2nd ASTERICS-OBELICS International School

4-8 June 2018, Annecy, France.

H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).

PYTHON LIBRARIES

Tamás Gál
tamas.gal@fau.de

https://github.com/tamasgal

@tamasgal

OVERVIEW

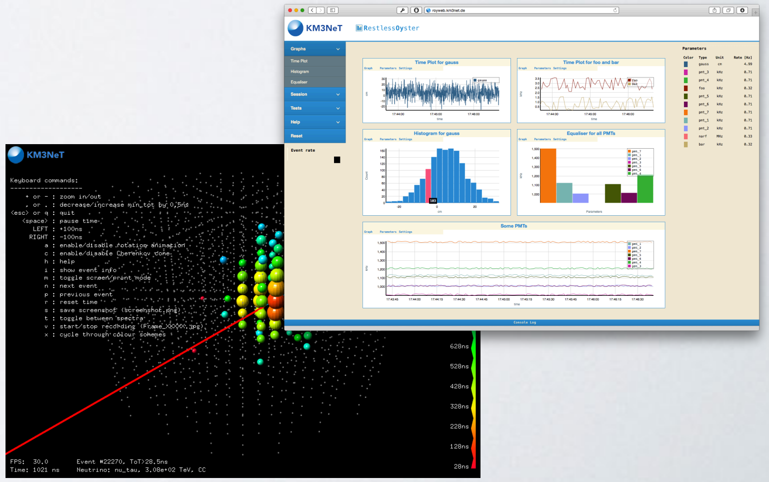
- Who is this clown?
- Python Introduction
- Basic Python Internals
- Libraries and Tools for Scientific Computing
    - NumPy
    - Numba
    - NumExpr
    - SciPy
    - AstroPy
    - Pandas
    - SymPy
    - Matplotlib
    - Jupyter
    - IPython

Make it faster!

Tools for scientists!

WHO IS THIS CLOWN?

- Tamás Gál, born 1985 in Debrecen (Hungary)
- PhD candidate in astro particle physics at Erlangen Centre for Astroparticle Physics (ECAP) working on the KM3NeT project
- Programming background:
  - Coding enthusiast since ~1993
  - First real application written in Amiga Basic (toilet manager, tons of GOTOs ;)
  - Python, Julia, JavaScript and C/C++/Obj-C for **work**
  - Haskell for **fun**
  - Earlier also Java, Perl, PHP, Delphi, MATLAB, whatsoever…
  - I also like playing around with integrated circuits and Arduino
- Some related projects:
  KM3Pipe (analysis framework in the KM3NeT experiment),
  RainbowAlga (interactive 3D neutrino event display),
  ROyWeb (interactive realtime visualisation/graphing)

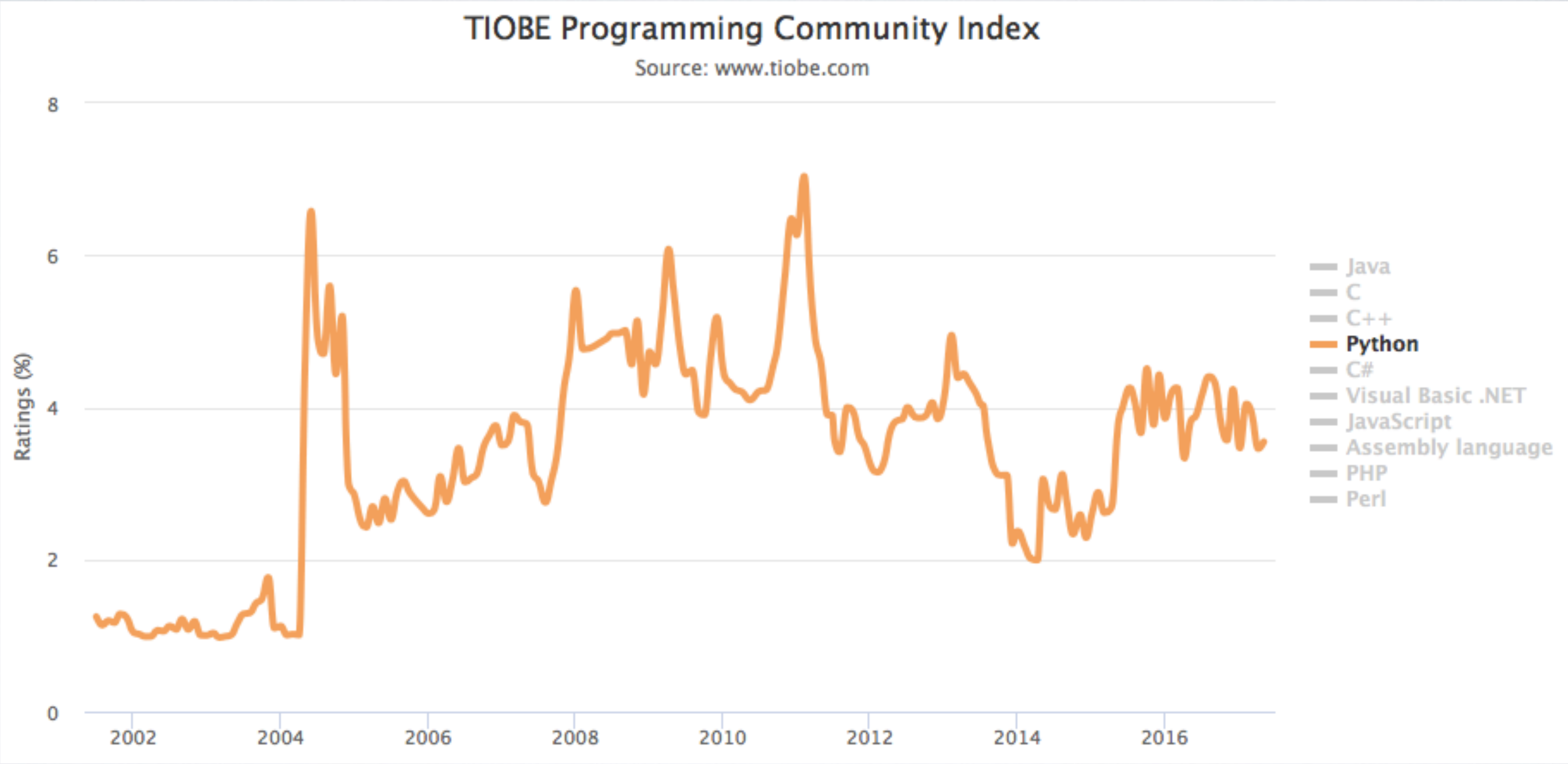


PYTHON

BRIEF HISTORY OF PYTHON

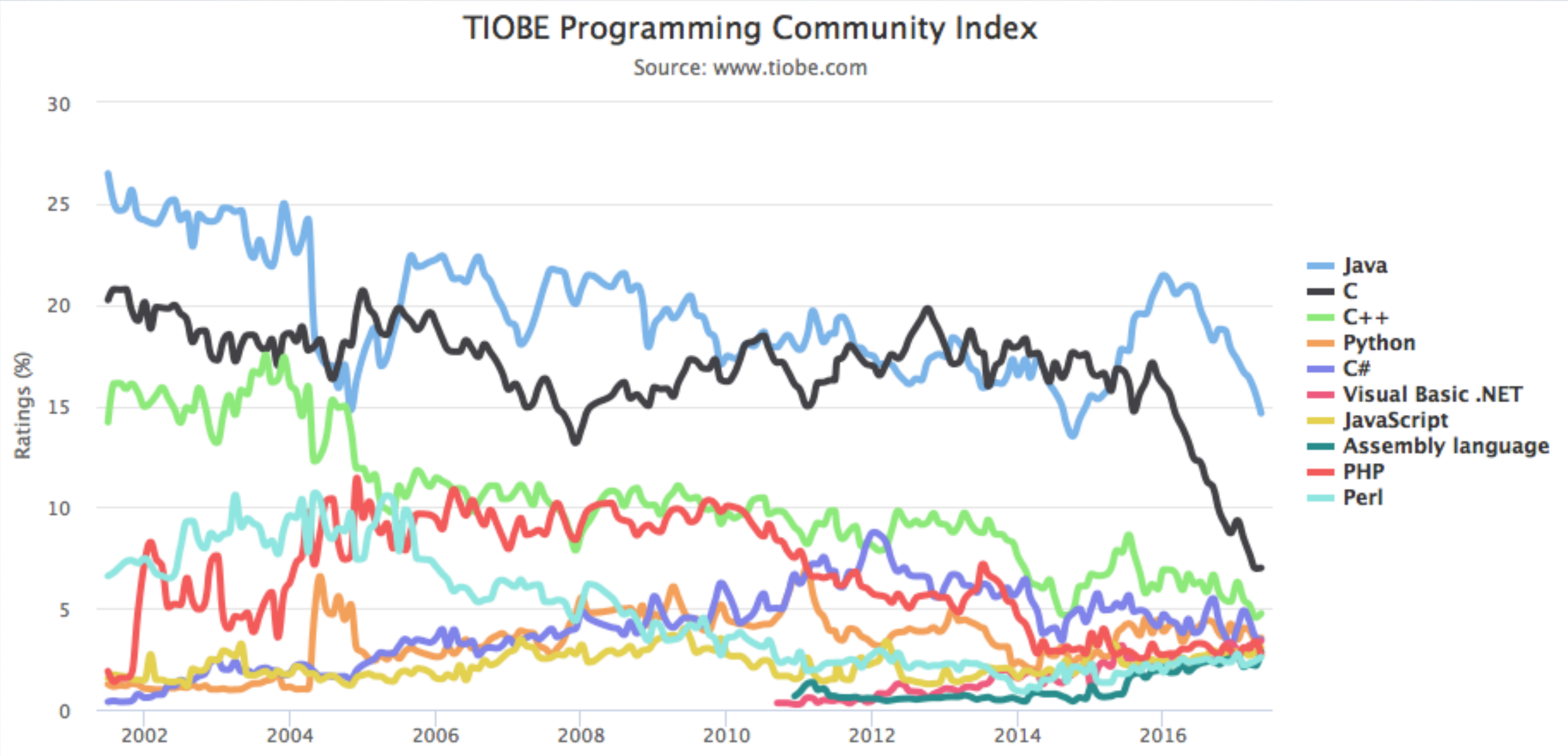
- Rough idea in the late 1980s
- Meant to descend the ABC language
- First line of code in December 1989 by Guido van Rossum
- Python 2.0 in October 2000
- Python 3.0 in December 2008

PYTHONS POPULARITY



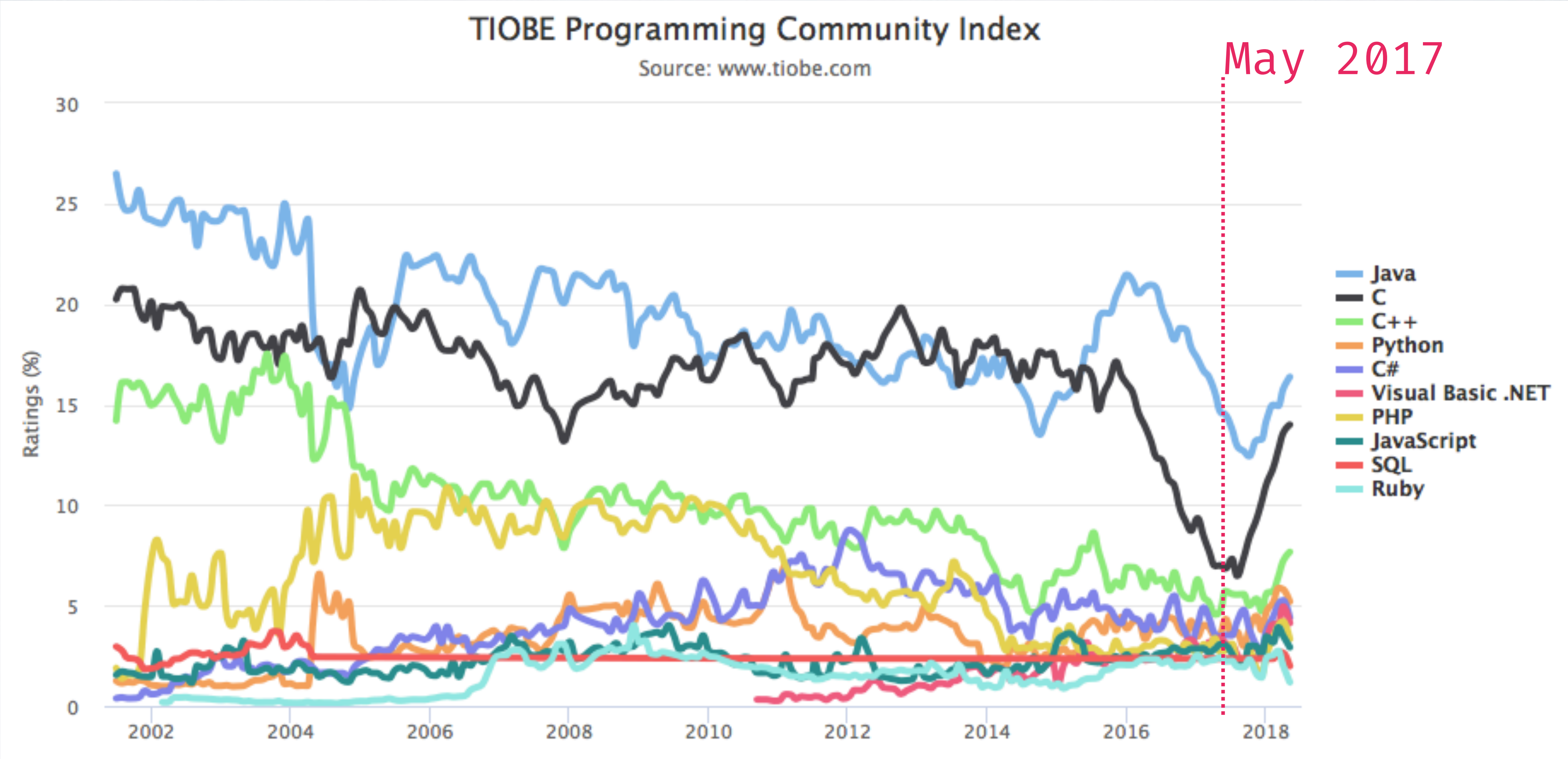
“Programming language of the year” in 2007 and 2010.

POPULAR LANGUAGES
(MAY 2017)



Python is the fourth most popular language
and rocks the top 10 since 2003.

POPULAR LANGUAGES (MAY 2018)



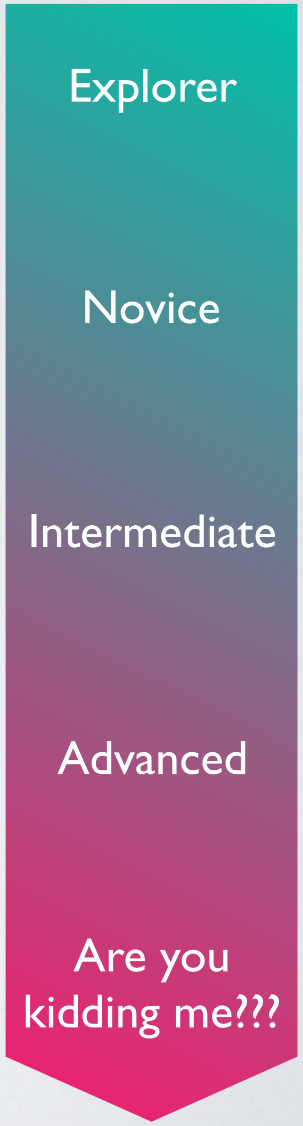
Python is still the fourth most popular language and rocks the top 10 since 2003.

YOUR JOURNEY THROUGH PYTHON?
(JUST A VERY ROUGH GUESS, NOT A MEAN GAME)

Raise your hand and keep it up until you answer a question with “no”.

- Have you ever launched the Python interpreter?
- Wrote for/while-loops or if/else statements?
- …your own functions?
- …classes?
- …list/dict/set comprehensions?
- Do you know what a generator is?
- Have you ever implemented a decorator?
- …a metaclass?
- …a C-extension?
- Do you know and can you explain the output of the following line?

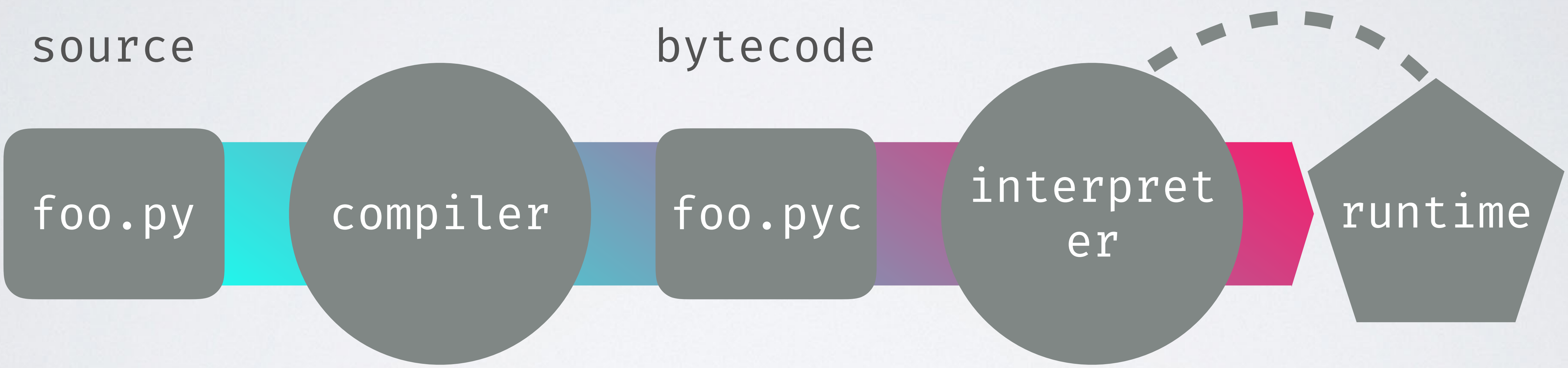
```
print(5 is 7 - 2, 300 is 302 - 2)
```



BASIC PYTHON INTERNALS

to understand the performance issues

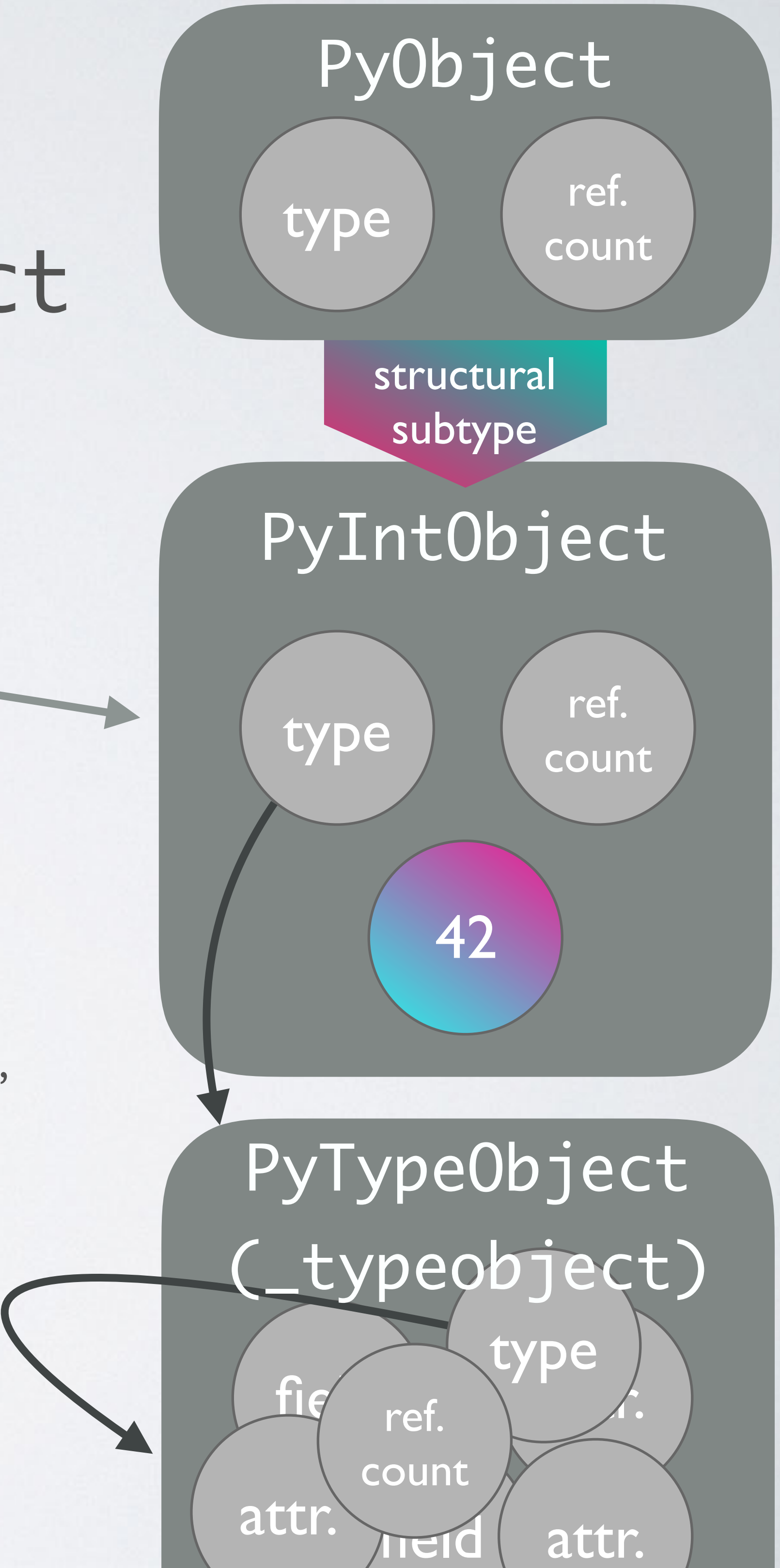
FROM SOURCE TO RUNTIME



DATA IN PYTHON

- Every piece of data is a PyObject

```
>>> dir(42)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length',
'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real',
'to_bytes']
```



THE TYPE OF A PyObject

“An object has a ‘type’ that determines what it represents and what kind of data it contains.
An object’s type is fixed when it is created.
Types themselves are represented as objects.
The type itself has a type pointer pointing to the object representing the type ‘type’, which contains a pointer to itself!”

— object.h

YOUR BEST FRIEND AND WORST ENEMY:

GIL - Global Interpreter Lock

- The GIL prevents parallel execution of (Python) bytecode
- Even though Python has real threads, they never execute code at the same time
- Context switching between threads creates overhead (the user cannot control thread-priority)
- Threads perform pretty bad on CPU bound tasks
- They do a great job speeding up I/O heavy tasks

THREADS AND CPU BOUND TASKS

single thread:

```
N = 100000000
```

```
def count(n):
    while n != 0: n -=1
```

```
%time count(N)
```

```
CPU times: user 5.59 s, sys: 32.5 ms, total: 5.62 s
Wall time: 7.71 s
```

two threads:

```
from threading import Thread
```

```
def count_threaded(n):
    t1 = Thread(target=count, args=(N/2,))
    t2 = Thread(target=count, args=(N/2,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

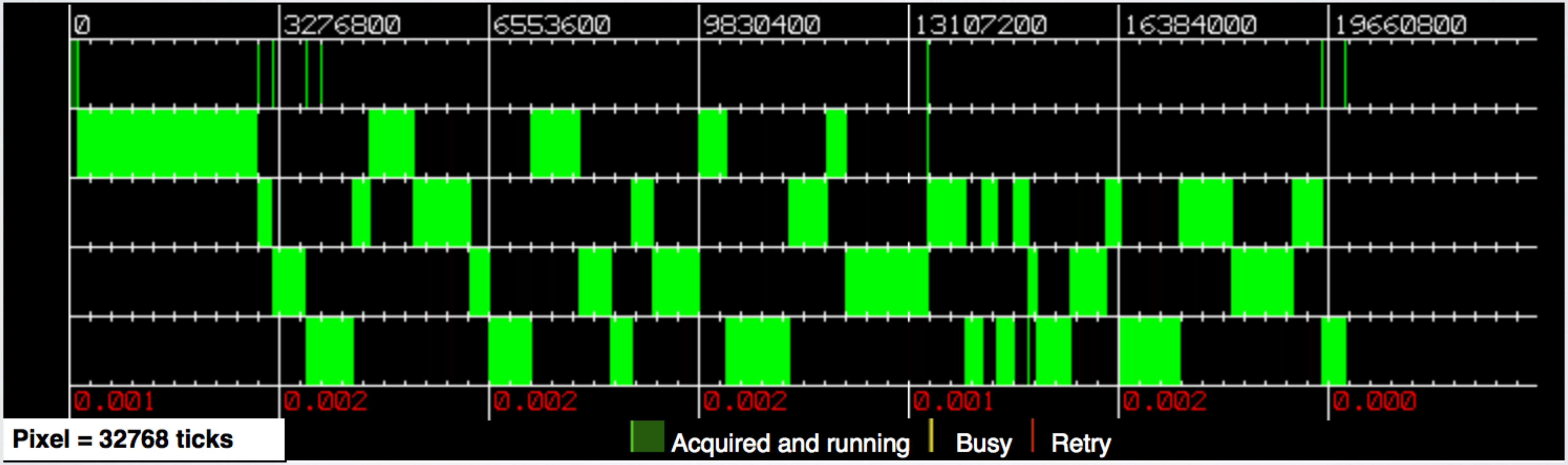
```
%time count_threaded(N)
```

```
CPU times: user 7.18 s, sys: 31 ms, total: 7.21 s
Wall time: 9.01 s
```

This is probably not really what you expected…

THREADS FIGHTING FOR THE GIL

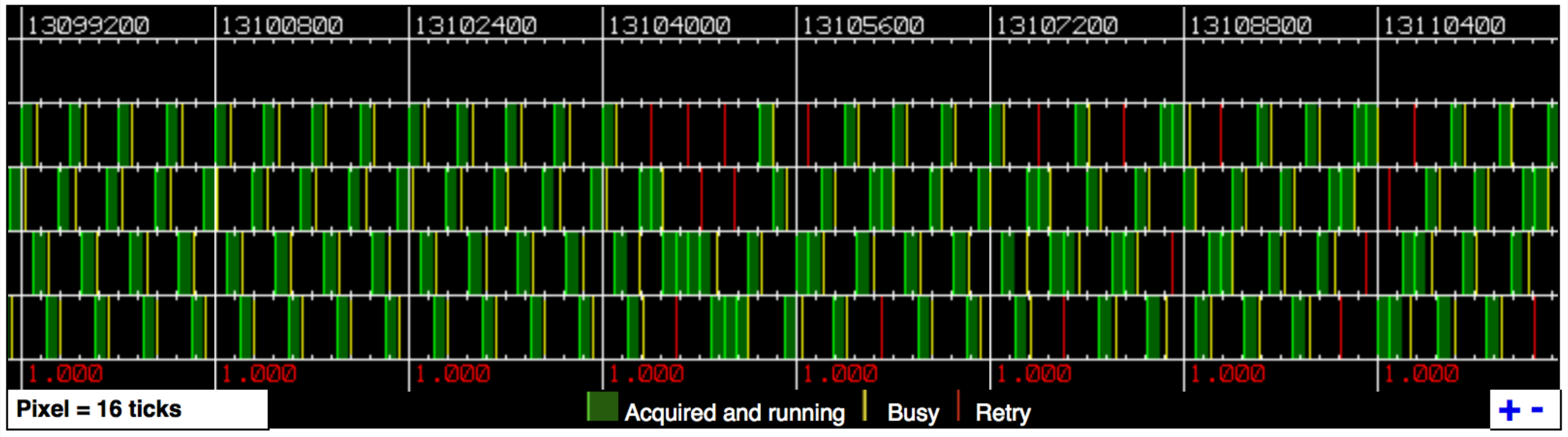
OS X: 4 threads on 1 CPU (Python 2.6)



By David M Beazley: http://dabeaz.com/GIL/gilvis

THREADS FIGHTING FOR THE GIL

OS X: 4 threads on 4 CPUs (Python 2.6)



By David M Beazley: http://dabeaz.com/GIL/gilvis

OK, huge overhead for every single object,
no real parallel execution of code…

How should Python ever compete with all those super fast C/Fortran libraries?

C-extensions and interfacing C/Fortran!

Those can release the GIL and do the heavy stuff in the background.

A DUMB SPEED COMPARISON

CALCULATING THE MEAN OF 1000000 RANDOM NUMBERS

pure Python:

```python
def mean(numbers):
    return sum(numbers)/len(numbers)
```

```python
numbers = list(range(1000000))
%timeit mean(numbers)
```

```
8.59 ms ± 234 µs per loop
```

NumPy (~13x faster):

```python
numbers = np.random.random(1000000)
%timeit np.mean(numbers)
```

```
638 µs ± 38.3 µs per loop
```

Numba (~8x faster):

```python
@nb.jit
def numba_mean(numbers):
    s = 0
    N = len(numbers)
    for i in range(N):
        s += numbers[i]
    return s/N
```

```python
numbers = np.random.random(1000000)
%timeit numba_mean(numbers)
```

```
1.1 ms ± 6.64 µs per loop
```

Julia (~16x faster):

```julia
numbers = rand(1000000)
@benchmark mean(numbers)
```

```
BenchmarkTools.Trial:
  memory estimate:  16 bytes
  allocs estimate:  1
  --------------
  minimum time:     464.824 µs (0.00% GC)
  median time:      524.386 µs (0.00% GC)
  mean time:        544.573 µs (0.00% GC)
  maximum time:     2.095 ms (0.00% GC)
  --------------
  samples:          8603
  evals/sample:     1
```

CRAZY LLVM COMPILER OPTIMISATIONS

SUMMING UP NUMBERS FROM 0 TO N=100,000,000

pure Python:

```
def simple_sum(N):
    s = 0
    for i in range(N):
        s += i
    return s
```

```
%time simple_sum(N)
```

```
CPU times: user 7.13 s, sys: 103 ms, total: 7.23 s
Wall time: 7.43 s

4999999950000000
```

NumPy (~80x faster):

```
np_numbers = np.array(range(N))

%time np.sum(np_numbers)
```

```
CPU times: user 84 ms, sys: 2.65 ms, total: 86.6 ms
Wall time: 91.1 ms

4999999950000000
```

Numba (~300000x faster):

```
@nb.jit
def simple_sum(N):
    s = 0
    for i in range(N):
        s += i
    return s
```

```
%time numba_sum(N)
```

```
CPU times: user 11 µs, sys: 3 µs, total: 14 µs
Wall time: 21.9 µs

4999999950000000
```

Julia (~7000000x faster):

```
function simple_sum(N)
    s = 0
    for i ∈ 1:N
        s += i
    end
    return s
end
```

```
simple_sum (generic fu
```

```
@time simple_sum(N)
```

```
  0.000002 seconds (5

4999999950000000
```

```
        pushq   %rbp
        movq    %rsp, %rbp
        xorl    %eax, %eax
Source line: 3
        testq   %rdi, %rdi
        jle     L32
        leaq    -1(%rdi), %rax
        leaq    -2(%rdi), %rcx
        mulq    %rcx
        shldq   $63, %rax, %rdx
        leaq    -1(%rdx,%rdi,2), %rax
Source line: 6
L32:
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

PYTHON LIBRARIES

for scientific computing

pandas

NumPy

Matplotlib

SymPy

IPython

Jupyter

AstroPy

Not part of NumFocus but covered in this talk:

Numba

Numexpr

NUMFOCUS
OPEN CODE = BETTER SCIENCE

SCIPY

Scientific Computing Tools for Python

THE SCIPY STACK

- Core packages
  - SciPy Library: numerical algorithms, signal processing, optimisation, statistics etc.
  - NumPy
  - Matplotlib: 2D/3D plotting library
  - pandas: high performance, easy to use data structures
  - SymPy: symbolic mathematics and computer algebra
  - IPython: a rich interactive interface to process data and test ideas
  - Jupyter: notebooks to document and code at the same time
  - nose: testing framework for Python code
- Other packages:
  - Chaco, Mayavi, Cython, Scikits (scikit-learn, scikit-image), h5py, PyTables and much more

https://www.scipy.org

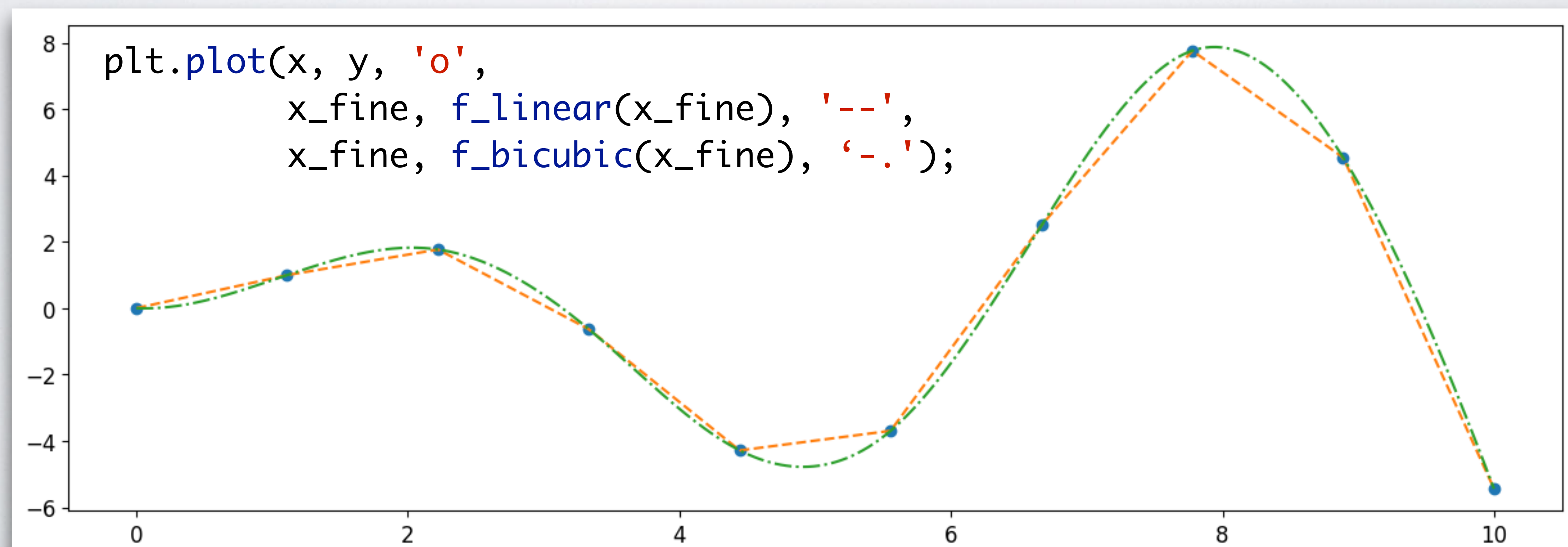
SCIPY CORE LIBRARY

- Clustering package (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms (scipy.fftpack)
- Integration and ODEs (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Miscellaneous routines (scipy.misc)
- Multi-dimensional image processing (scipy.ndimage)
- Orthogonal distance regression (scipy.odr)
- Optimization and root finding (scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial algorithms and data structures (scipy.spatial)
- Special functions (scipy.special)
- Statistical functions (scipy.stats)
- Statistical functions for masked arrays (scipy.stats.mstats)

SCIPY INTERPOLATE

```
from scipy import interpolate

x = np.linspace(0, 10, 10)
y = np.sin(x)

x_fine = np.linspace(0, 10, 500)

f_linear = interpolate.interp1d(x, y, kind='linear')
f_bicubic = interpolate.interp1d(x, y, kind='cubic')
```

```
plt.plot(x, y, 'o',
         x_fine, f_linear(x_fine), '--',
         x_fine, f_bicubic(x_fine), ‘-.');
```



NUMFOCUS

OPEN CODE = BETTER SCIENCE

NUMPY

Numerical Python

NUMPY

NumPy is the fundamental package for scientific computing with Python.

- gives us a powerful N-dimensional array object: ndarray
- broadcasting functions
- tools for integrating C/C++ and Fortran
- linear algebra, Fourier transform and random number capabilities
- most of the scientific libraries build upon NumPy

NUMPY: ndarray

```
a = np.arange(6)
a
```

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

ndim: 1
shape: (6,)

Continuous array in memory with a fixed type,
no pointer madness!

C/Fortran compatible memory layout,
so they can be passed to those
without any further efforts.

NUMPY: ARRAY OPERATIONS AND ufuncs

```
a * 23
```

```
array([  0,  23,  46,  69,  92, 115])
```

```
a**a
```

```
array([   1,    1,    4,   27,  256, 3125])
```

easy and intuitive element-wise operations

a ufunc, which can operate both on scalars and arrays (element-wise)

```
np.exp(a)
```

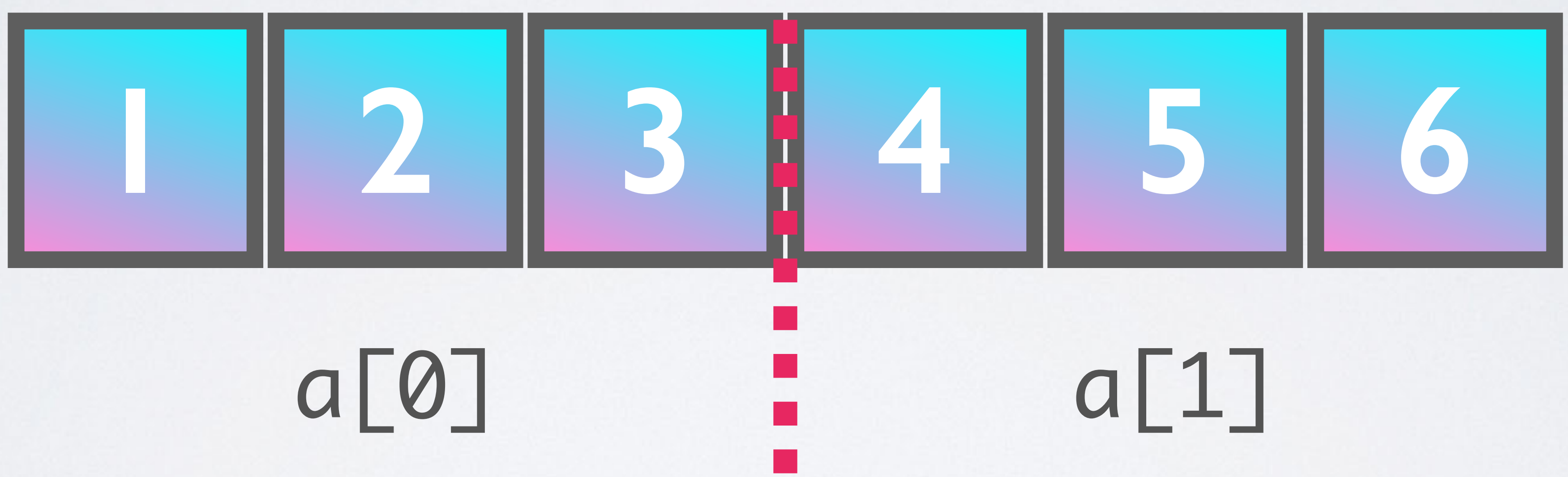
```
array([   1.        ,    2.71828183,    7.3890561 ,   20.08553692,
         54.59815003,  148.4131591 ])
```

RESHAPING ARRAYS

```
a = np.arange(6)
a
```

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

ndim: 1
shape: (6,)




```
a.reshape(2, 3)
```

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

No rearrangement of the elements
but setting the iterator limits internally!

RESHAPING ARRAYS IS CHEAP

```
a = np.arange(10000000)
```

```
%timeit b = a.reshape(100, 5000, 20)
```

```
563 ns ± 8.18 ns per loop (mean ± std.
```

Don't worry, we will discover NumPy in the hands-on workshop!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

matplotlib

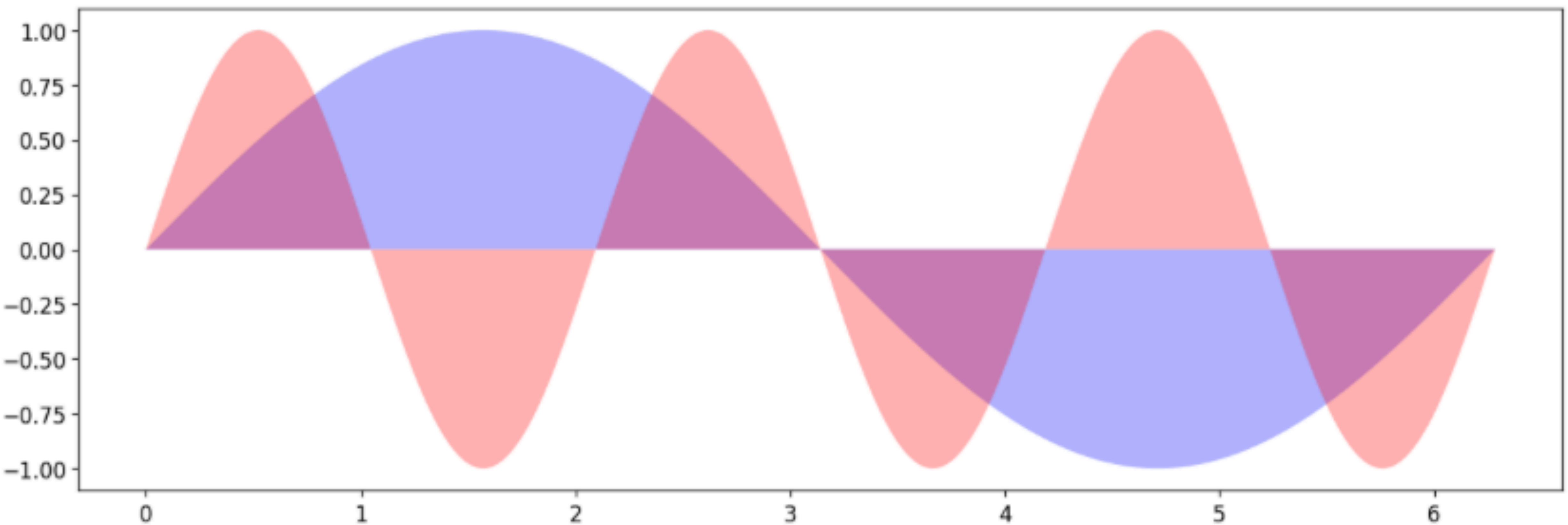
MATPLOTLIB

A Python plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments.

- Integrates well with IPython and Jupyter
- Plots, histograms, power spectra, bar charts, error chars, scatterplots, etc. with an easy to use API
- Full control of line styles, font properties, axes properties etc.
- The easiest way to get started is browsing its wonderful gallery full of thumbnails and copy&paste examples: http://matplotlib.org/gallery.html

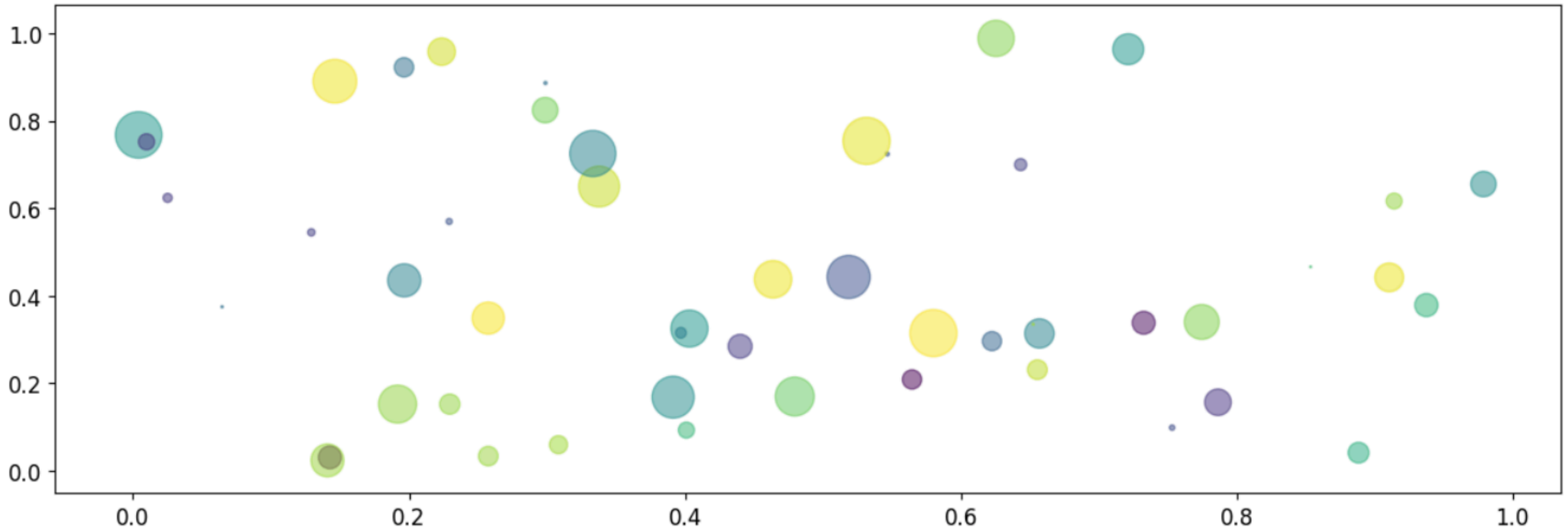
MATPLOTLIB EXAMPLE

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 500)
y1 = np.sin(x)
y2 = np.sin(3 * x)

fig, ax = plt.subplots()
ax.fill(x, y1, 'b', x, y2, 'r', alpha=0.3)
plt.show()
```



MATPLOTLIB EXAMPLE

```
import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



NUMFOCUS

OPEN CODE = BETTER SCIENCE

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

PANDAS

A Python Data Analysis Library inspired by data frames in R:

- gives us a powerful data structure: *DataFrame*
- database-like handling of data
- integrates well with NumPy
- wraps the Matplotlib API (which can also cause troubles ;)
- has a huge number of I/O related functions to parse data: CSV, HDF5, SQL, Feather, JSON, HTML, Excel, and more…

THE DataFrame

A table-like structure, where you can access elements by row and column.

```
hits = pd.read_hdf("event_file.h5", "events/23")
hits.head(3)
```

	channel_id	dom_id	event_id	id	pmt_id	time	tot	triggered
0	25	808430036	0	0	0	30652287	21	0
1	18	808430036	0	0	0	30656200	16	0
2	15	808430449	0	0	0	30648451	26	0

THE DataFrame

Lots of functions to allow filtering, manipulating and aggregating the data to fit your needs.

```
active_doms = hits.pivot_table(index='event_id',
                               values='dom_id',
                               aggfunc=lambda x: set(x))
```

Don't worry, we will discover Pandas in the hands-on workshop!

sponsored by

CONTINUUM®
ANALYTICS

NUMBA

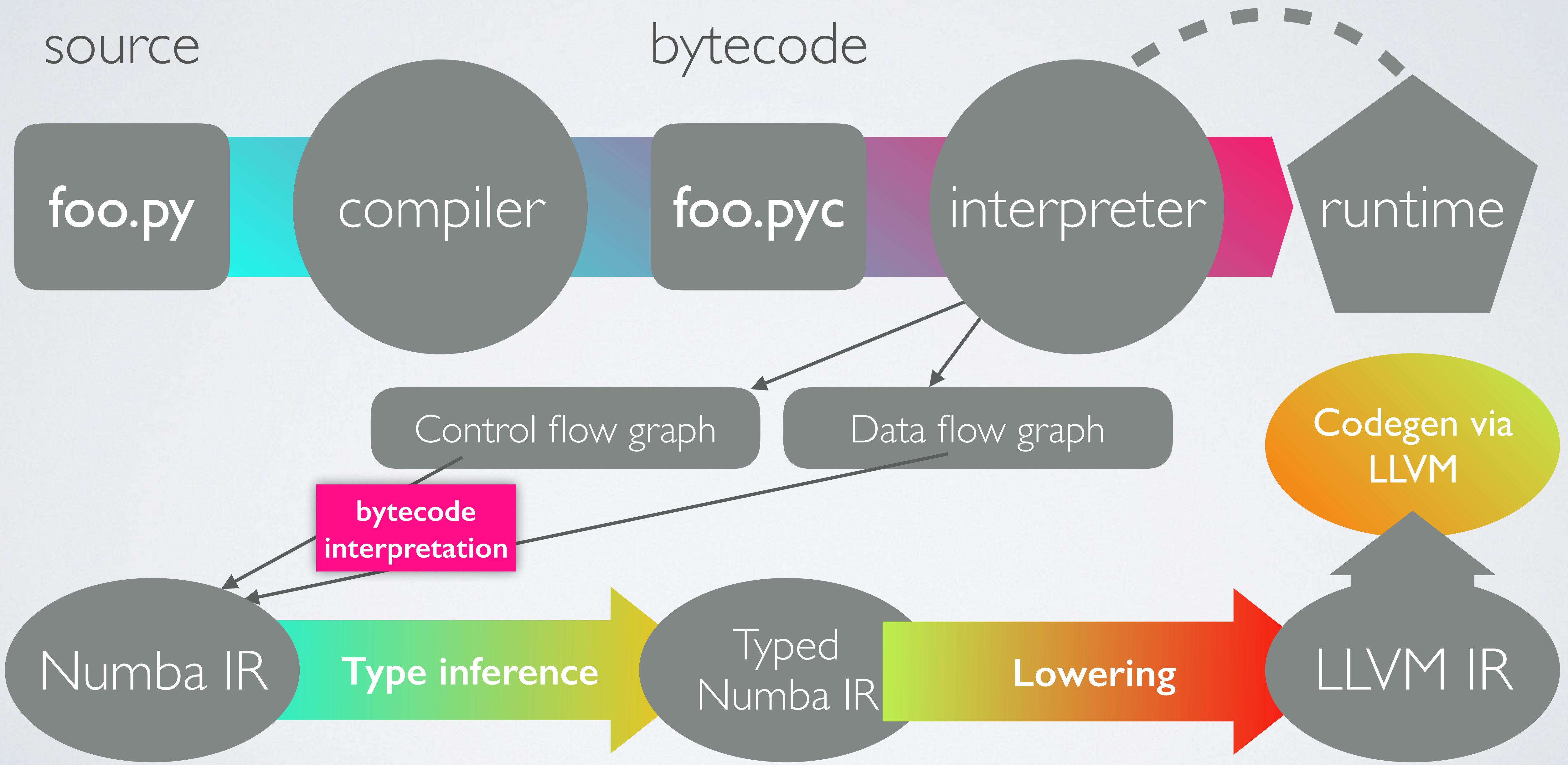
JIT (LLVM) compiler for Python

NUMBA

Numba is a compiler for Python array and numerical functions that gives you the power to speed up code written directly in Python.

- uses LLVM to boil down pure Python code to JIT optimised machine code
- only accelerates selected functions decorated by yourself
- native code generation for CPU (default) and GPU
- integration with the Python scientific software stack (thanks to NumPy)
- runs side by side with regular Python code or third-party C extensions and libraries
- great CUDA support
- N-core scalability by releasing the GIL (beware: no protection from race conditions!)
- create NumPy ufuncs with the @[gu]vectorize decorator(s)

FROM SOURCE TO RUNTIME



NUMBA JIT-EXAMPLE

```
numbers = np.arange(1000000).reshape(2500, 400)
```

```
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
```

```
@nb.jit
def sum2d_jit(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
```

289 ms ± 3.02 ms per loop

2.13 ms ± 42.6 µs per loop

~135x faster, with a single line of code

NUMBA VECTORIZE-EXAMPLE

```
a = np.arange(1000000, dtype='f8')
b = np.arange(1000000, dtype='f8') + 23
```

NumPy:

```
np.abs(a - b) / (np.abs(a) + np.abs(b))
```

23 ms ± 845 µs per loop

Numba @vectorize:

```
@nb.vectorize
def nb_rel_diff(a, b):
    return abs(a - b) / (abs(a) + abs(b))

rel_diff(a, b)
```

3.56 ms ± 43.2 µs per loop

~6x faster

NUMEXPR

initially written by David Cooke

Routines for the fast evaluation of array expressions element-wise
by using a vector-based virtual machine.

NUMEXPR USAGE EXAMPLE

```
import numpy as np
import numexpr as ne

a = np.arange(5)
b = np.linspace(0, 2, 5)

ne.evaluate("a**2 + 3*b”)


array([  0. ,    2.5,    7. ,   13.5,   22. ])
```

NUMEXPR SPEED-UP

```
a = np.random.random(1000000)
```

NumPy:

```
2 * a**3 - 4 * a**5 + 6 * np.log(a)
```

82.4 ms ± 1.88 ms per loop

Numexpr with 4 threads:

```
ne.set_num_threads(4)

ne.evaluate("2 * a**3 - 4 * a**5 + 6 * log(a)")
```

7.85 ms ± 103 µs per loop

~10x faster

NUMEXPR - SUPPORTED OPERATORS

- Logical operators: &, |, ~
- Comparison operators:
  <, <=, ==, !=, >=, >
- Unary arithmetic operators: -
- Binary arithmetic operators:
  +, -, *, /, **, %, <<, >>

NUMEXPR - SUPPORTED FUNCTIONS

- where(bool, number1, number2): number -- number1 if the bool condition is true, number2 otherwise.
- {sin,cos,tan}(float|complex): float|complex -- trigonometric sine, cosine or tangent.
- {arcsin,arccos,arctan}(float|complex): float|complex -- trigonometric inverse sine, cosine or tangent.
- arctan2(float1, float2): float -- trigonometric inverse tangent of float1/float2.
- {sinh,cosh,tanh}(float|complex): float|complex -- hyperbolic sine, cosine or tangent.
- {arcsinh,arccosh,arctanh}(float|complex): float|complex -- hyperbolic inverse sine, cosine or tangent.
- {log,log10,log1p}(float|complex): float|complex -- natural, base-10 and log(1+x) logarithms.
- {exp,expm1}(float|complex): float|complex -- exponential and exponential minus one.
- sqrt(float|complex): float|complex -- square root.
- abs(float|complex): float|complex -- absolute value.
- conj(complex): complex -- conjugate value.
- {real,imag}(complex): float -- real or imaginary part of complex.
- complex(float, float): complex -- complex from real and imaginary parts.
- contains(str, str): bool -- returns True for every string in `op1` that contains `op2`.
- sum(number, axis=None): Sum of array elements over a given axis. Negative axis are not supported.
- prod(number, axis=None): Product of array elements over a given axis. Negative axis are not supported.

NUMFOCUS

OPEN CODE = BETTER SCIENCE

astropy

A Community Python Library for Astronomy

THE HISTORY OF ASTROPY

(standard situation back in 2011)

- Example Problem: `convert from EQ J2000 RA/Dec to Galactic coordinates`
- Solution in Python
  - ~~pyast~~
  - ~~Astrolib~~
  - ~~Astrophysi~~
  - ~~PyEphem~~
  - ~~PyAstro~~
  - ~~Kapteyn~~
  - ~~???~~

huge discussion started in June 2011
series of votes

astropy

First public version (v0.2) presented and described in the following paper:
http://adsabs.harvard.edu/abs/2013A%26A...558A..33A

ASTROPY CORE PACKAGE

A community-driven package intended to contain much of the core functionality and some common tools needed for performing astronomy and astrophysics with Python.

- **Data structures and transformations**
  - constants, units and quantities, N-dimensional datasets, data tables, times and dates, astronomical coordinate system, models and fitting, analytic functions
- **Files and I/O**
  - unified read/write interface
  - FITS, ASCII tables, VOTable (XML), Virtual Observatory access, HDF5, YAML, …
- **Astronomy computations and utilities**
  - cosmological calculations, convolution and filtering, data visualisations, astrostatistics tools

ASTROPY AFFILIATED PACKAGES

- Tons of astronomy related packages
- which are not part of the core package,
- but has requested to be included as part of the Astropy project’s community

ASTROPY EXAMPLE

```
from astropy.utils.data import download_file
from astropy.io import fits

image_file = download_file('http://data.astropy.org/tutorials/FITS-images/HorseHead.fits')
```

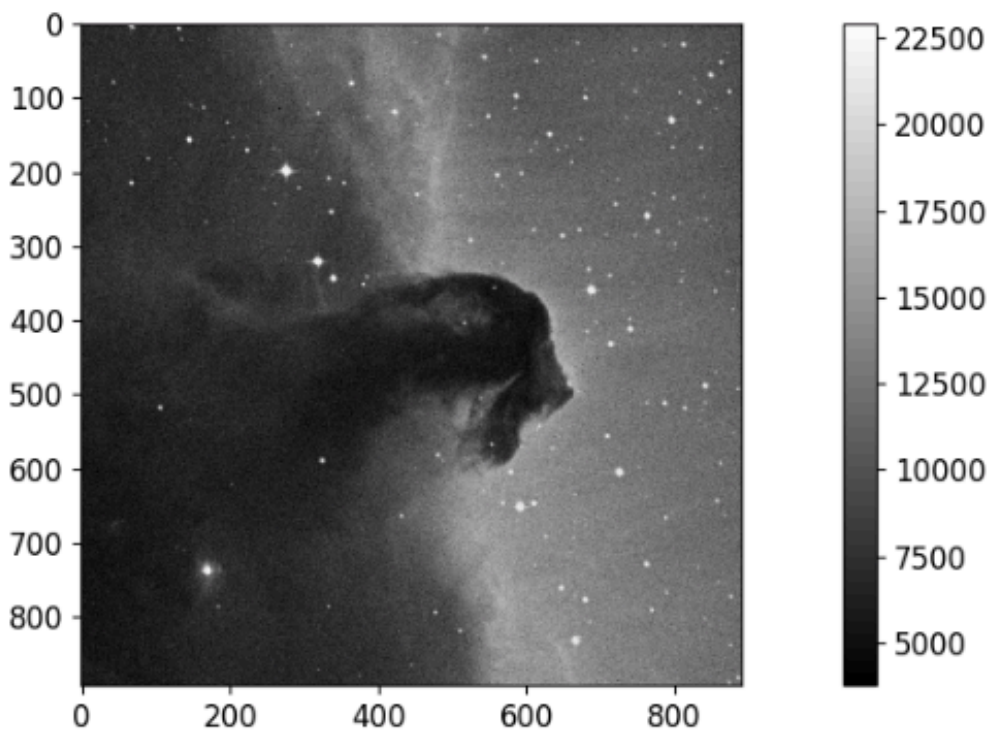
```
Downloading http://data.astropy.org/tutorials/FITS-images/HorseHead.fits [Done]
```

```
fits.info(image_file)
```

```
Filename: /Users/tamasgal/.astropy/cache/download/py3/2c9202ae878ecfcb60878ceb63837f5f
No.    Name         Type      Cards   Dimensions   Format
  0  PRIMARY     PrimaryHDU     161   (891, 893)   int16
  1  er.mask     TableHDU        25   1600R x 4C   [F6.2, F6.2, F6.2, F6.2]
```

```
image_data = fits.getdata(image_file, ext=0)
```

```
plt.figure()
plt.imshow(image_data, cmap='gray');
plt.colorbar();
```



← downloading via HTTP

← checking some FITS meta

← extracting image data

← plotting via Matplotlib

ASTROPY EXAMPLE

```
from astropy.coordinates import SkyCoord
import astropy.units as u

m13 = SkyCoord.from_name('m13')
m13
```

```
<SkyCoord (ICRS): (ra, dec) in deg
    ( 250.4234583,  36.4613056)>
```

```
m13.ra, m13.ra.to(u.hourangle)
```

```
(<Longitude 250.4234583 deg>, <Longitude 16.69489722 hourangle>)
```

Don’t worry, we will discover AstroPy in the hands-on workshop!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

SymPy

A Python library for symbolic mathematics.

SIMPY

- It aims to become a full-featured computer algebra system (CAS)
- while keeping the code as simple as possible
- in order to be comprehensible and easily extensible.
- SymPy is written entirely in Python.
- It only depends on mpmath, a pure Python library for arbitrary floating point arithmetic

SIMPY

- solving equations
- solving differential equations
- simplifications: trigonometry, polynomials
- substitutions
- factorisation, partial fraction decomposition
- limits, differentiation, integration, Taylor series
- combinatorics, statistics, …
- much much more

SIMPY EXAMPLE

Base Python

```
In [1]: import math

In [2]: math.sqrt(8)
Out[2]: 2.8284271247461903

In [3]: math.sqrt(8)**2
Out[3]: 8.000000000000002
```

SymPy

```
In [4]: import sympy

In [5]: sympy.sqrt(8)
Out[5]: 2*sqrt(2)

In [6]: sympy.sqrt(8)**2
Out[6]: 8
```

SIMPY EXAMPLE

```
In [15]: x, y = sympy.symbols('x y')

In [16]: expr = x + 2*y

In [17]: expr
Out[17]: x + 2*y

In [18]: expr + 1
Out[18]: x + 2*y + 1

In [19]: expr * x
Out[19]: x*(x + 2*y)

In [20]: sympy.expand(expr * x)
Out[20]: x**2 + 2*x*y
```

SIMPY EXAMPLE

```
In [1]: import sympy

In [2]: from sympy import init_printing, integrate, diff, exp, cos, sin, oo

In [3]: init_printing(use_unicode=True)

In [4]: x = sympy.symbols('x')

In [5]: diff(sin(x)*exp(x), x)
Out[5]:
 x            x
e ⋅sin(x) + e ⋅cos(x)

In [6]: integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
Out[6]:
 x
e ⋅sin(x)

In [7]: integrate(sin(x**2), (x, -oo, oo))
Out[7]:
√2⋅√π
─────
  2
```

IPython

IPYTHON

- The interactive Python shell!
- Object introspection
- Input history, persistent across sessions
- Extensible tab completion
- “Magic” commands (basically macros)
- Easily embeddable in other Python programs and GUIs
- Integrated access to the pdb debugger and the Python profiler
- Syntax highlighting
- real multi-line editing
- Provides a kernel for Jupyter
- …and such more!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

jupyter

Project Jupyter is an open source project that offers a set of tools for interactive and exploratory computing.

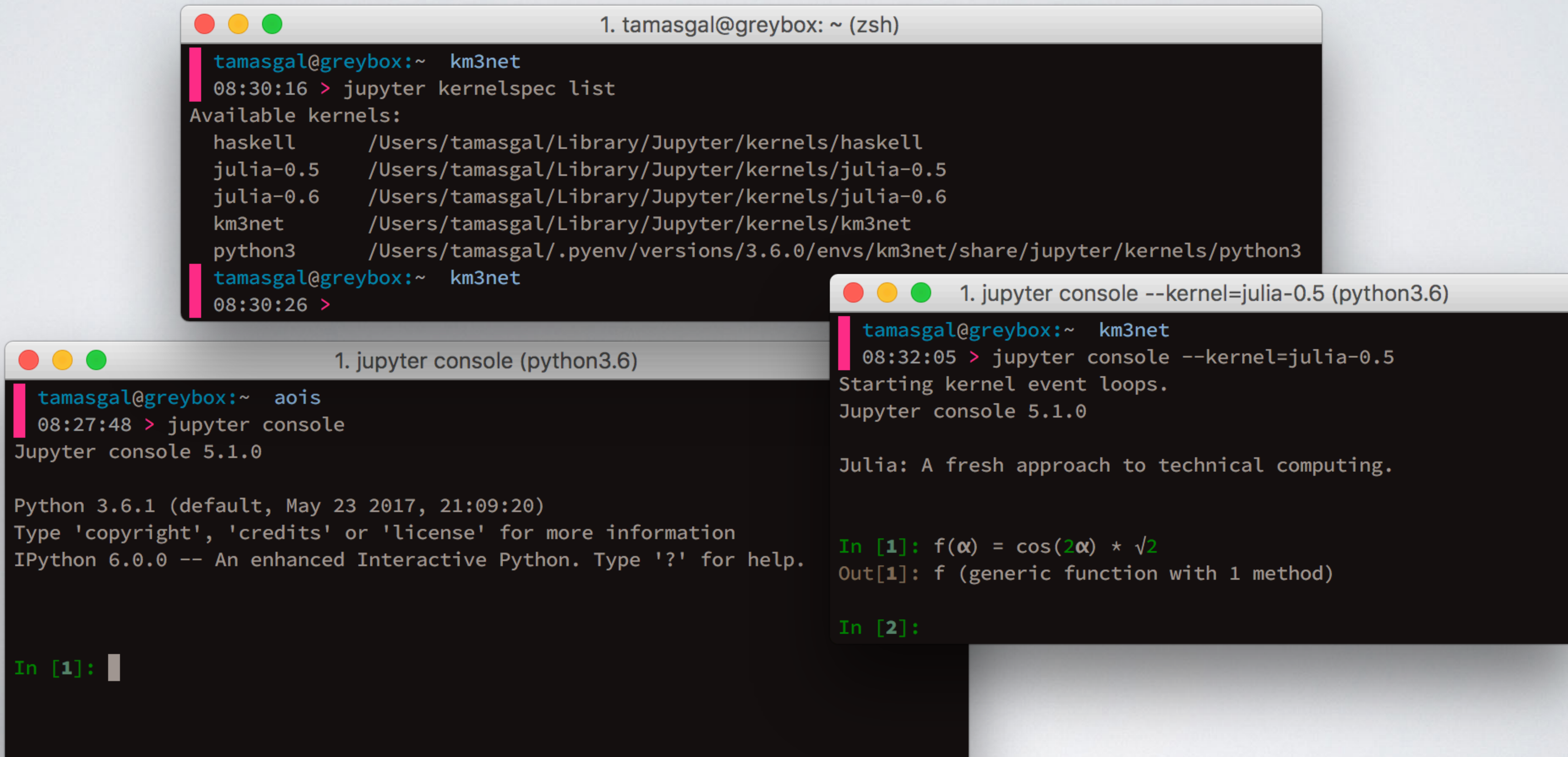
JUPYTER

- Born out of the IPython project in 2014
- Jupyter provides a console and a notebook server for all kinds of languages

  (the name Jupyter comes from **Ju**lia, **Pyt**hon and **R**)
- An easy way to explore and prototype
- Notebooks support Markdown and LaTeX-like input and rendering
  - Allows sharing code and analysis results
  - Extensible (slideshow plugins, JupyterLab, VIM binding, …)

JUPYTER CONSOLE

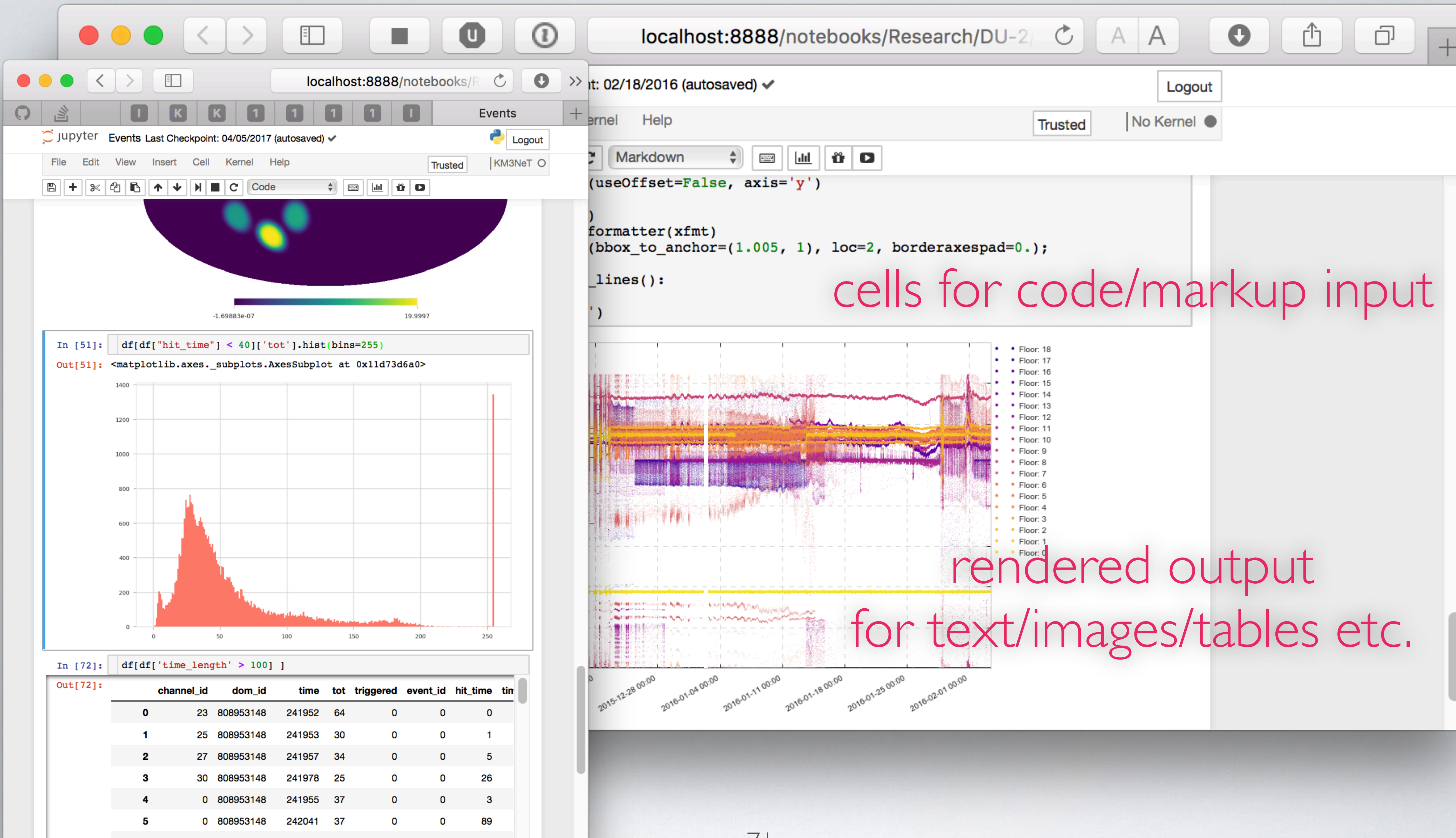
A terminal frontend for kernels which use the Jupyter protocol.



JUPYTER NOTEBOOK

- A Web-based application suitable for capturing the whole computation process:
  - developing
  - documenting
  - and executing code
  - as well as communicating the results.
- Two main components:
  - a web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.
  - notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

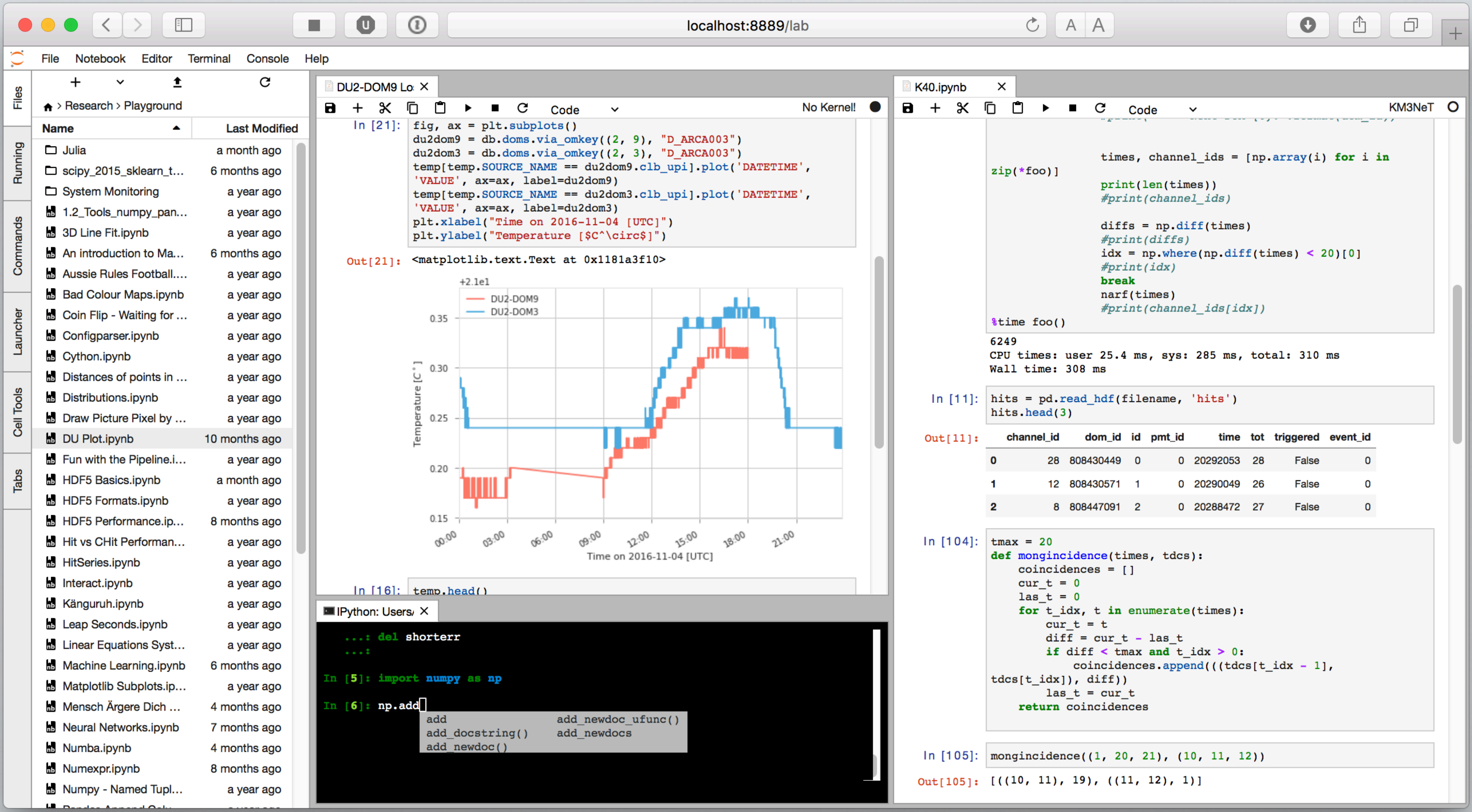
JUPYTER NOTEBOOK



JUPYTERLAB

- The next level of interacting with notebooks
- Extensible: terminal, text editor, image viewer, etc.
- Supports editing multiple notebooks at once
- Drag and drop support to arrange panes

JUPYTERLAB


localhost:8889/lab
File Notebook Editor Terminal Console Help
Research > Playground
Name Last Modified
DU2-DOM9 Lo
K40.ipynb
No Kernel!
KM3NeT
IPython: Users
Time on 2016-11-04 [UTC]
Temperature [C°]
DU2-DOM9
DU2-DOM3

JUPYTERHUB

- JupyterHub creates a multi-user Hub which spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server
- A nice environment for teaching
- Great tool for collaborations (ask your IT admin ;)

SOME OTHER USEFUL LIBRARIES

SEABORN

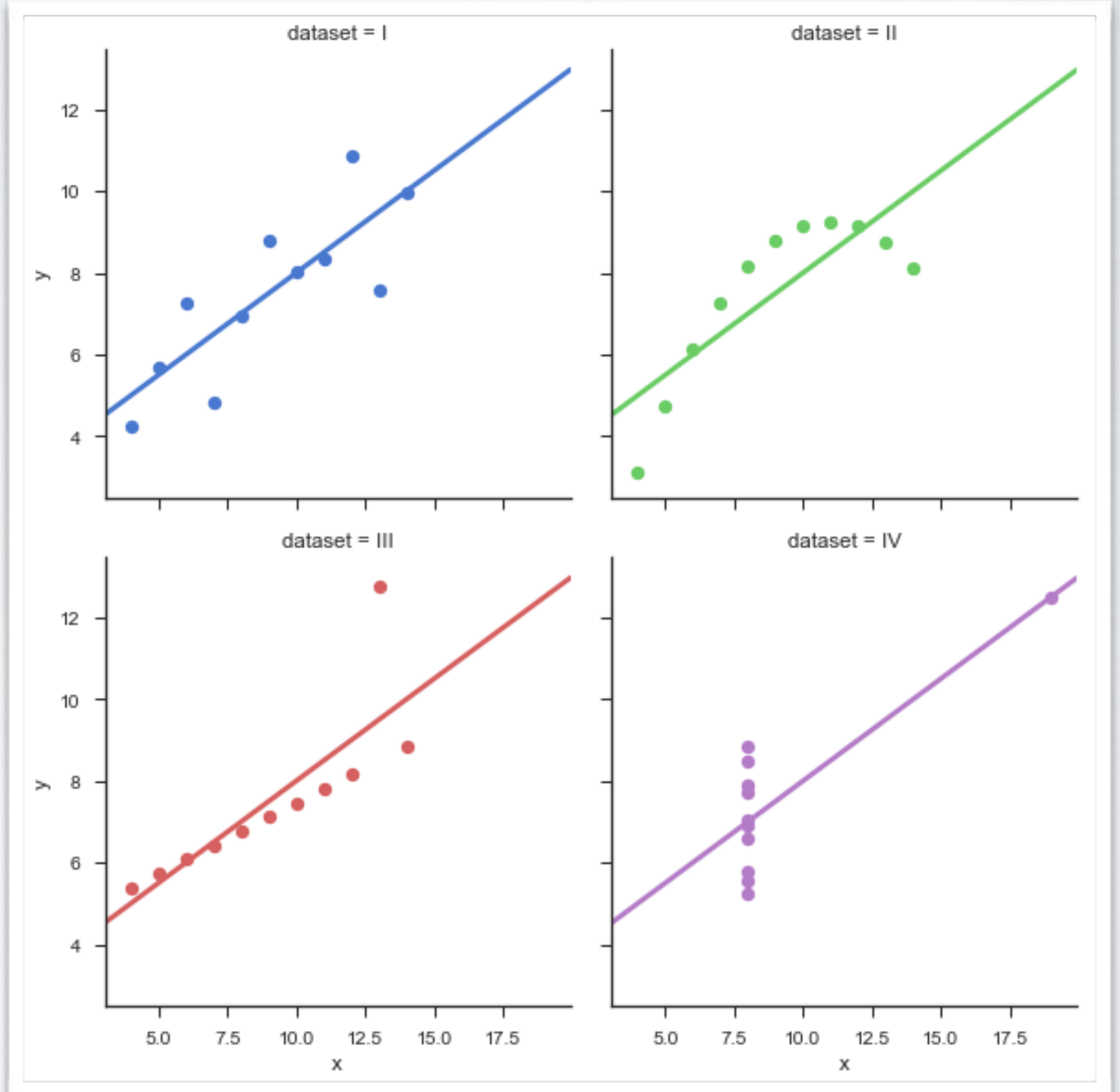
statistical data visualisation

uses matplotlib as backend

https://seaborn.pydata.org

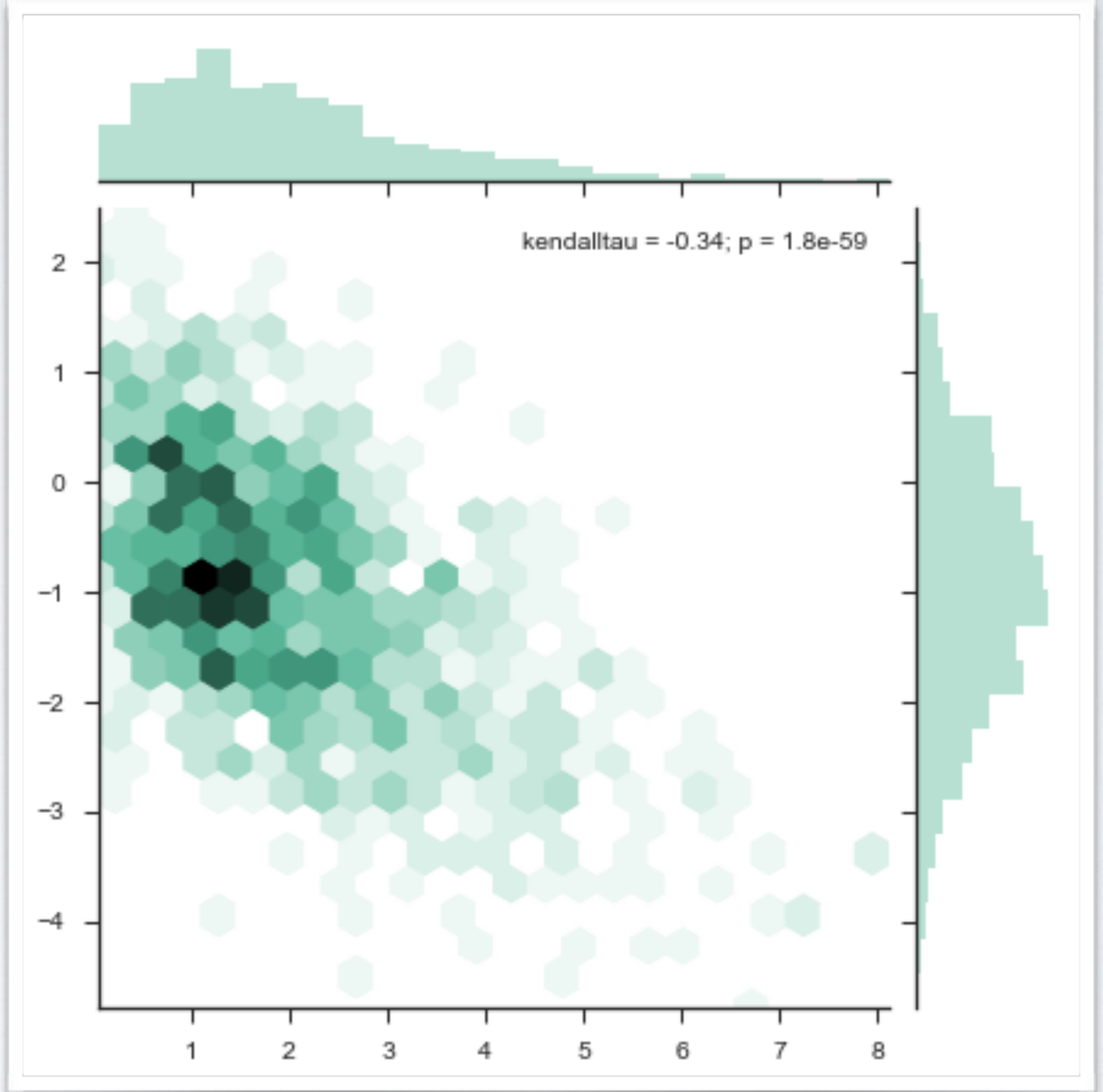
CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("anscombe")

# Show the results of a linear regression
# within each dataset
sns.lmplot(x="x", y="y", col="dataset",
           hue="dataset", data=df,
           col_wrap=2, ci=None,
           palette="muted", size=4,
           scatter_kws={"s": 50, "alpha": 1})
```



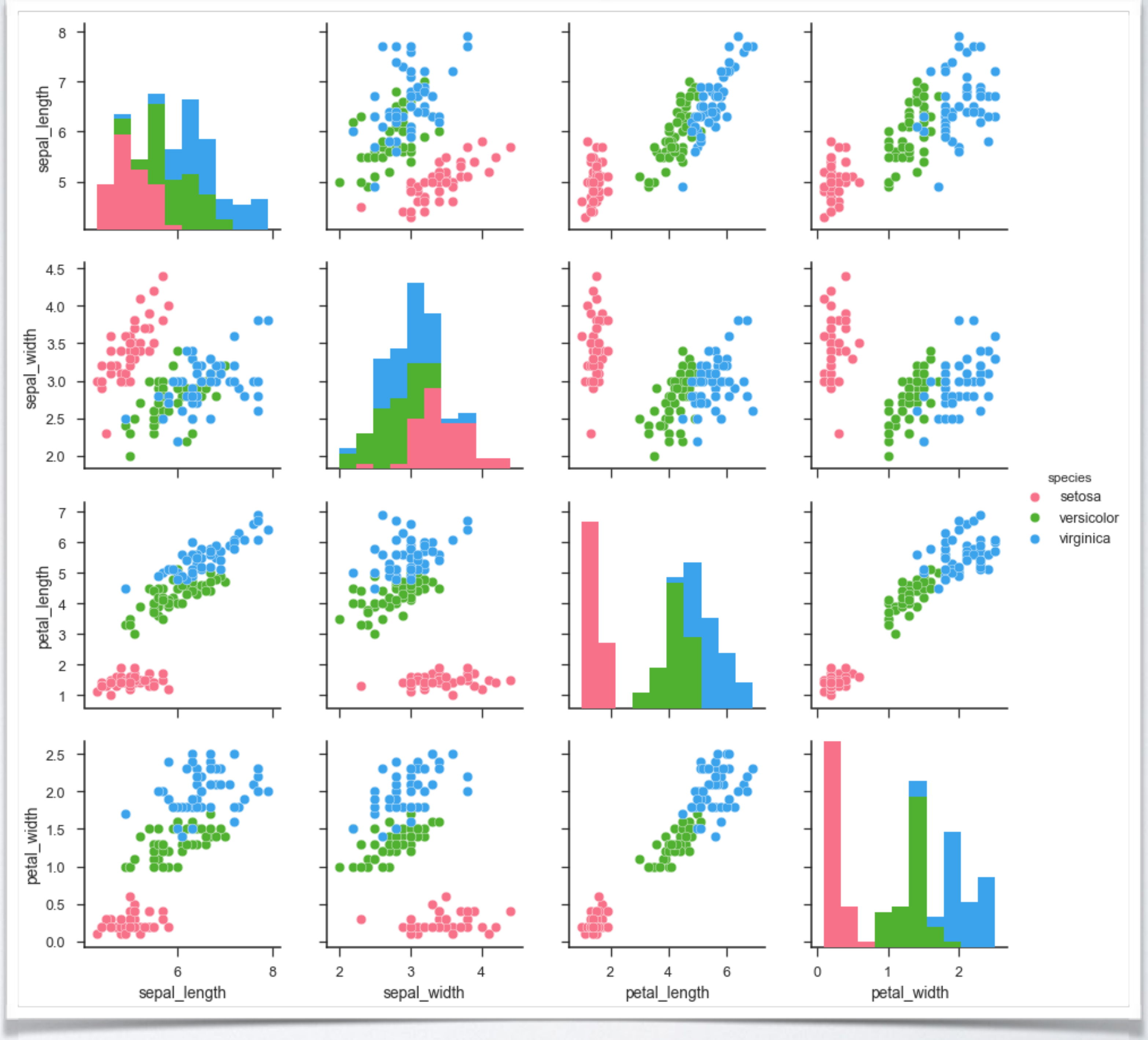
CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import numpy as np
from scipy.stats import kendalltau
import seaborn as sns
sns.set(style="ticks")

rs = np.random.RandomState(11)
x = rs.gamma(2, size=1000)
y = -.5 * x + rs.normal(size=1000)

sns.jointplot(x, y, kind="hex",
              stat_func=kendalltau,
              color="#4CB391")
```



CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import seaborn as sns
sns.set(style="ticks",
        color_codes=True)

iris = sns.load_dataset("iris")
sns.pairplot(iris,
             hue="species",
             palette="husl")
```



You will learn more about seaborn from **David Kirkby**!

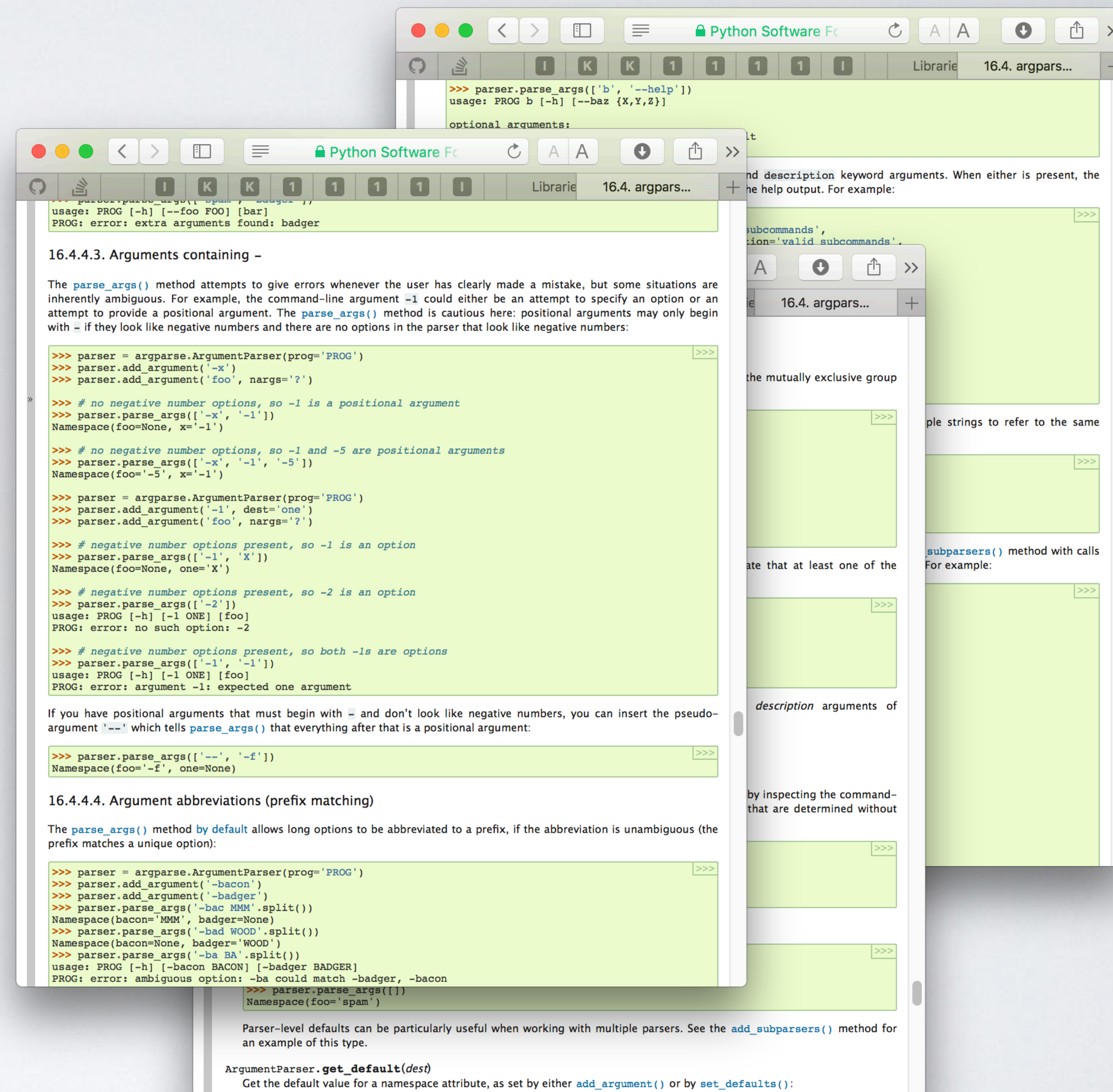
DOCOPT

creates beautiful command-line interfaces

by Vladimir Keleshev
https://github.com/docopt/docopt

ARGPARSE/OPTPARSE

Many classes and functions,
default values,
extensive documentation,
very hard to memorise
a basic setup.



DOCOPT

```
#!/usr/bin/env python
"""

Naval Fate.

Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate -h | --help
  naval_fate --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.


"""
from docopt import docopt
arguments = docopt(__doc__, version='Naval Fate 2.0')
```

DOCOPT

```
naval_fate ship Guardian move 10 50 --speed=20
```

```
arguments =
{
  "--drifting": false,
  "--help": false,
  "--moored": false,
  "--speed": "20",
  "--version": false,
  "<name>": [
    "Guardian"
  ],
  "<x>": "10",
  "<y>": "50",
  "mine": false,
  "move": true,
  "new": false,
  "remove": false,
  "set": false,
  "ship": true,
  "shoot": false
}
```

CLICK

a mature command line utility interface package

http://click.pocoo.org

CLICK

- Much more advanced compared to docopt
- The no.1 choice if you want to go crazy with command line utilities

```
import click

@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo('Hello %s!' % name)

if __name__ == '__main__':
    hello()
```

SO, WHAT NOW?

FINAL PERSONAL THOUGHTS

I spent a lot of time optimising Python code in the past years, here is a short summary of my personal experience.

- There were several attempts to make Python itself faster w.r.t. low level programming, none of them are satisfying (PyPy may have a future, but still doesn't fully support Python 3), many of them were abandoned
- Think twice (or more) before you bake Cython or any other static compilation into your project. The two language problem is real and it's hard to get it right. The performance gain is often disillusioning compared to the work, workarounds and "mess" one needs to deal with.
- Me and my lovely dev-team made the best experiences with numba
  - no clutter or double bookkeeping, no (static) compilation
  - minimal dependencies (basically only LLVMlite)
  - often orders of magnitudes faster than comparable low level algorithms utilising custom Cython class instances or ctypes
  - downside: super slow without numba…
- When it comes to high performance code using Python, you have to think in numpy arrays and cannot model your own datatypes like e.g. in C or C++ (structs, classes…)

MY RECEIPT FOR PERFORMANT PYTHON CODE

- **Avoid massive amounts of Python class instances**
  (e.g. don't create a class for a Point and then a list of 10 million points!)
- **Use numpy arrays for large homogenous data**
  (w.r.t. the "points" example above, create a 3xN numpy recarray instead, so you can access points.x, points.y and point.z. Subclass the array if you need some special functionality)
- **Vectorisation is a good idea (most of the time).**
  For basic operations, you most likely find a dedicated function in numpy or scipy.
- **Try to reuse already allocated memory** (allocations are expensive!)
- **Always profile first, before you do heavy optimisations!**
  "[…] premature optimization is the root of all evil." —D. Knuth

  Keep in mind, this doesn't mean that you sit down and hack together code, whatever works, this is not what Donald meant! Take care of the basic principles of performant code from the very beginning, otherwise you will have a hard time to refactor.
- **Do not reinvent the wheel.**
  You mostly find a lib which does what you need, better, faster and for no cost.

Ohne more thing…

AN EXAMPLE WHY IT'S SO HARD TO MAKE PYTHON FAST?

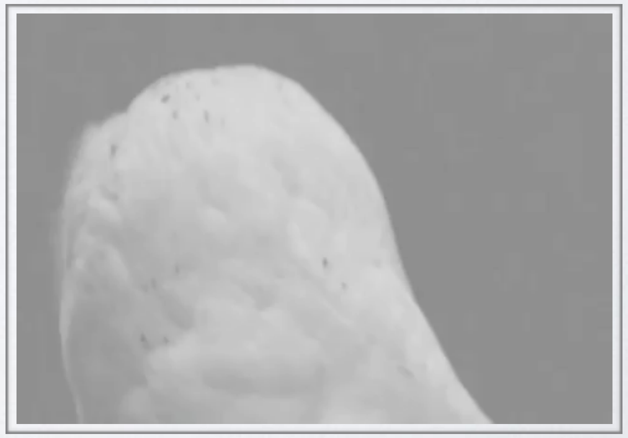
JUST A SIMPLE, BUT CRUCIAL ASPECT…

- Python lets you do anything.
- Here is a "pure" function, written in Python:

```python
def square(x):
    return float(x)**2
```

- Every decent compiler should now be able to optimise code using this function (repeated calls, tail recursion elimination, inlining, thread safety guarantees, etc.)

```python
import builtins
builtins.float = int
```



THANK YOU!

...also many thanks to Vincent and Jayesh,
and the whole organising committee!

Acknowledgement

- H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).