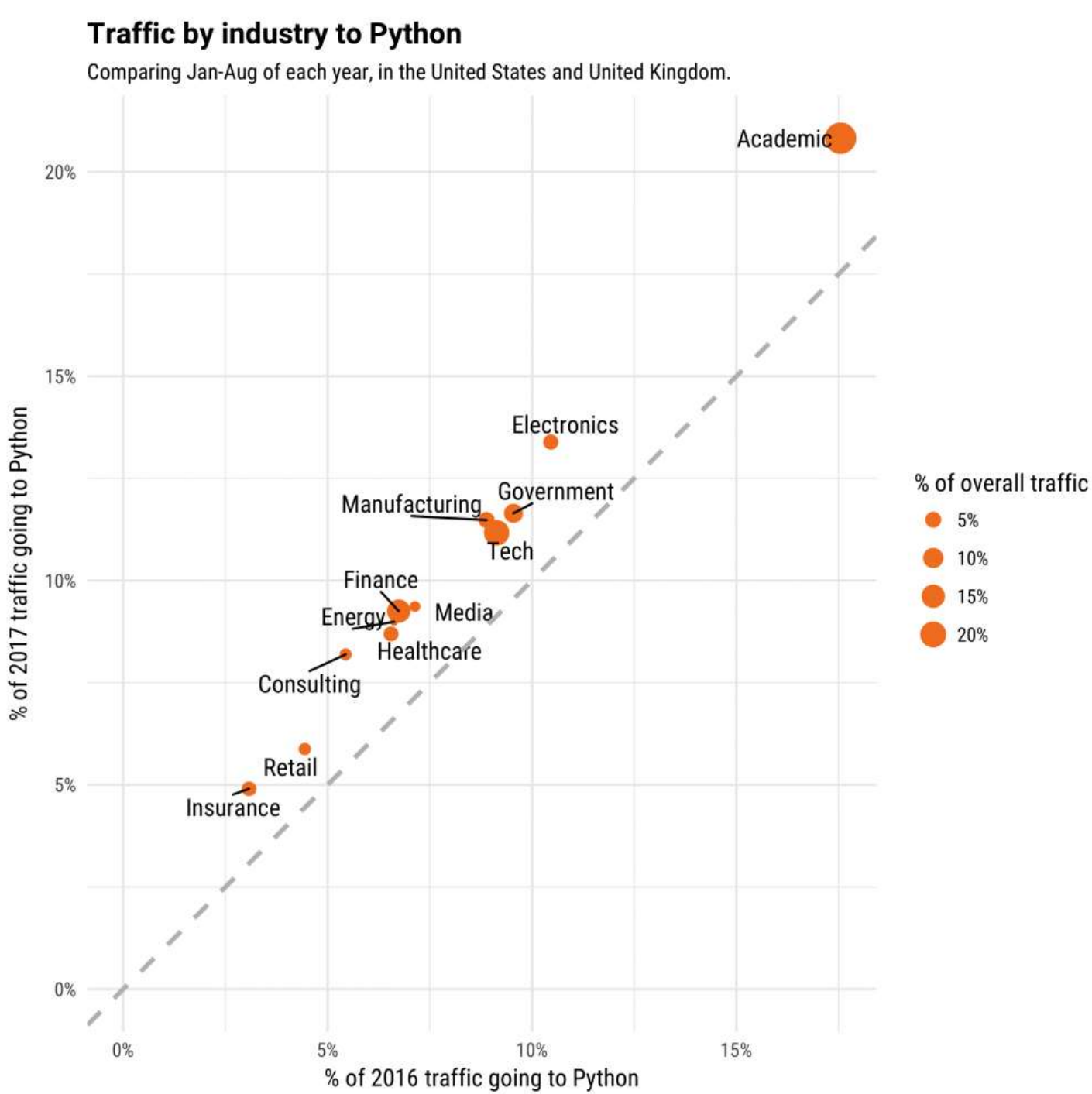
https://trends.google.com/trends/explore?date=all&q=machine%20learning
https://trends.google.com/trends/explore?date=all&q=deep%20learning

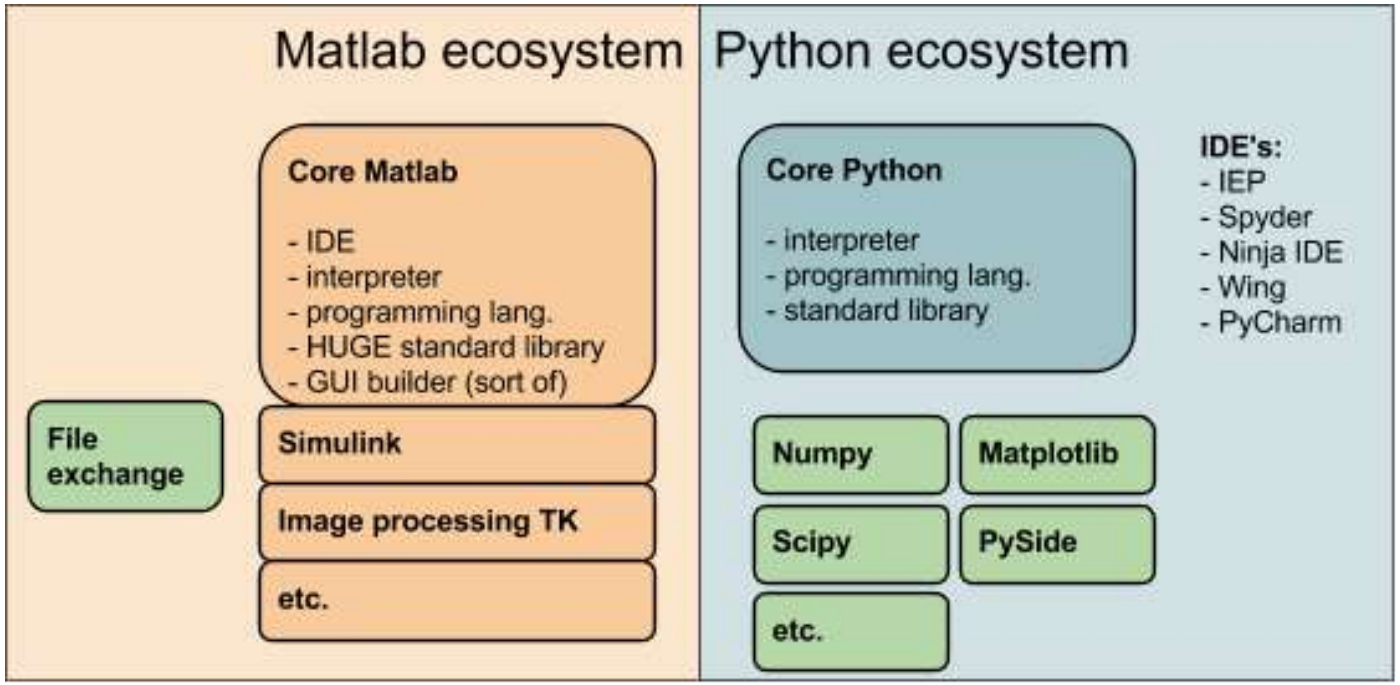


https://stackoverflow.blog/2017/09/14/python-growing-quickly/

These analyses suggest two conclusions. First, **the fastest-growing use of Python is for data science, machine learning and academic research**. This is particularly visible in the growth of the pandas package, which is the fastest-growing Python-related tag on the site. As for which industries are using Python, we found that it is more visited in a few industries, such as electronics, manufacturing, software, government, and *especially* universities. However, **Python's growth is spread pretty evenly across industries**. In combination this tells a story of data science and machine learning becoming more common in many types of companies, and Python becoming a common choice for that purpose.

https://stackoverflow.blog/2017/09/14/python-growing-quickly/


Traffic by industry to Python
Comparing Jan-Aug of each year, in the United States and United Kingdom.
Academic
Electronics
Government
Manufacturing
Tech
Finance
Energy
Media
Healthcare
Consulting
Retail
Insurance
20%
15%
10%
5%
0%
0%
5%
10%
15%
% of 2017 traffic going to Python
% of 2016 traffic going to Python
% of overall traffic
5%
10%
15%
20%



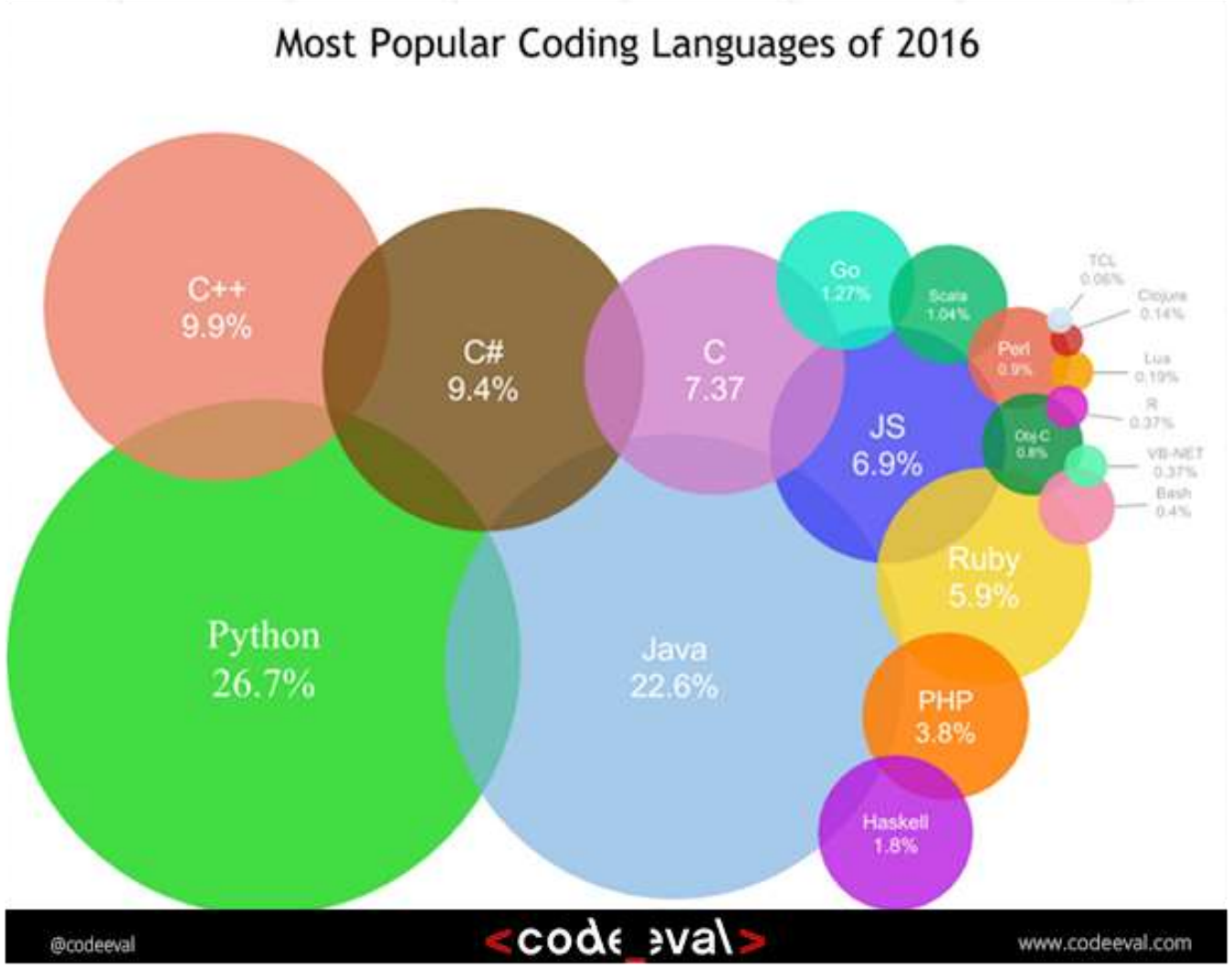
A commercial product called **Matlab** is also used extensively and has a long(er) **history** in both **engineering** and **science**. **Python** is **younger** but has since matured into a product which **out-performs Matlab in many fields**.

https://www.linkedin.com/pulse/matlab-vs-python-jan-rhebergen/

http://www.pyzo.org/python_vs_matlab.html

http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html

Why Python(2016 statistics)?

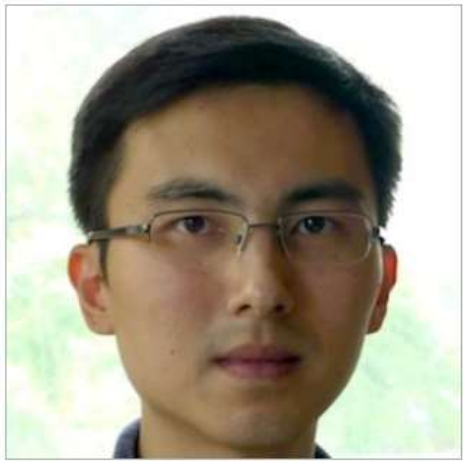


Python was and still is the MOST popular, high-level, open-source programming language!

BLOG@CACM

Python Is Now the Most Popular Introductory Teaching Language at Top U.s. Universities

By Philip Guo
July 7, 2014
Comments (12)



Summary

At the time of writing (July 2014), Python is currently the most popular language for teaching introductory computer science courses at top-ranked U.S. departments.

Specifically, eight of the top 10 CS departments (80%), and 27 of the top 39 (69%), teach Python in introductory CS0 or CS1 courses.

Motivation

Python has been getting more popular as the first language to

2019

The 10 most popular programming languages, according to the Microsoft-owned GitHub

#2: Python

Python is both one of the most popular programming languages and one of the fastest growing ones. In terms of popularity, it went from third place last year to second place in 2019. This open source language is frequently used for artificial intelligence applications and data science, but it's also known to be easy to get started with. There's a large community around Python, and even conferences and meetups dedicated to it.

Shutterstock

https://www.businessinsider.de/international/most-popular-programming-languages-github-2019-11/?r=US&IR=T

What about 2020?!

Top 10 Most Popular Programming Languages

1. Python

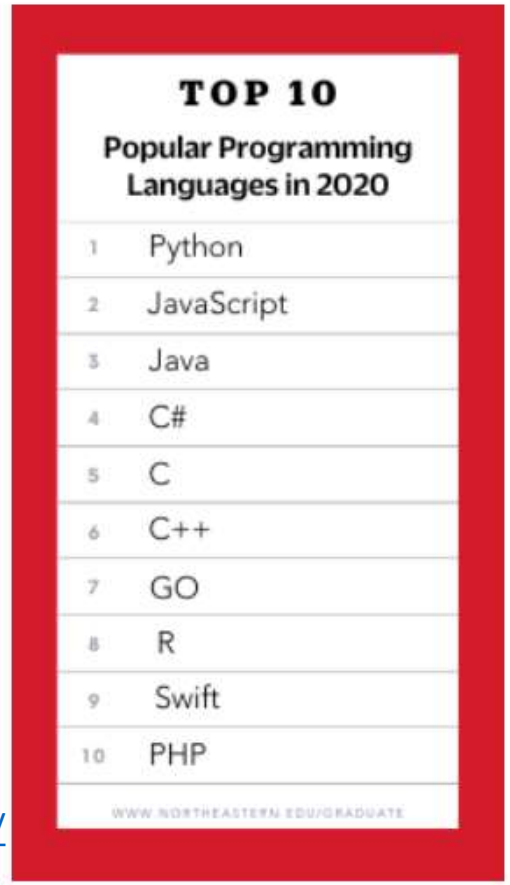
Number of jobs: 19,000

Average annual salary: $120,000

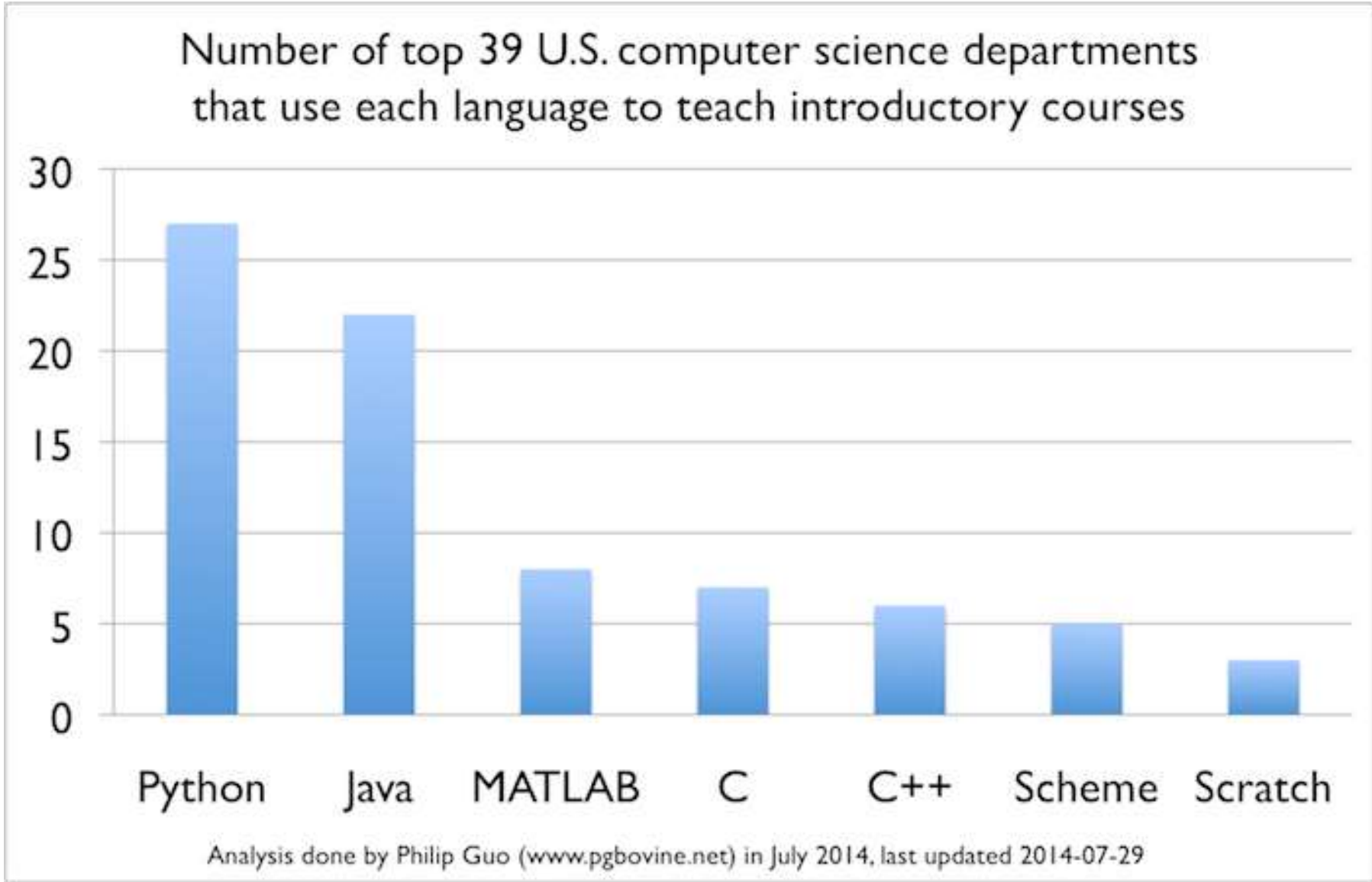
Benefits: Python is widely regarded as a programming language that's easy to learn, due to its simple syntax, a large library of standards and toolkits, and integration with other popular programming languages such as C and C++. In fact, it's the first language that students learn in the Align program, Gorton says. "You can cover a lot of computer science concepts quickly, and it's relatively easy to build on." It is a popular programming language, especially among startups, and therefore Python skills are in high demand.

Drawbacks: Python is not suitable for mobile application development.

Common uses: Python is used in a wide variety of applications, including artificial intelligence, financial services, and data science. Social media sites such as Instagram and Pinterest are also built on Python.



https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/


Number of top 39 U.S. computer science departments that use each language to teach introductory courses
30
25
20
15
10
5
0
Python
Java
MATLAB
C
C++
Scheme
Scratch
Analysis done by Philip Guo (www.pgbovine.net) in July 2014, last updated 2014-07-29

Python Libraries

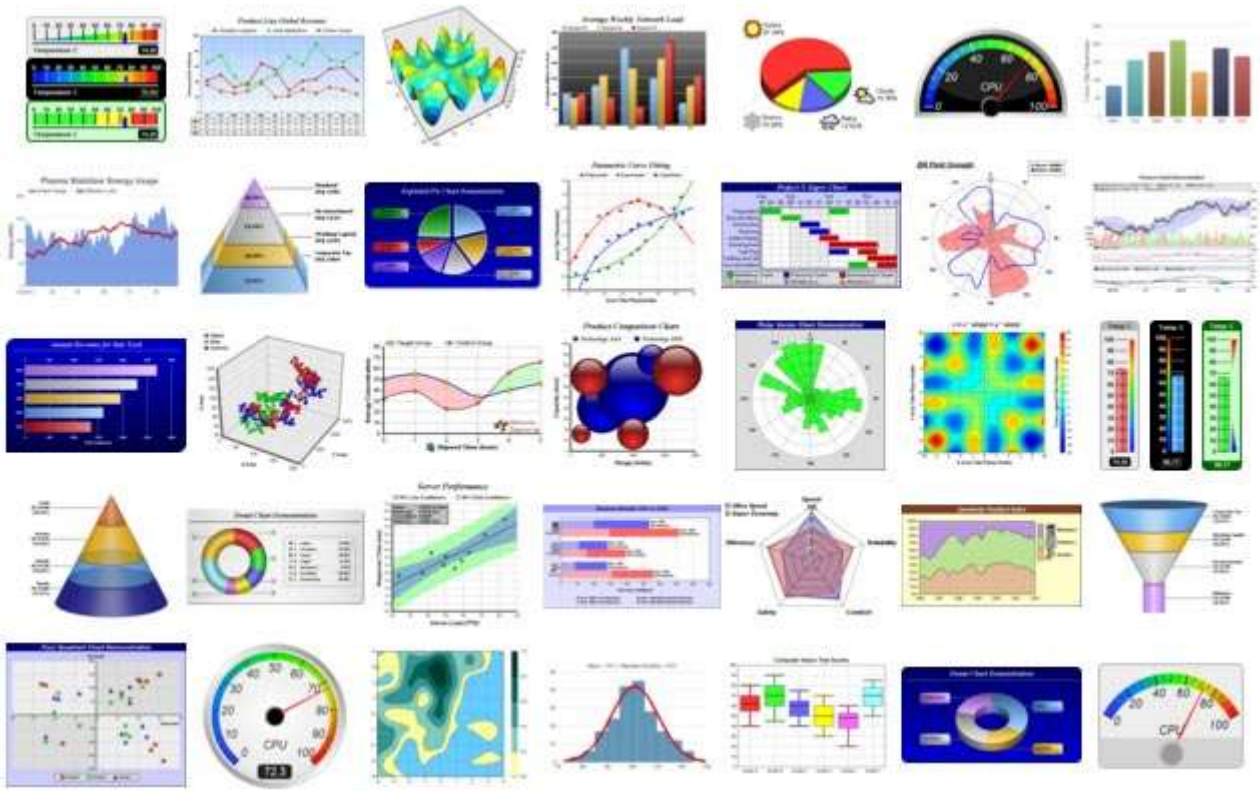


Image Source: http://www.advsofteng.com/

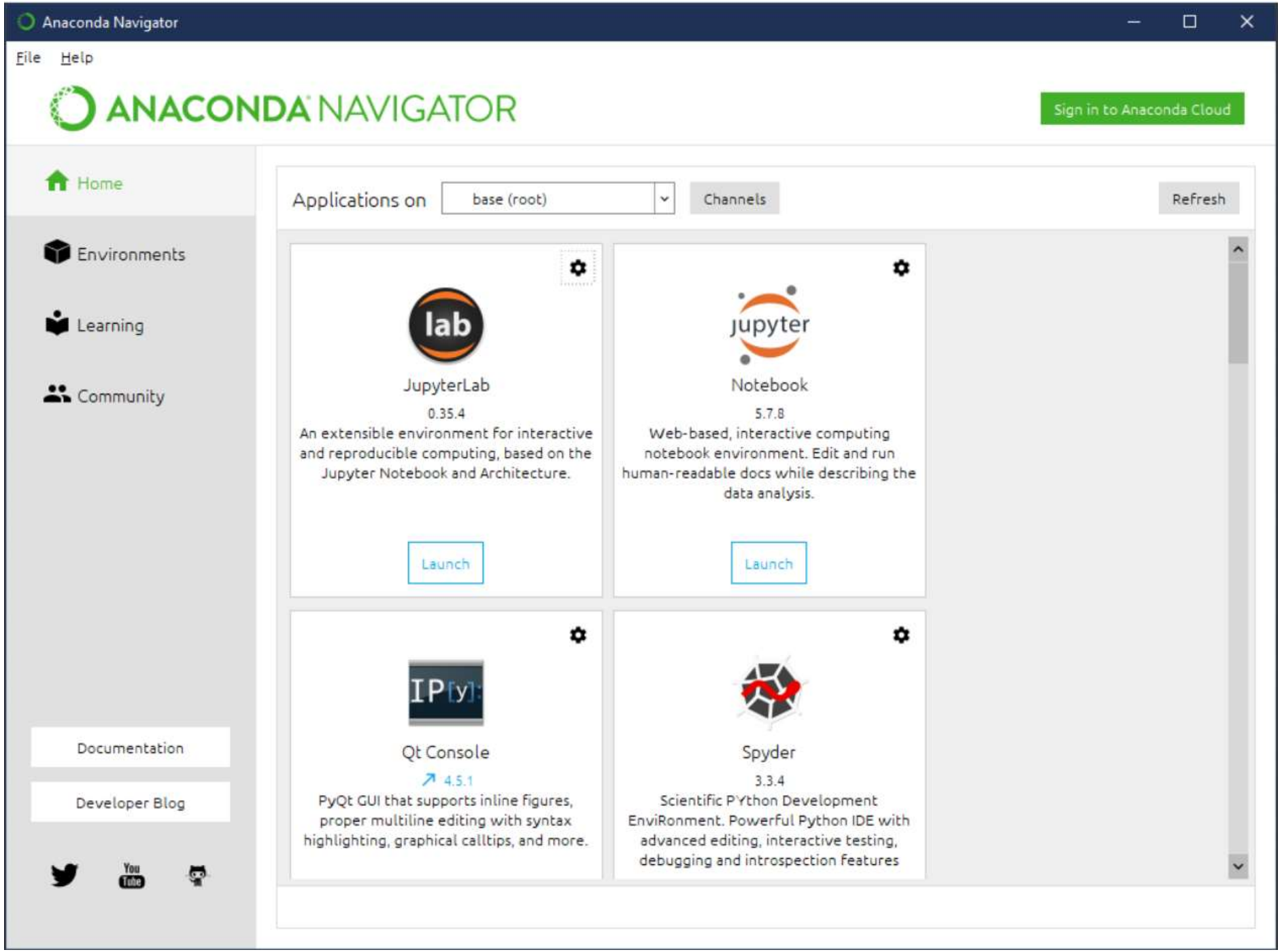
How to install

python™

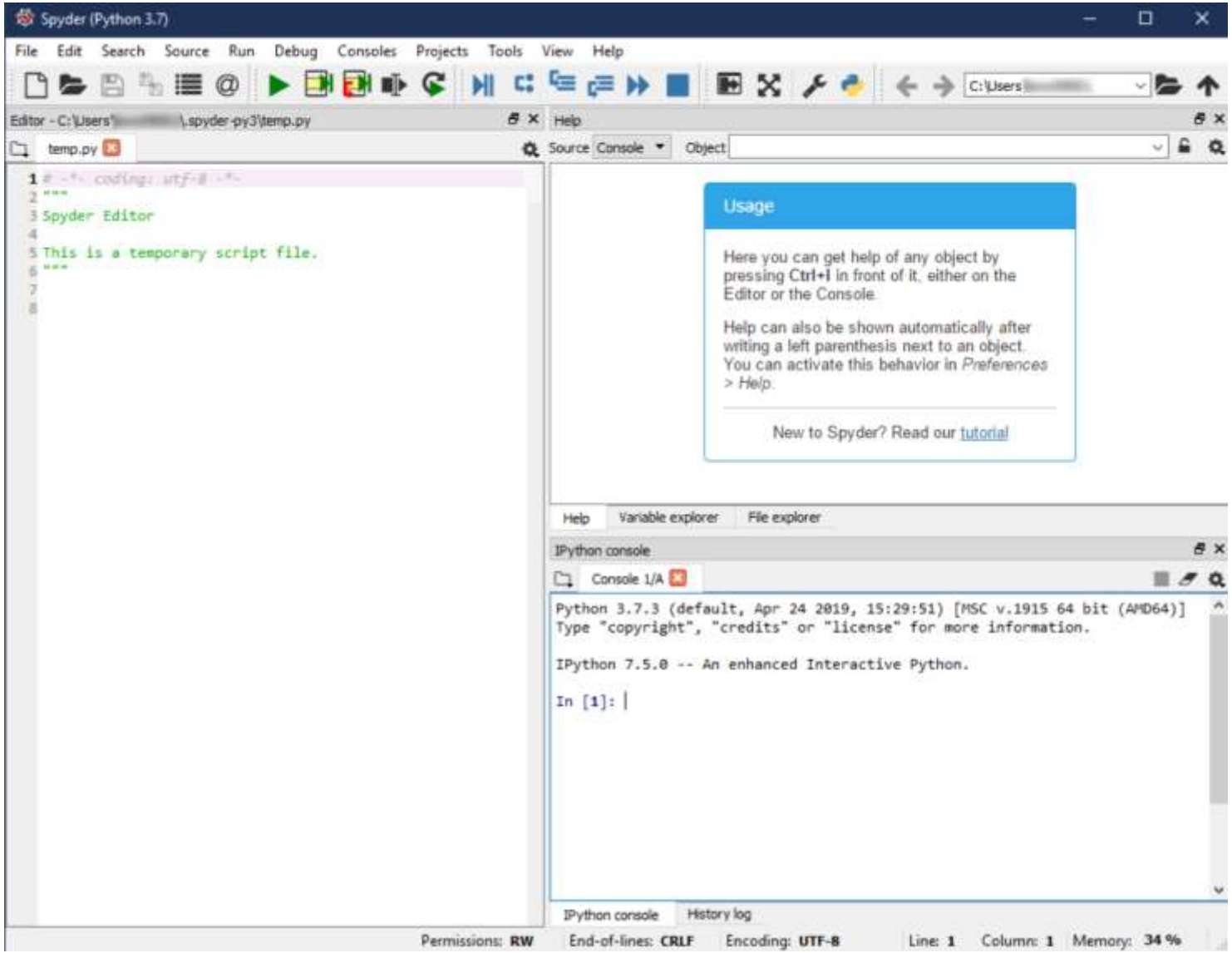
Getting Python via Anaconda

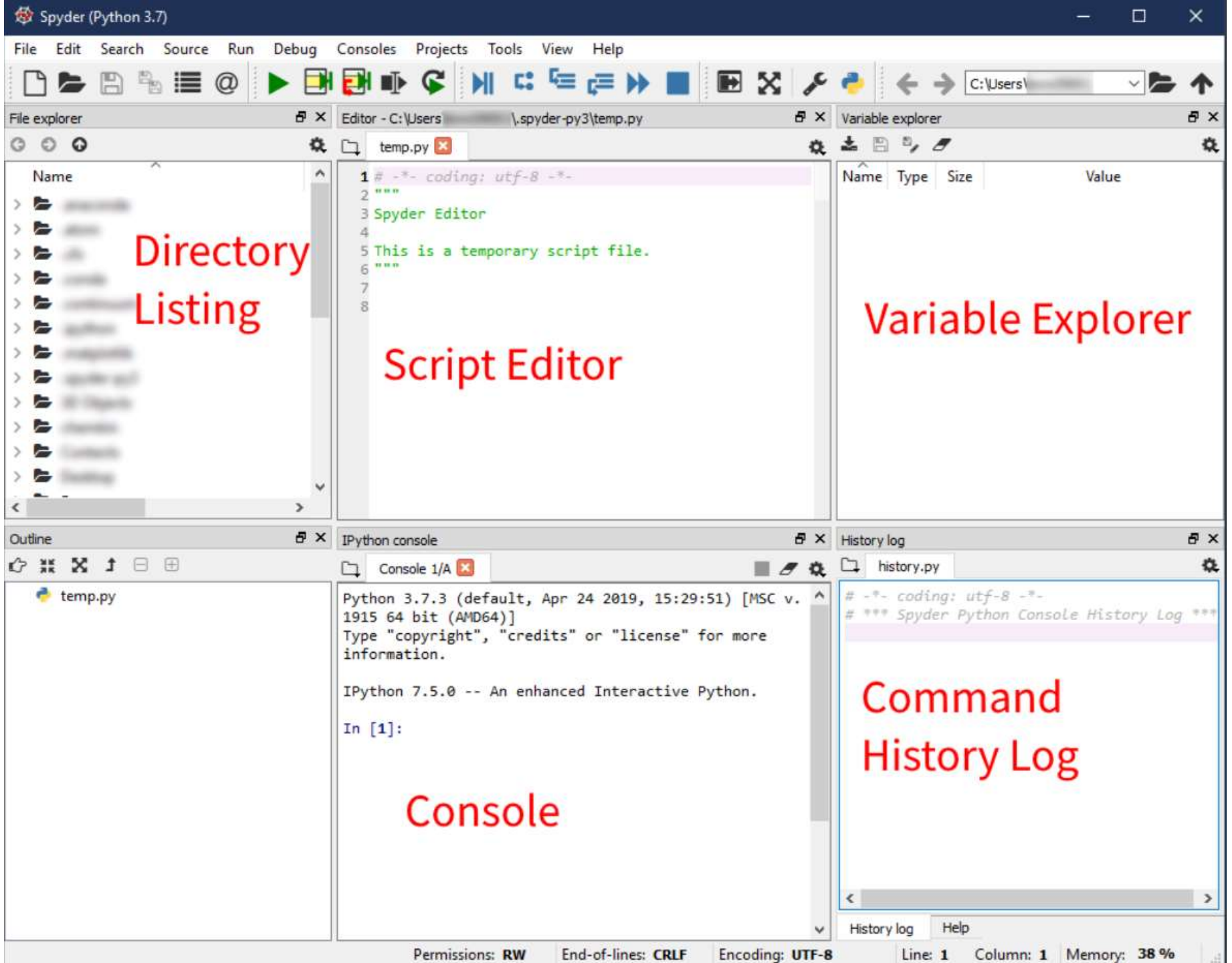
- Python can be downloaded from a number of different sources, called **distributions**.

- Official Python website
  - https://python.org/
- Anaconda distribution
  - Another very popular Python distribution, particularly for math, science, engineering, and data science applications
  - https://anaconda.com/


Anaconda Navigator
File
Help
ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Learning
Community
Documentation
Developer Blog
Applications on
base (root)
Channels
Refresh
JupyterLab
0.35.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
5.7.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Qt Console
4.5.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Spyder
3.3.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Spyder


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\ \.spyder-py3\temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
Help
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Help Variable explorer File explorer
IPython console
Console 1/A
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.5.0 -- An enhanced Interactive Python.
In [1]:
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 1 Memory: 34 %


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
File explorer
Name
Directory Listing
Editor - C:\Users\.spyder-py3\temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
Script Editor
Variable explorer
Name Type Size Value
Variable Explorer
Outline
temp.py
IPython console
Console 1/A
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v. 1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.5.0 -- An enhanced Interactive Python.
In [1]:
Console
History log
history.py
-*- coding: utf-8 -*-
*** Spyder Python Console History Log ***
Command History Log
History log
Help
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 1
Column: 1
Memory: 38 %

دانلود و نصب آناکوندا برای کار با پایتون

- http://blog.class.vision/1396/11/intro-anaconda/

Jupyter Notebook

Logout

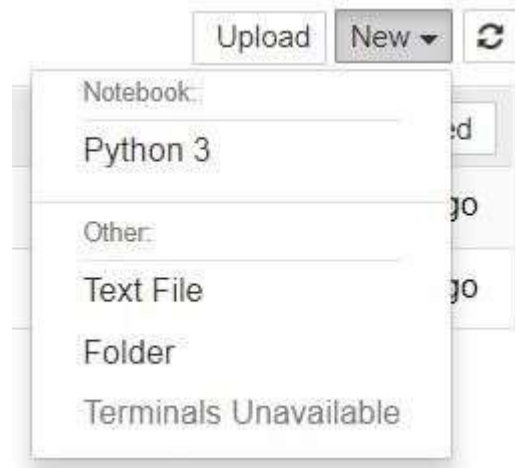
Files | Running | Clusters

Select items to perform actions on them.

Upload | New | 

0 | Name | Last Modified

3D Objects	11 days ago
Contacts	11 days ago
Desktop	11 days ago
Documents	5 days ago
Downloads	2 days ago
Favorites	11 days ago


Upload
New
Notebook:
Python 3
Other:
Text File
Folder
Terminals Unavailable

More…

https://www.dataquest.io/blog/jupyter-notebook-tutorial/

Python Basics

python™

Python Basics

180 / 60

```
Print("Hello Python")
```

Hello Python

14	**int**
13.874	**float**
“Hello World”	**str**

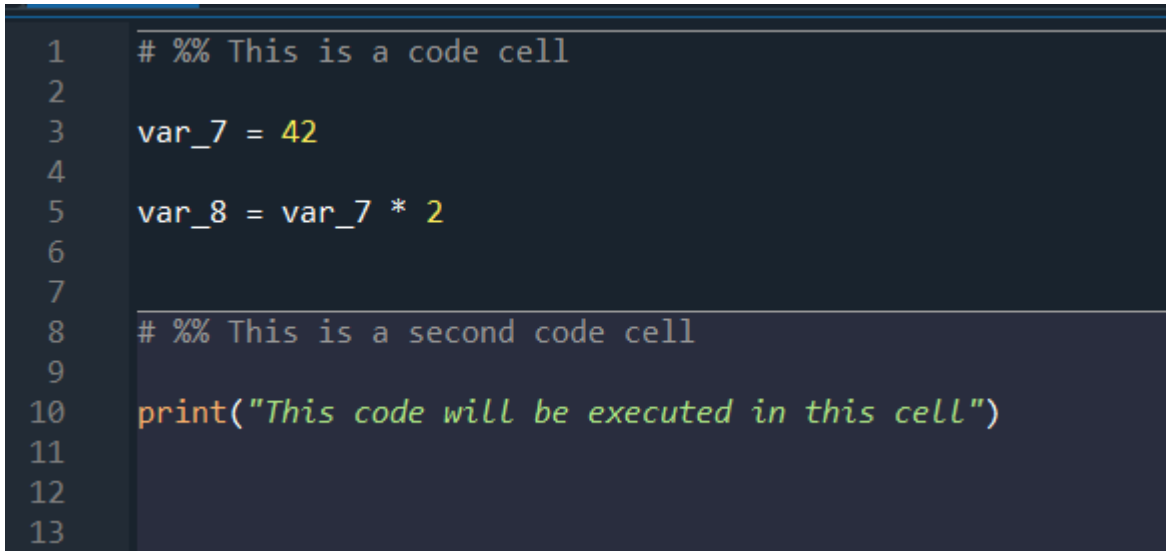
Python Basics

Python

```
In [3]: var_1 = 10
In [4]: var_2 = 20
In [5]: var_3 = var_1 + var_2
In [6]: var_3
Out[6]: 30
```

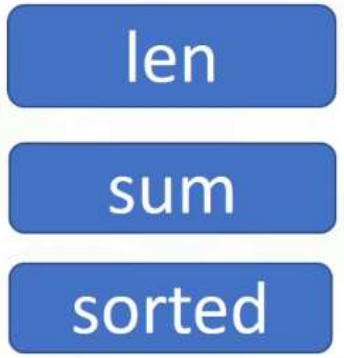
Code cell in spider

```
# %% This is a code cell

var_7 = 42

var_8 = var_7 * 2


# %% This is a second code cell

print("This code will be executed in this cell")
```



Python Programming Fundamental

Python Programming Fundamental




```
def function(a):
    """add 1 to a"""
    b=a+1;

    print(a, " +1 = ",b)
    return b
```

```
if age>18:
else:
elif age==18:
```

Name of object Name of Class

```
RedCircle = Circle(10, "red")
```

```
for i in range(0,5):
while(squares[i]=='green'):
```

Syntax Error

Frint(“my Python Class")

SyntaxError: invalid character in identifier

Semantic Error

```
print(“ny Python Class")
```

```
ny Python Class
```

جنبه‌های زبان:
ساختار دستور (دستور زبان)

- **syntax**
  - **English:** "`cat dog boy`“ → not syntactically valid
    "`cat hugs boy`" → syntactically valid
  - **programming language:** `"hi"5` → not syntactically valid
    `3.2*5` → syntactically valid

جنبه‌های زبان:
معنایی ایستا

- **static semantics** is which syntactically valid strings have meaning
  - **English:** "`I are hungry`" → syntactically valid
    but **static semantic error**
  - **programming language:** `3.2*5` → syntactically valid
    `3+"hi"` → static semantic error

جنبه‌های زبان:
معنایی

- **Semantics** is the meaning associated with a syntactically correct string of symbols with no static semantic errors
- **English:** can have many meanings –
  - "Flying planes can be dangerous"
- **فارسی**
  - علی به دوستش گفت که مقاله‌اش منتشر شده است.
  - عفو لازم نیست اعدامش کنید.
- **programming languages: have only one meaning but** may **not be what programmer intended**

- **syntactic errors**
  - common and easily caught
- **static semantic errors**
  - some languages check for these before running program
  - can cause unpredictable behavior
- no semantic errors but **different meaning than what programmer intended**
  - program crashes, stops running
  - program runs forever
  - program gives an answer but different than expected

یک برنامه پایتون

- a **program** is a sequence of definitions and commands
  - definitions **evaluated**
  - commands **executed** by Python interpreter in a shell
- **commands** (statements) instruct interpreter to do something
- can be typed directly in a **shell** or stored in a **file** that is read into the shell and evaluated

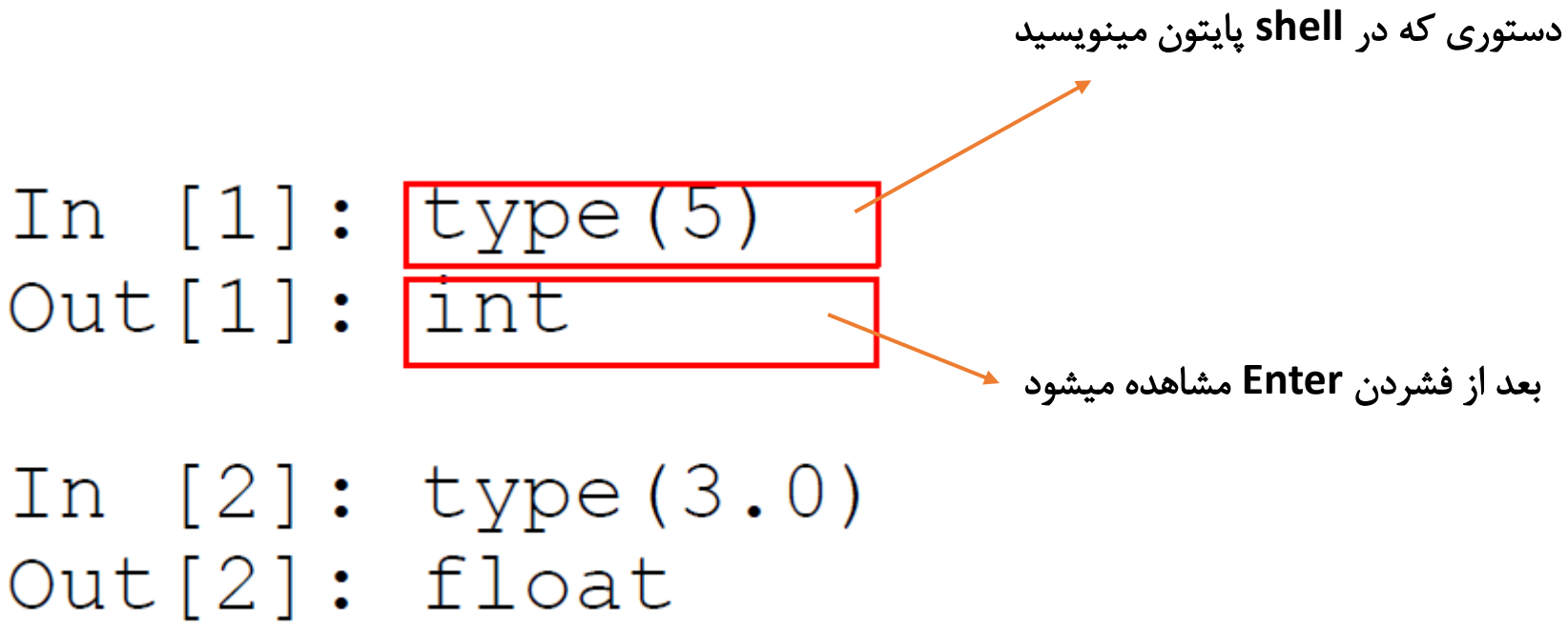
Object (شئ)

- در پایتون برنامه‌ها اشیاء داده ای (**data objects**) را تغییر میدهند.
- اشیاء دارای نوع (**type**) هستند. این نوع تعیین کننده ی کارهایی است که میتوان در برنامه با آن شئ انجام داد.
- اشیاء دو نوع اند:
  - Scalar : به قطعات کوچکتر تقسیم نمی شود
  - non-scalar: دارای ساختار درونی است که میتوان به آن دسترسی پیدا کرد

SCALAR OBJECTS

- **int**
  - ✓ نماینده ی اعداد صحیح، برای مثال `5`
- **float**
  - ✓ نماینده ی اعداد حقیقی، برای مثال `4.26`
- **bool**
  - ✓ نماینده ی مقادیر منطقی شامل `True` و `False` است
- **NoneType**
  - ✓ یک نوع منحصر به فرد که تنها یک مقدار دارد؛ `None`

با استفاده از `type()` میتوان نوع شئ را تشخیص داد.



TYPE CONVERSIONS (CAST)

- can **convert object of one type to another**
- `float(3)` converts integer `3` to float `3.0`
- `int(3.9)` truncates float `3.9` to integer `3`

PRINTING TO CONSOLE

- To show output from code to a user, use `print` command

```
In [11]: 3+2
Out[11]: 5

In [12]: print(3+2)
5
```

no 'Out' because no value returned, just something printed

If - else

- ارزش عددی و منطقی یک متغیر عملیات در پایتون دستور ifو elifو elseدستورات ifتو در تو شرط های ترکیبی

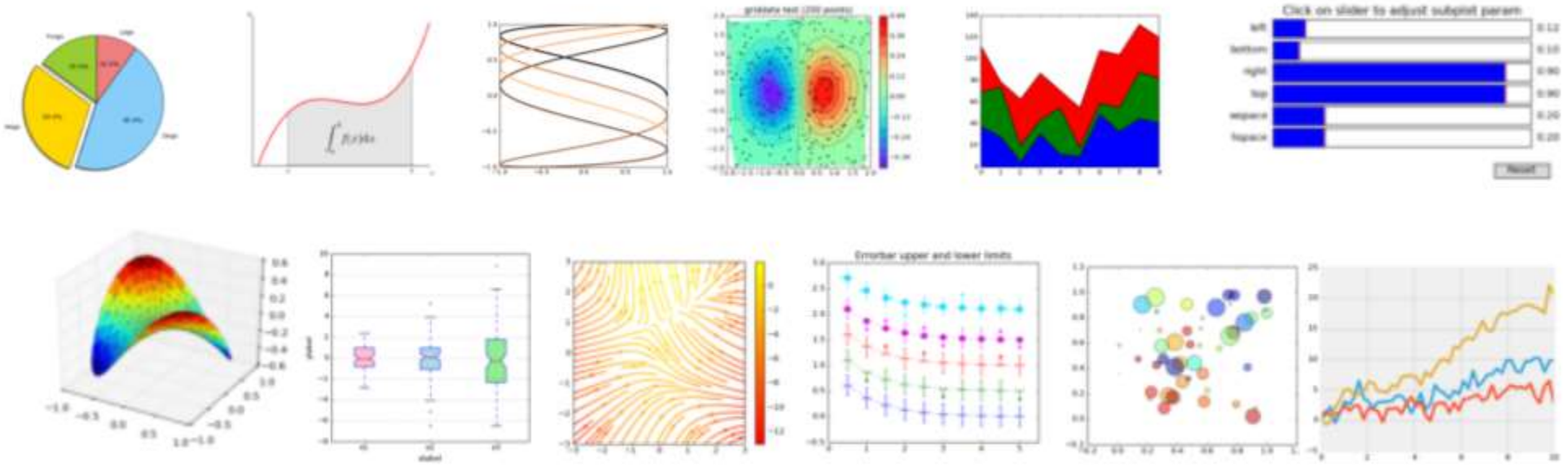
- http://python.akhavanpour.ir/w7s5/session5.pdf
- http://python.akhavanpour.ir/w7s5/code.zip

Loop

- http://python.akhavanpour.ir/w9s6/session6-loop.pdf
- http://python.akhavanpour.ir/w10s7/session7-for-string-break.pdf

plot

- http://python.akhavanpour.ir/w14s9/pyplot.pdf



Python Data Structures

Python Data Structures

```
L = [“Salam”, 12, 30.7]
```

Lists

```
Ratings = (10, 4, 10, 6 ,8)
```

Tuples

```
album_set= set(album_lists)
```

Set

```
{“key1”:3, “key2”:”salam”, “key3”:’b’}
```

Dictionary

List and tuples:

- https://cs231n.github.io/python-numpy-tutorial/#jupyter-and-colab-notebooks
- http://pylab.akhavanpour.ir/
- http://pylab.akhavanpour.ir/slides/tuples_lists.pdf

List

Index

0	Element 1
1	Element 2
2	Element 3
3	Element 4
.....	.....

Element

Dictionary

Key: is a index by label

Key 1	Value 1
Key 1	Value 2
Key 2	Value 3
Key 3	Value 4
.....	.....

Element/Values

Dictionaries

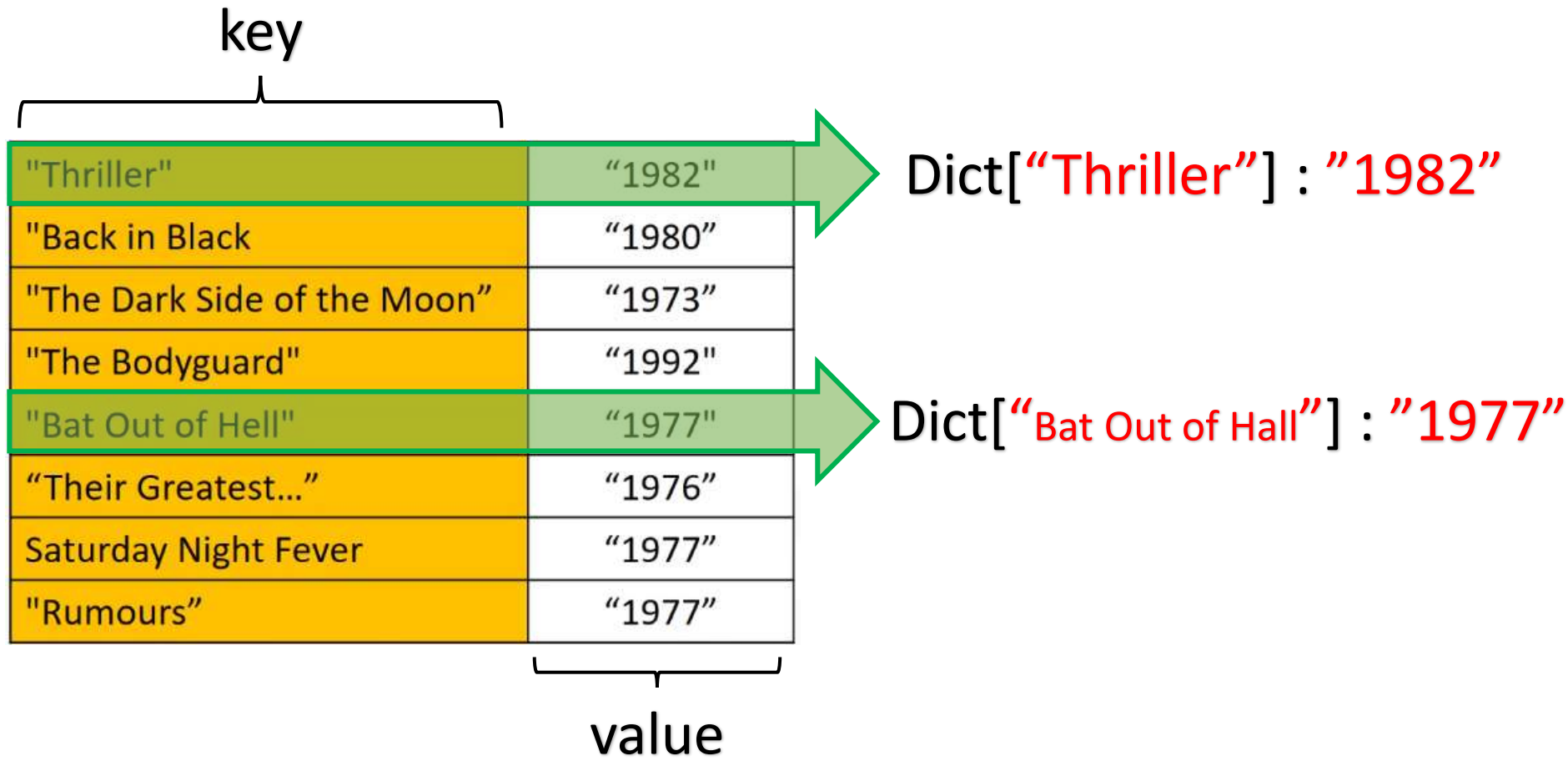
- Dictionaries are denoted with curly Brackets {}
  - **کلیدهای دیکشنری باید immutable و یکتا (unique) باشند**
  - **مقادیر(value‌ها) میتونند immutable یا mutable باشند**
  - **هر زوج key و value با ورگول یا کاما , از هم جدا میشوند.**

{“key1” :1, "key2 ” :"2", 'key3” :[3,3,3], "key4":(4,4,4), ('key5’):5}

"Thriller":	1982,	"Back in Black:	1980,	"The Dark Side of the Moon":	1973,	"The Bodyguard":	1992,

Dict =

"Thriller"	"1982"
"Back in Black	"1980"
"The Dark Side of the Moon"	"1973"
"The Bodyguard"	"1992"
"Bat Out of Hell"	"1977"
"Their Greatest…"	"1976"
Saturday Night Fever	"1977"
"Rumours"	"1977"


key
"Thriller"
“1982"
"Back in Black
“1980”
"The Dark Side of the Moon”
“1973”
"The Bodyguard"
“1992"
"Bat Out of Hell"
“1977"
“Their Greatest…”
“1976”
Saturday Night Fever
“1977”
"Rumours”
“1977”
value
Dict[“Thriller”] : ”1982”
Dict[“Bat Out of Hall”] : ”1977”

افزودن به دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

'Graduation'	“2007”

پاک کردن از دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

del(Dict[“Thriller”])

بررسی وجود کلید در دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

‘The Bodyguard’ in Dict

True

بررسی وجود کلید در دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

‘Jumong’ in Dict

False

لیست کردن کلیدهای دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
“Saturday Night Fever”	“1977”
"Rumors”	“1977”

Dict.keys() =[“Thriller” ، “Back in Black”, “The Dark Side of the Moon”, “The Body guard” ،
“Bat of Hell”, “The Greatest..”, “Saturday Night Fever”, “Rumors”]

لیست کردن Valueهای دیکشنری

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
“Saturday Night Fever”	“1977”
"Rumors”	“1977”

Dict.values() =[“1982”, “1980 ,“1973”, “1992”, “1977”,“1976”, “1977”, “1977” **]**

Dics

- https://cs231n.github.io/python-numpy-tutorial/#jupyter-and-colab-notebooks

- http://colab.research.google.com/github/Alireza-Akhavan/python-labs/blob/master/5-1_Dictionaries_v5.ipynb

Virtual env

- https://cs231n.github.io/setup-instructions/#working-locally-on-your-machine

- https://realpython.com/python-windows-machine-learning-setup/

Numpy: 1D


Beautiful is better than ugly.
Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules.
Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!
python
TM

Creating Arrays

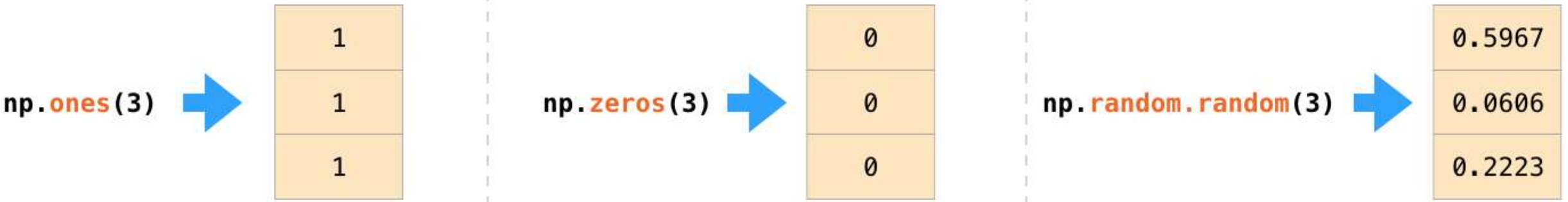
Command

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

NumPy Array

1
2
3

Creating Arrays



Array Arithmetic

```
data = np.array([1,2])
```

data

1
2

```
ones = np.ones(2)
```

ones

1
1

```
data + ones
```

=

data

1
2

+

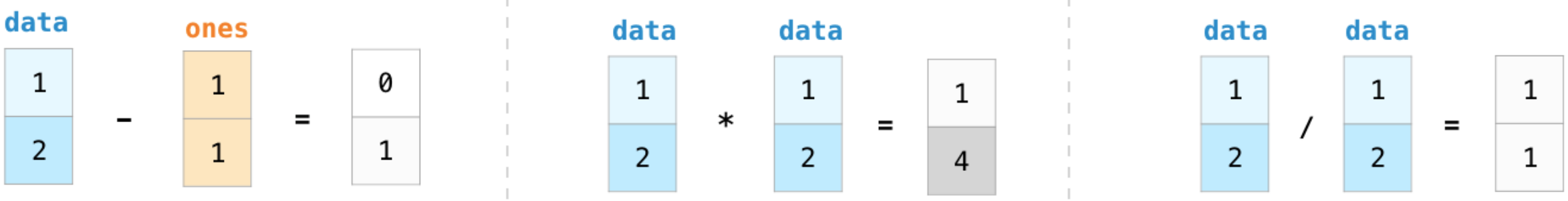
ones

1
1

=

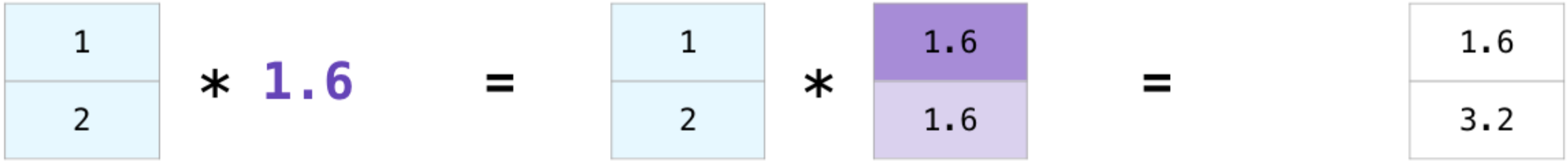
2
3

Array Arithmetic

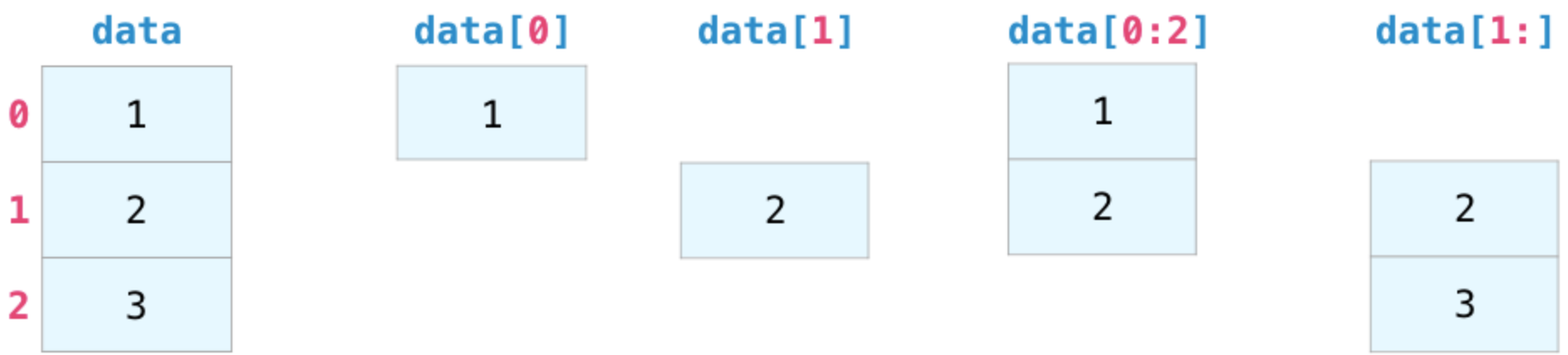


Array Arithmetic

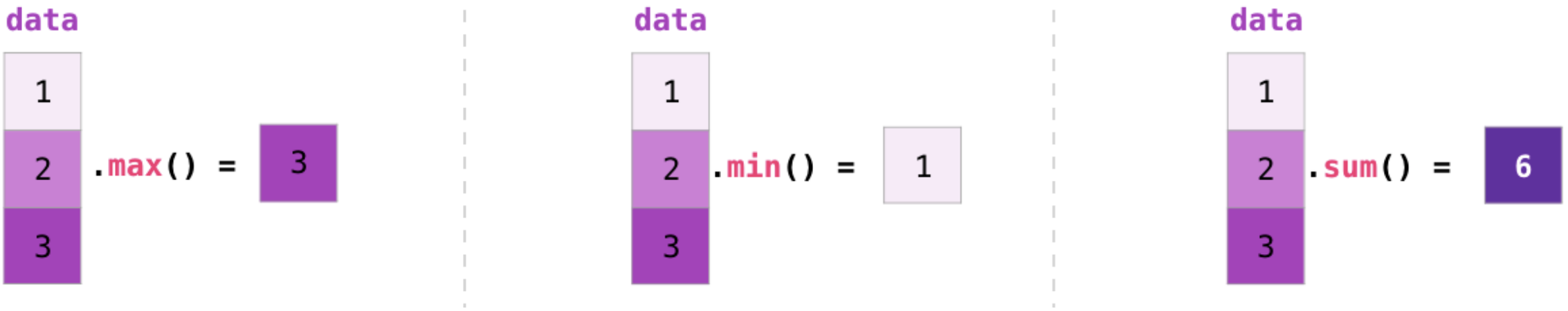
Broadcasting!



Indexing



Aggregation



- min, max, and sum
- mean: to get the average
- prod: to get the result of multiplying all the elements together
- std: to get standard deviation
- plenty of others:
  - https://jakevdp.github.io/PythonDataScienceHandbook/02.04-computation-on-arrays-aggregates.html

Hands on…

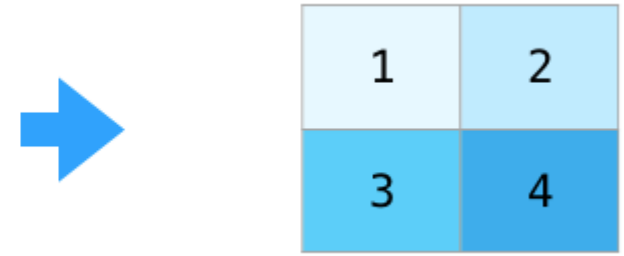
- https://github.com/Alireza-Akhavan/python-labs/blob/master/6-1_Numpy_1D.ipynb
- http://colab.research.google.com/github/Alireza-Akhavan/python-labs/blob/master/6-1_Numpy_1D.ipynb

Numpy: 2D

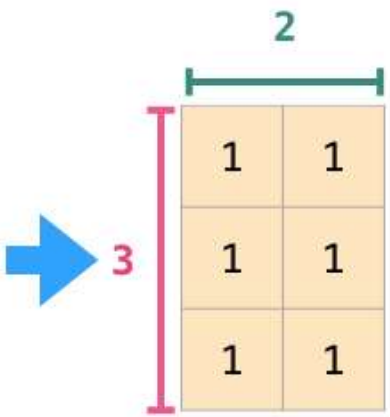

Beautiful is better than ugly.
Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules.
Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!
python
TM

Creating Matrices

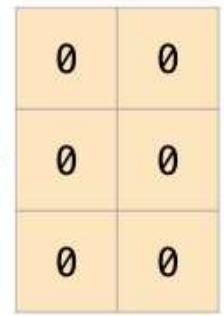
```
np.array([[1,2],[3,4]])
```



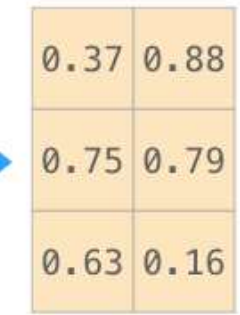

```
np.ones((3,2))
```




```
np.zeros((3,2))
```

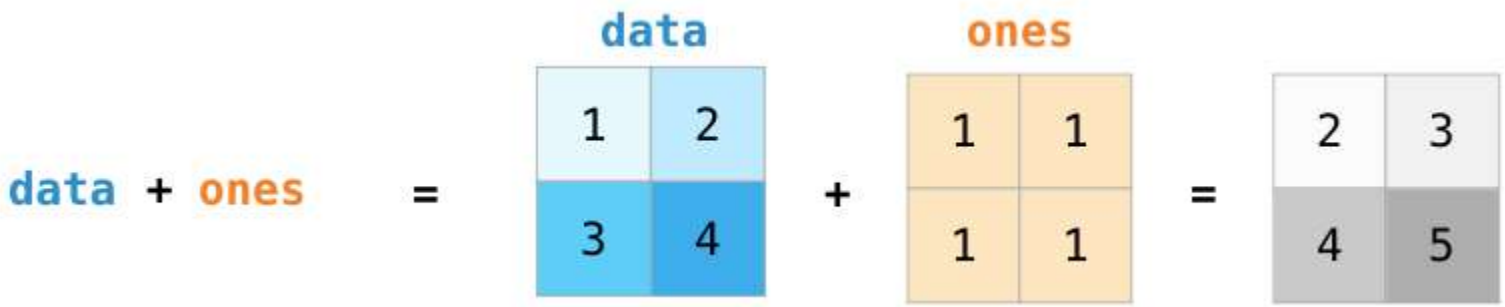



```
np.random.random((3,2))
```



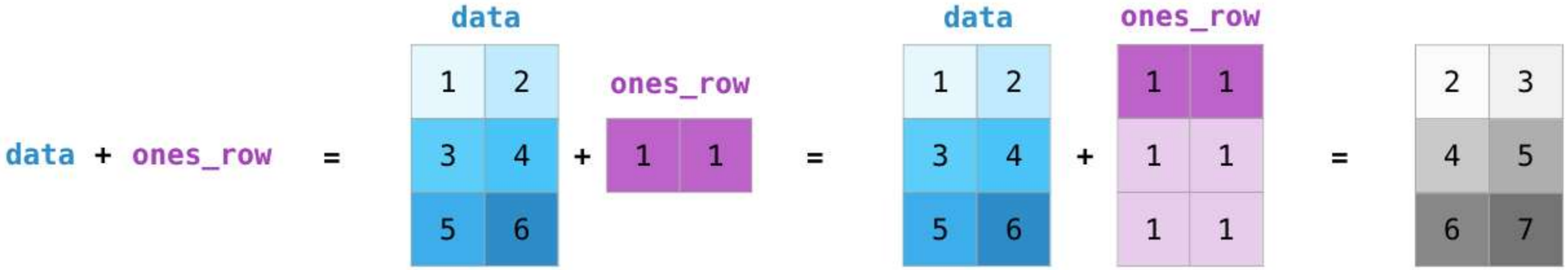
Matrix Arithmetic

- We can add and multiply matrices using arithmetic operators (+-*/) if the two matrices are the same size. NumPy handles those as position-wise operations:



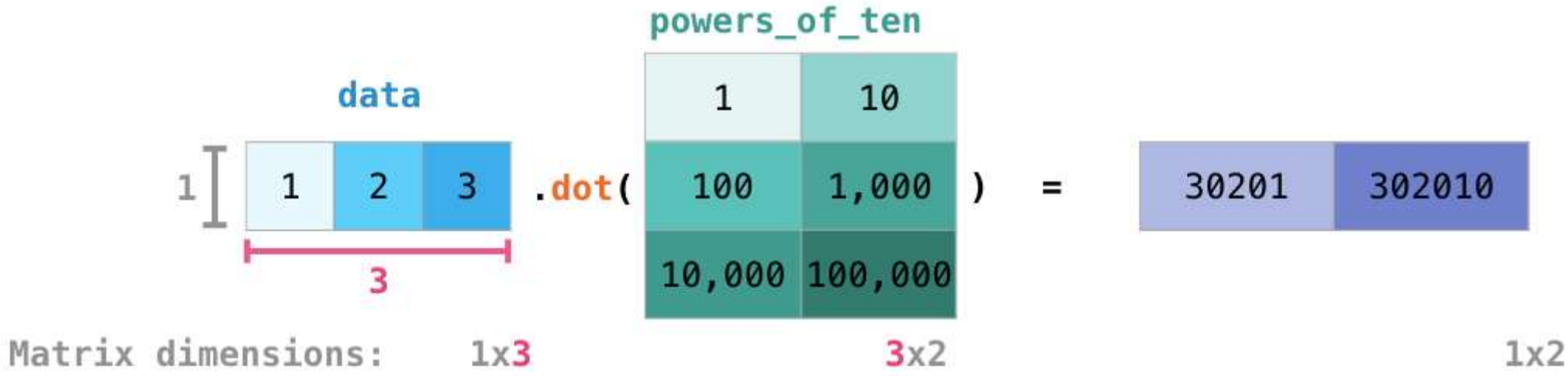
Matrix Arithmetic

- We can get away with doing these arithmetic operations on matrices of different size only if the different dimension is one (e.g. the matrix has only one column or one row), in which case NumPy uses its broadcast rules for that operation:

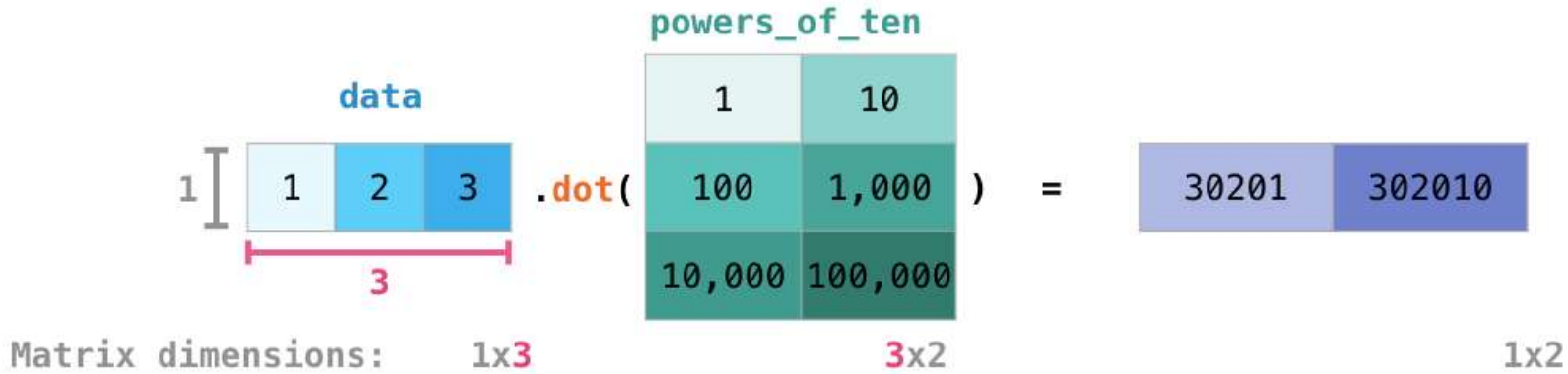


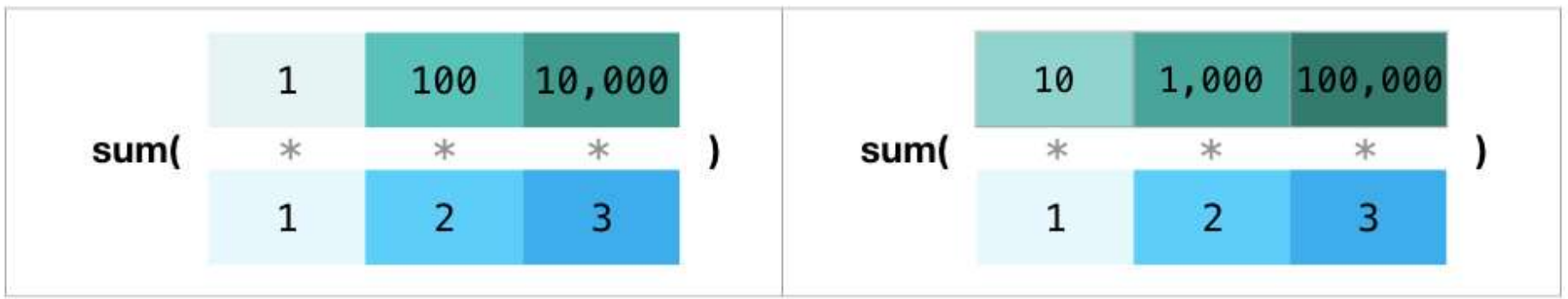
Dot Product

- A key distinction to make with arithmetic is the case of matrix multiplication using the **dot product**. **NumPy gives every matrix a dot()** method we can use to carry-out dot product operations with other matrices:



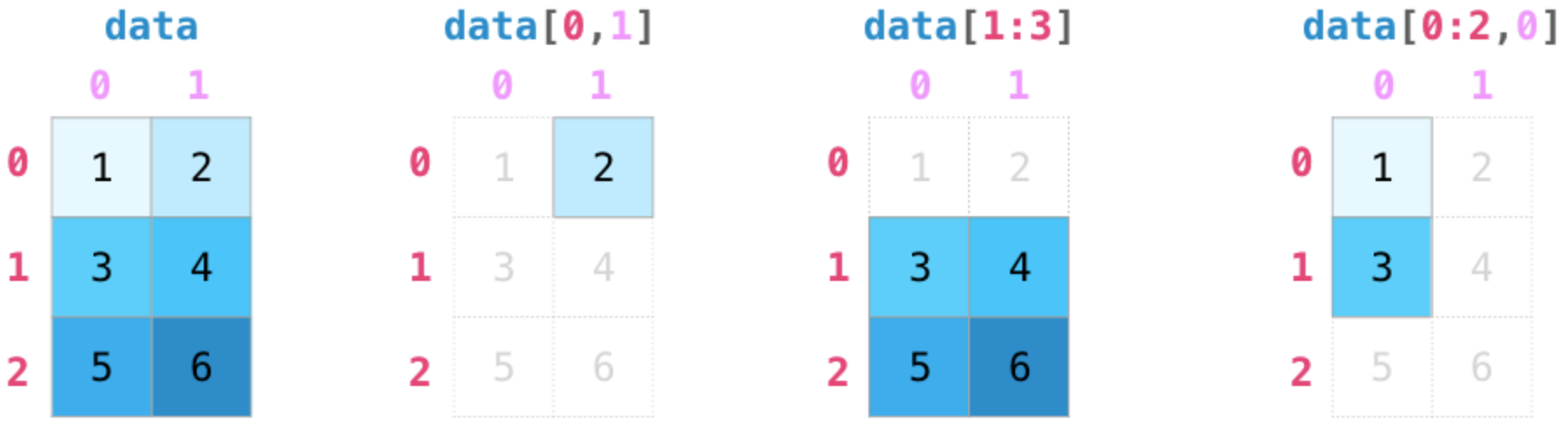
Dot Product



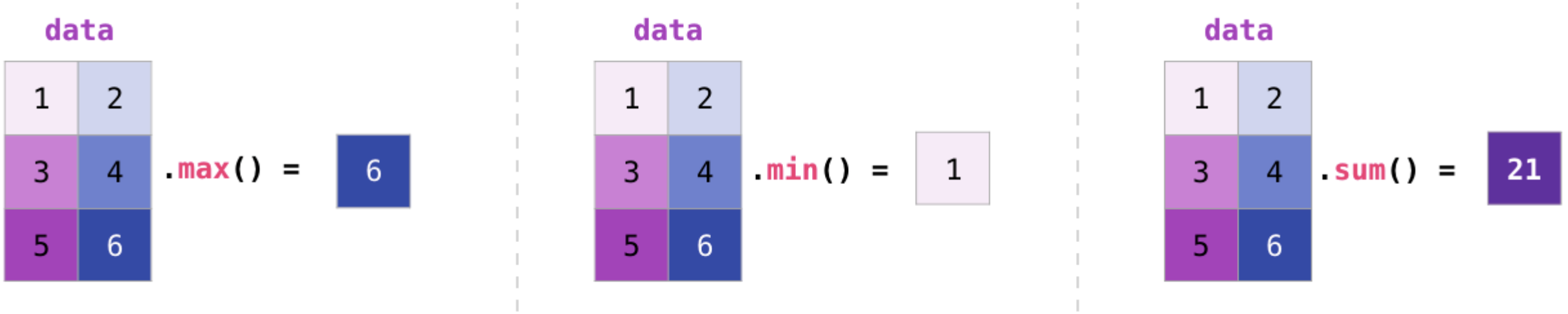


1*1 + 2*100 + 3*10,000	1*10 + 2*1,000 + 3*100,000	=	30201	302010

Matrix Indexing

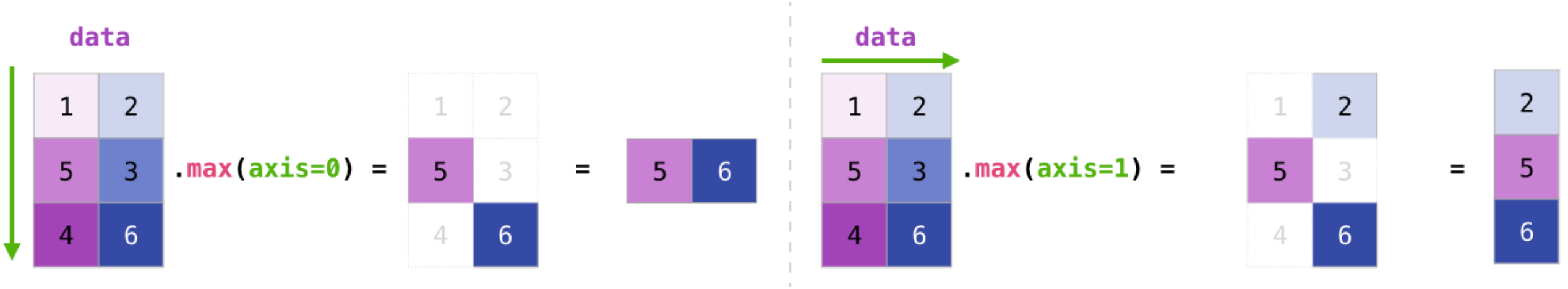


Matrix Aggregation



Matrix Aggregation

- Not only can we aggregate all the values in a matrix, but we can also aggregate across the rows or columns by using the axis parameter:



Transposing and Reshaping

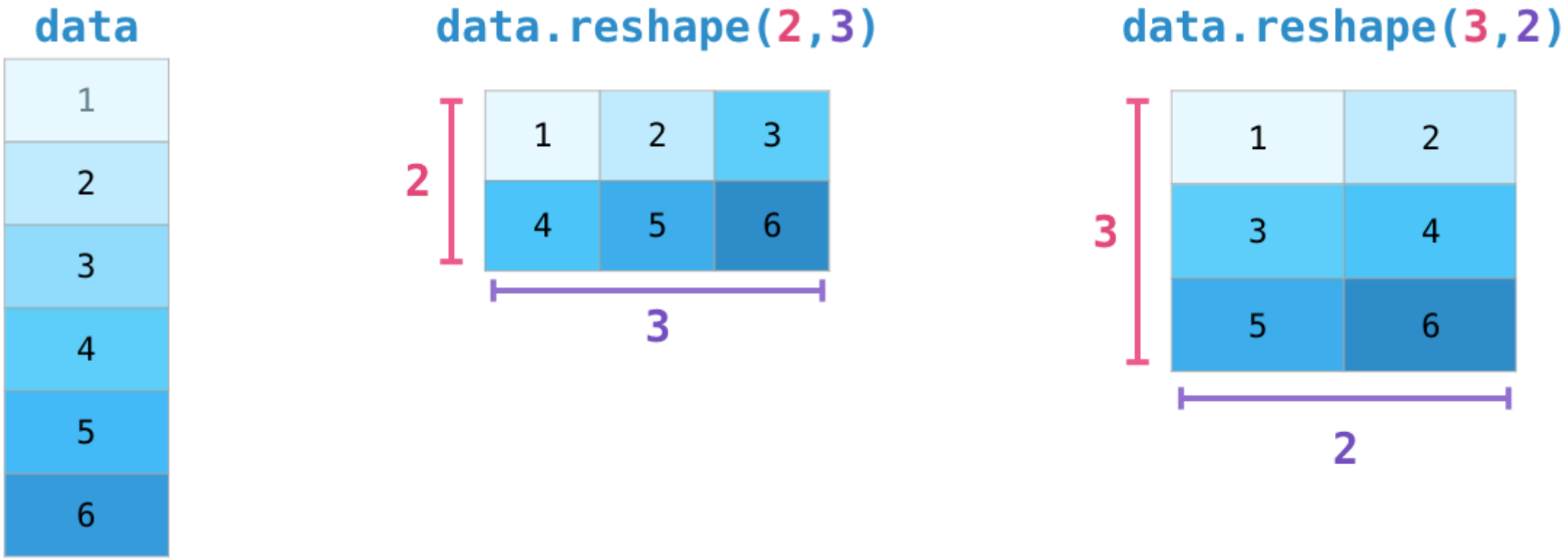
`data`

1	2
3	4
5	6

`data.T`

1	3	5
2	4	6

Transposing and Reshaping



Hands on …

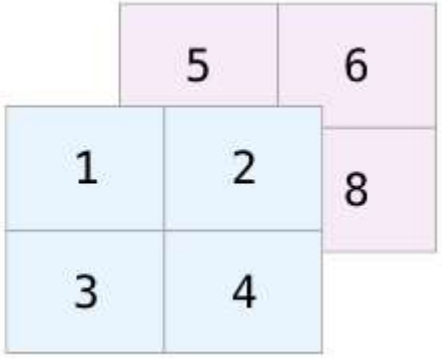
- https://github.com/Alireza-Akhavan/python-labs/blob/master/6-2_Numpy_2D.ipynb
- https://colab.research.google.com/github/Alireza-Akhavan/python-labs/blob/master/6-2_Numpy_2D.ipynb

Numpy: **Yet More Dimensions**

Yet More Dimensions

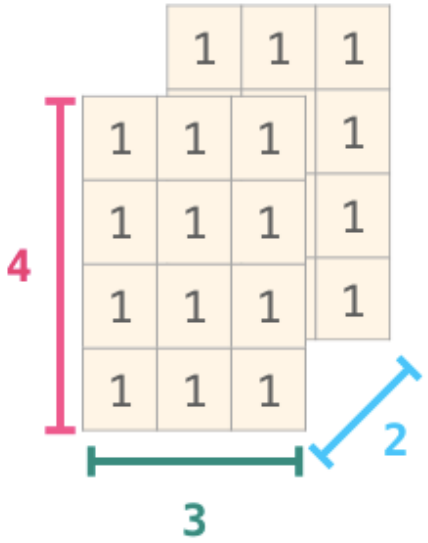
- NumPy can do everything we've mentioned in any number of dimensions.
- Its central data structure is called ndarray (N-Dimensional Array) for a reason.

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

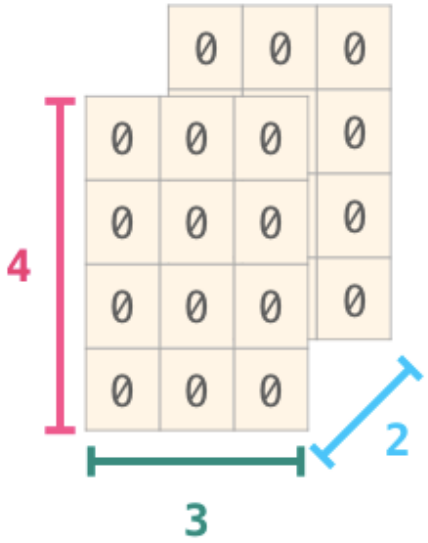


Yet More Dimensions

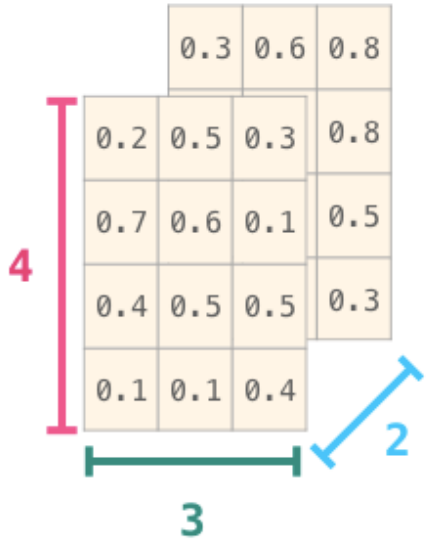
`np.ones((4,3,2))`



`np.zeros((4,3,2))`



`np.random.random((4,3,2))`



Yet More Dimensions

np.ones((4,3,2))

```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

Numpy: **Practical Usage**

python™

Formulas

$$MeanSquareError = \frac{1}{n} \sum_{i=1}^{n} (Y\_prediction_i - Y_i)^2$$

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

Formulas

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

```
                                          predictions   labels
                                              1            1
error = (1/3) * np.sum(np.square(             1      -     2     ))
                                              1            3
```

Formulas

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

predictions labels

```
error = (1/3) * np.sum(np.square( | 1 | - | 1 | ))
                                  | 1 |   | 2 |
                                  | 1 |   | 3 |
```

```
error = (1/3) * np.sum(np.square( | 0 | ))
                                  | -1 |
                                  | -2 |
```

Formulas

```
error = (1/3) * np.sum(np.square(
```

0
-1
-2

```
))
```

```
error = (1/3) * np.sum(
```

0
1
4

```
)
```

```
error = (1/3) * 5
```

Hands on …

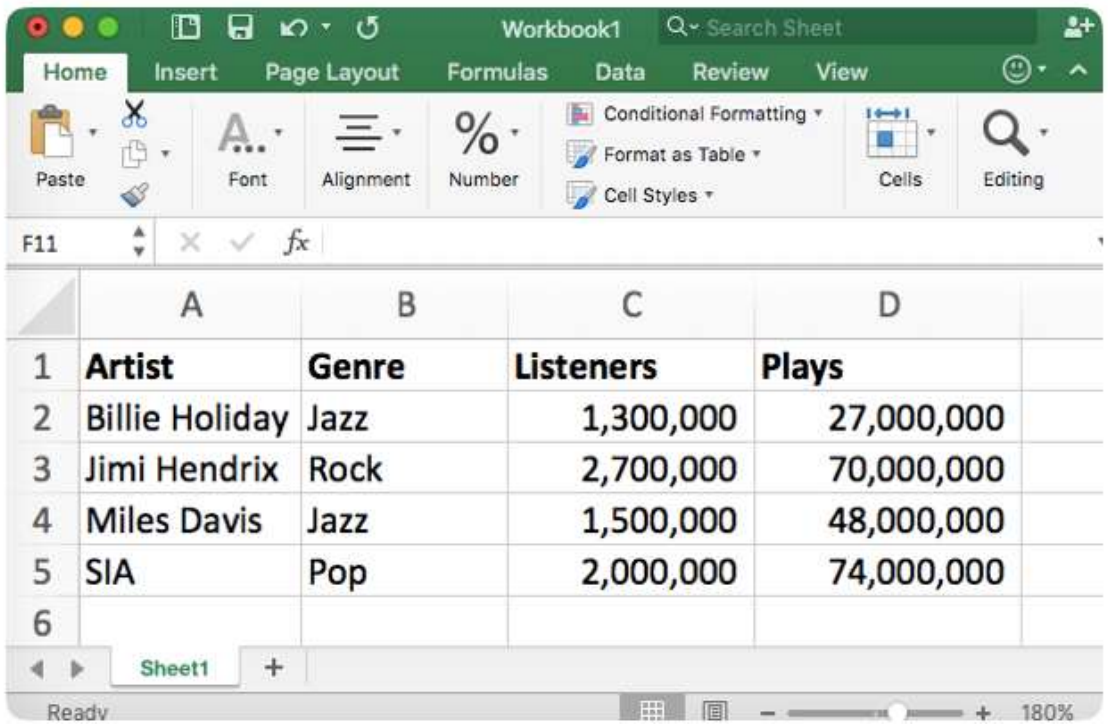
- https://github.com/Alireza-Akhavan/python-labs/blob/master/6-3_Numpy_review.ipynb
- https://colab.research.google.com/github/Alireza-Akhavan/python-labs/blob/master/6-3_Numpy_review.ipynb

Pandas


Beautiful is better than ugly.
Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules.
Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!
python
TM

Tables and Spreadsheets

music.csv



	A	B	C	D
1	**Artist**	**Genre**	**Listeners**	**Plays**
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

```
pandas.read_csv('music.csv')
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

pandas

```
df = pandas.read_csv('music.csv')
```

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Selection

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df['Artists']
```

	Artist
0	Billie Holiday
1	Jimi Hendrix
2	Miles Davis
3	SIA

Selection

`df`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

`df[1:3]`

	Artist	Genre	Listeners	Plays
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000

Selection

- We can select any slice of the table using a both column label and row numbers using loc:

```
df
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df.loc[1:3,['Artist']]
```

	Artist
1	Jimi Hendrix
2	Miles Davis
3	SIA

Selection

- We can select any slice of the table using a both column label and row numbers using loc:

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

inclusive

```
df.loc[1:3,['Artist']]
```

	Artist
1	Jimi Hendrix
2	Miles Davis
3	SIA

Filtering

- e can easily filter rows using the values of a specific row.

```
df
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df[df['Genre'] == 'Jazz']
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000

Filtering

```
df
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df[df['Listeners'] > 1800000]
```

	Artist	Genre	Listeners	Plays
1	Jimi Hendrix	Rock	2,700,000	70,000,000
3	SIA	Pop	2,000,000	74,000,000

Dealing with Missing Values

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Dealing with Missing Values

`df`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

`df.dropna()`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Dealing with Missing Values

`df`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

`df.dropna()`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Another way would be to fill-in the missing value using fillna() (with 0, for example).
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Grouping

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df.groupby('Genre').sum()
```

	Listeners	Plays
Genre		
Jazz	2,800,000	75,000,000
Pop	2,000,000	74,000,000
Rock	2,700,000	70,000,000

Grouping

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df.groupby('Genre').sum()
```

	Listeners	Plays
Genre		
Jazz	2,800,000	75,000,000
Pop	2,000,000	74,000,000
Rock	2,700,000	70,000,000

In addition to sum(), pandas provides multiple aggregation functions including mean() to compute the average value, min(), max(), and multiple other functions. More on **groupyby()** in the Group By User Guide.

Creating New Columns from Existing Columns

```
df
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

```
df['Avg Plays'] = df['Plays']/df['Listeners']
```

	Artist	Genre	Listeners	Plays	Avg Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000	20
1	Jimi Hendrix	Rock	2,700,000	70,000,000	25
2	Miles Davis	Jazz	1,500,000	48,000,000	32
3	SIA	Pop	2,000,000	74,000,000	37

Get Hands On!

https://nbviewer.jupyter.org/github/Alireza-Akhavan/python-labs/blob/master/7-Pandas_Intro.ipynb
https://colab.research.google.com/github/Alireza-Akhavan/python-labs/blob/master/7-Pandas_Intro.ipynb

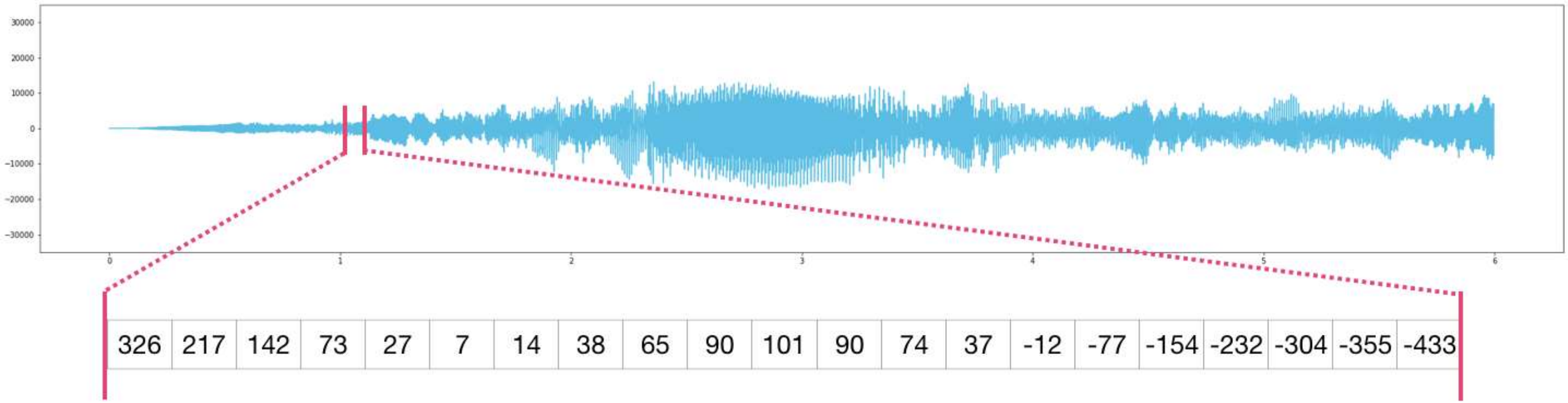
https://nbviewer.jupyter.org/github/Alireza-Akhavan/python-labs/blob/master/7-1-Pandas_review.ipynb

Audio and Timeseries

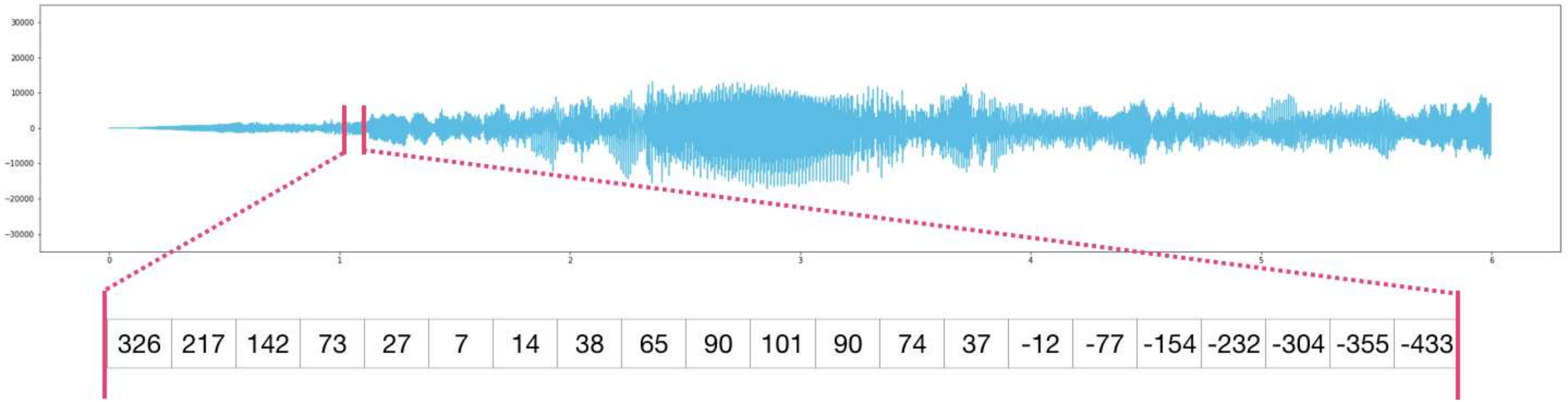
Audio and Timeseries

- An audio file is a one-dimensional array of samples.
- Each sample is a number representing a tiny chunk of the audio signal.
- CD-quality audio may have 44,100 samples per second and each sample is an integer between -32767 and 32768.
- Meaning if you have a ten-seconds WAVE file of CD-quality, you can load it in a NumPy array with length 10 * 44,100 = 441,000 samples.
- Want to extract the first second of audio?
  - simply load the file into a NumPy array that we'll call audio, and get audio[:44100].

Audio and Timeseries



Audio and Timeseries



✓ **The same goes for time-series data (for example, the price of a stock over time).**

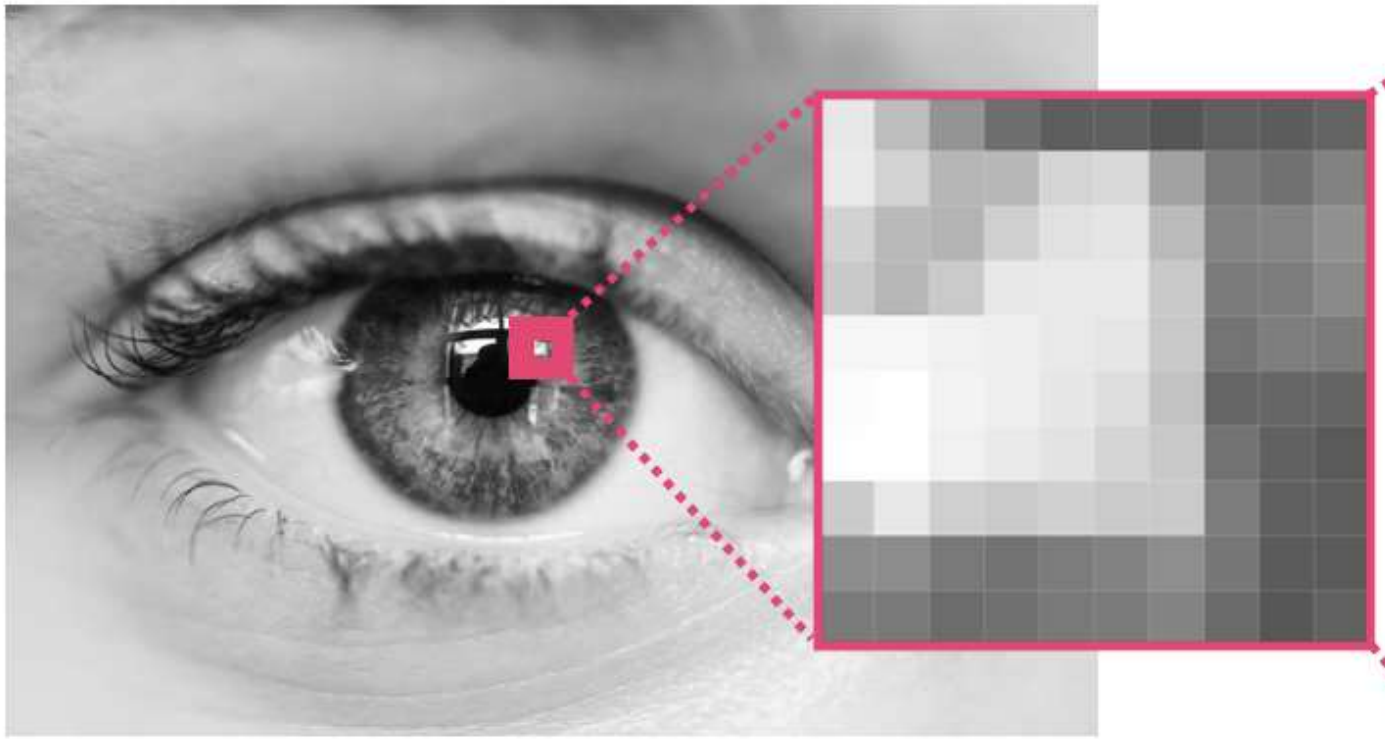
Images


Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!
python
TM

Image

- An image is a matrix of pixels of size (height x width).
  - If the image is black and white (a.k.a. grayscale), each pixel can be represented by a single number (commonly between 0 (black) and 255 (white)).
  - Want to crop the top left 10 x 10 pixel part of the image?
    - tell NumPy to get you image[:10,:10].

Image



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

Image

If the image is colored, then each pixel is represented by three numbers - a value for each of red, green, and blue. In that case we need a 3rd dimension (because each cell can only contain one number). So a colored image is represented by an ndarray of dimensions: (height x width x 3).

	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

Python and module versions

In [2]:
```
%load_ext version_information

%version_information numpy, scipy, matplotlib, sympy, version_information
```

Out[2]:

Software	Version
Python	2.7.10 64bit [GCC 4.2.1 (Apple Inc. build 5577)]
IPython	3.2.1
OS	Darwin 14.1.0 x86_64 i386 64bit
numpy	1.9.2
scipy	0.16.0
matplotlib	1.4.3
sympy	0.7.6
version_information	1.0.3
	Sat Aug 15 10:38:48 2015 JST

Some sources:

- https://realpython.com/matlab-vs-python/#matlab-vs-python-comparing-features-and-philosophy
- https://jalammar.github.io/visual-numpy/