Follow-along instructions

- All examples will include links!
  - Grab the slides from http://github.com/jgerity/talks
- If you'd like to follow along on your own machine, I recommend Python 3.5+
  - 3.6 has a *really* good string feature, I recommend it a lot!

PHYSICS & ASTRONOMY
TEXAS A&M UNIVERSITY

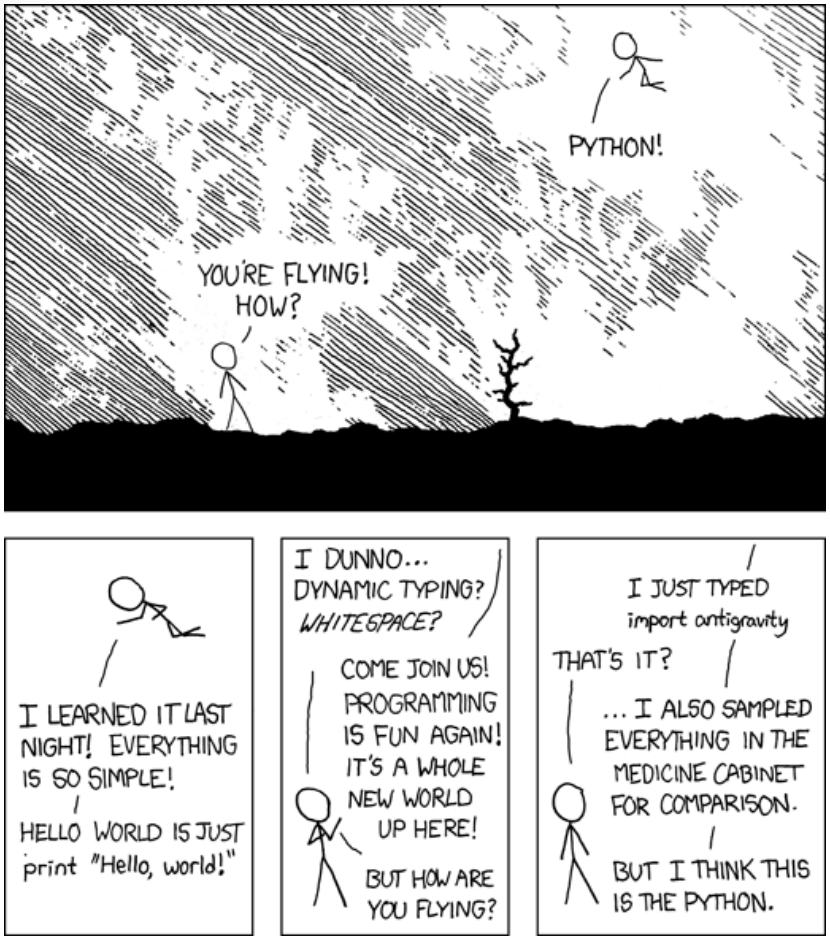
Introduction to Python

James Gerity

September 25, 2018

Slides available at http://www.github.com/jgerity/talks

Scope of this talk



Scope of this talk

- More things to say than will fit in this talk, see the list of notes and reading material online.
  - And ask questions!
- Assuming *some* programming familiarity
- Programming is most rewarding as an auto-didactic process!

Scope of this talk

- Python basics
  - Basic types, flow control, functions, classes, modules
- Python tools
  - Jupyter and other editors
- Python libraries
  - Primarily the SciPy stack

What is Python?

- Python is an **interpreted** programming language with **dynamic typing**

What is Python?

- Python is an **interpreted** programming language with **dynamic typing**
- **Interpreted** (roughly*) means instructions are “translated” from a high level to machine code
  - Python details are kinda complicated, but punchline: need to have a Python **interpreter** installed in order to run our programs!

* If you’re curious about CPython’s innards, check out this post series, which explains the whole process!
https://akaptur.com/blog/2013/11/15/introduction-to-the-python-interpreter/

What is Python?

- Python is an **interpreted** programming language with **dynamic typing**
- The **type** of an object must be known to check validity of operations, optimize, etc. In Python, types are tracked **dynamically**, i.e. at runtime.
  - Contrast with **static** typing in e.g. C/C++ (*declaring* type with `int, char`, etc.)

What is Python?

- An example in Python. What does this do?
  - `a+b`
- A: It depends on the types!
  - `1+1` gives 2
  - `'a'+'b'` gives `'ab'`
  - `[1,2]+[3]` gives `[1,2,3]`
  - `1+'a'` gives `TypeError`

What is Python?

- An example in Python. What does this do?
  - `a+b`
- Under the hood:
  - Ask `a` if it knows how* to add `b` to itself
  - If that fails, ask `b` if it knows how* to add `a`
  - If all else fails, give up and raise `TypeError`

* "knows how" here means "dunder" methods like `__add__()` on the type's class. You can learn more in section 3.3.8 of the Python docs: https://docs.python.org/3/reference/datamodel.html#emulating-numeric-types

A note on Python 2/3

- "*Python 2.x is legacy, Python 3.x is the present and future of the language*"
- Many libraries have already dropped support for 2.x, and official support ends in 2020. https://python3statement.org/
- All the cool new stuff is in 3, you might as well learn it now!

A note on Python 2/3

- A distinction worth considering:
  - Python 2:

```
>>> θ = 3.1415926/2
  File "<stdin>", line 1
    θ = 3.1415926/2
    ^
SyntaxError: invalid syntax
```

  - Python 3:

```
>>> θ = 3.1415926/2
>>> print(θ)
1.5707963
```

What is Python good for?

- Development quality of life
  - Fewer “hoops” to jump through, more legible code (most pseudocode is *almost* Python!)
- Portability
  - As long as the target has a Python interpreter, we can run Python there!
- Rich ecosystem
  - Great libraries exist for most common tasks

What does a Python program look like?

- The simplest Python programs fit on a single line.

  python –c 'print("Hello world!")'

- Less trivial programs are stored in files (usually with the *.py* extension) and delimited by whitespace.
  - No {} to define blocks of code in Python!

A second short program in Python

- **python loops_example.py**
- Online at https://repl.it/@SnoopJeDi/WigglyHatefulWebpage

Functions

- Because we often repeat the same task, it is convenient to write **functions**
  - Don't Repeat Yourself (DRY)
- In Python, we write:
  - `def funcname(foo, bar):`
  - 'funcname' is the name of our function, 'foo, bar' are the **arguments**

A third program in Python

- **python –i functions_example.py**
- Online at https://repl.it/@SnoopJeDi/TangibleElectricAddition

Classes

- We can also define our own objects using the `class` keyword, e.g.:

```
class Point():
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def magnitude(self):
    return (self.x**2 + self.y**2)**(.5)
```

What is Python *not* good for?

- In general, a Python program may be slower to execute than a similar program in another language (C/C++/Fortran).
  - How fast is "fast enough?" A few 100 ms of slowdown is usually worth it.
- Best of both worlds: fast libraries for tasks that must go fast, with "glue code" in Python

A word on speed

- Is it actually slow?
  - Often faster to just try, you'll be surprised!
- Where is it slow?
  - Use a profiler! Look at the `cProfile` module
- Why is it slow?
  - A working knowledge of asymptotic complexity ("big oh" notation) can help a lot

Numeric types

- `int` – integer with at least 32 bits precision
- `boolean` – truth value, True or False
- `None` – indicates no value
- `float` – a floating-point number (precision varies)
- `complex` – complex number with floating point components
  - Literal form is `A+Bj`

Other types in Python

- `object` – the base type of any object.
- `str` – a string like `"I'm a string!"`
- `list` – a *mutable* sequence of values, `[1,2,3]`
- `tuple` – an *immutable* sequence, `(1,2,3)`
- `dict` – a mapping of keys and values
- `set` – sets of *unique* immutable objects

Iterables, iterators, and sequences, oh my!

- Iterable –
  - *"An object capable of returning its members one at a time."*
- Iterator –
  - *"An object representing a stream of data."*
  - More helpful: the object that **does the iterating**!
- Sequence
  - *"An iterable which supports access using integer indices…and [that] defines a…method that returns the length of the sequence."*

Exploring these types in REPL

- Let's explore each of these types using the Python interpreter's REPL (Read, Eval, Print Loop)
  - **python basictypes_example.py**
  - https://repl.it/@SnoopJeDi/CharmingSoftRuntimes

Other keywords

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

- `is` does **NOT** mean equality! It tests that two objects are exactly the same object!
  - Try `[1,2,3] is [1,2,3]`

Managing your Python environment

- You will likely use libraries like numpy/scipy, astropy, matplotlib, pandas, hdf5.
- **pip** and **Anaconda** are two excellent tools for managing what's installed.
- E.g. to install numpy: '**pip install numpy**'
- **Anaconda** allows you to create separate collections of packages, for version conflicts, etc.

Three more Python Programs

- Determining last Friday's date
  - https://git.io/v6bXt
- Reading data from a text file
  - https://git.io/viqsg
- Brownian motion
  - http://scipy-cookbook.readthedocs.io/items/BrownianMotion.html

IPython/Jupyter interface

- A REPL is helpful when you want to type a handful of lines once, but not useful for long sessions
- Jupyter notebooks allow the user to define “cells” of Python input which produce output, and can be run/edited arbitrarily.
  - Think Mathematica without the symbolic math


And now for something completely different

The Python Standard Library

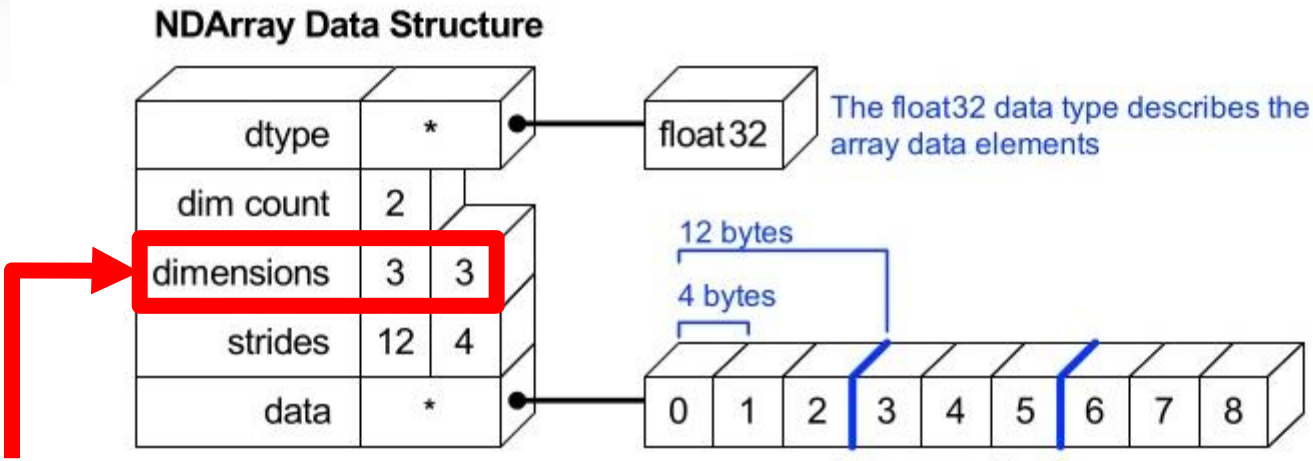
- Some fairly self-explanatory ones:
  - `random, math, time, datetime, argparse`
- Data wrangling:
  - `csv, xml, json, re`
- Quality of life:
  - `itertools, io, logging, os, pathlib, sys, glob, pickle, collections`

Numpy

- Numpy implements a basic type called the `ndarray` that is useful for computation
  - As the name implies, an array with N dimensions
    - Uniform data type, “gridlike” structure.
  - In memory, a conventional array (1D)
  - In use, the array knows about the axes, so can jump directly to the right data as needed

Numpy

Array Data Structure



Also known as the **shape**

Python View :

0	1	2
3	4	5
6	7	8

Slide originally by Enthought, used under fair use

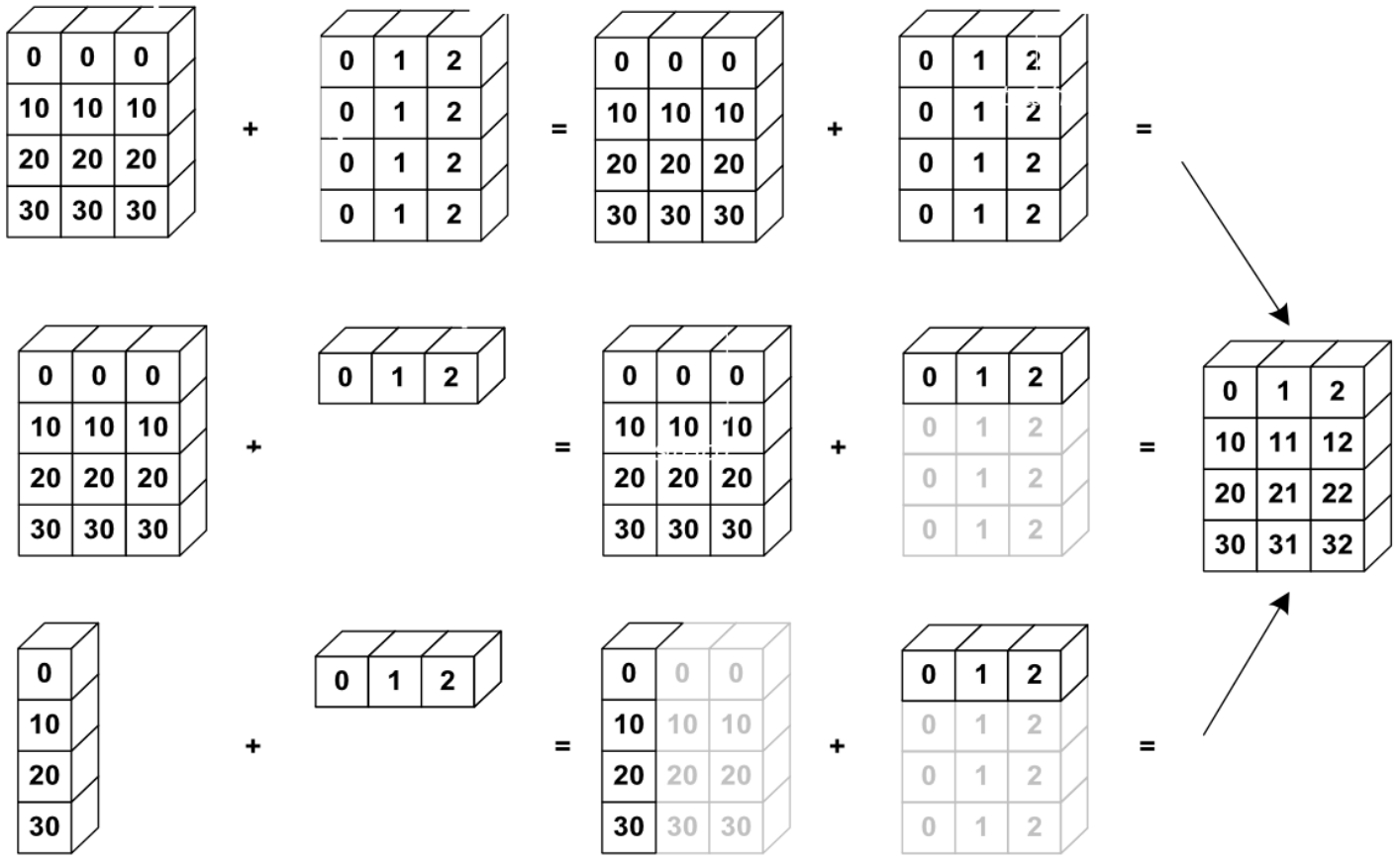
Numpy

- Numpy's main strength is that it is implemented in C and Fortran
  - Static typing and direct memory access → *fast* array operations
- For a crash-course intro to numpy, see numpy/numpy_intro.ipynb

Numpy broadcasting

- Often we have arrays of different shapes, and wish to ‘copy’ data across one or more arrays for an arithmetic operation
  - For example: adding a series of numbers to every row of a matrix
- If we actually copy the data, this might be expensive in terms of memory!

Numpy broadcasting



Numpy broadcasting

- Broadcasting rule: trailing axes must either be the same size, or one of the operands must have size 1
- An extremely useful article about how broadcasting works, with examples: http://scipy.github.io/old-wiki/pages/EricsBroadcastingDoc

SciPy

- “SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.”

SciPy subpackages

- tl;dr – there are a ton of them!
- These are the ones we'll talk about.

SciPy Organization

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

scipy.integrate

- The `integrate` package has useful tools for simple numerical integration
- See scipy/numerical_integration.ipynb

scipy.optimize

- Multiple minimization algorithms
- Fitting example from cookbook:
  see scipy/fitting_data.ipynb
- Optimization is a very tricky subject, so be careful!
  - …but for common problems, this will do well
- Non-linear solvers

scipy.signal

- A variety of tools for splining and filtering of 1D/2D data
- Chirp generation
- Low/high/bandpass filters, see scipy/FIR_filter.ipynb
- Also of interest is `scipy.fftpack`, which has (potentially slow!) FFT / DCT / DST transform implementations.

scipy.stats

- Large number of available distributions for sampling random variables
- Some statistical tests, methods
- http://www.scipy-lectures.org/packages/statistics

SciPy library

- Tons of examples in the cookbook:
  - http://scipy-cookbook.readthedocs.io/
- Full documentation:
  - http://docs.scipy.org/doc/scipy-0.18.0/reference/

Pandas

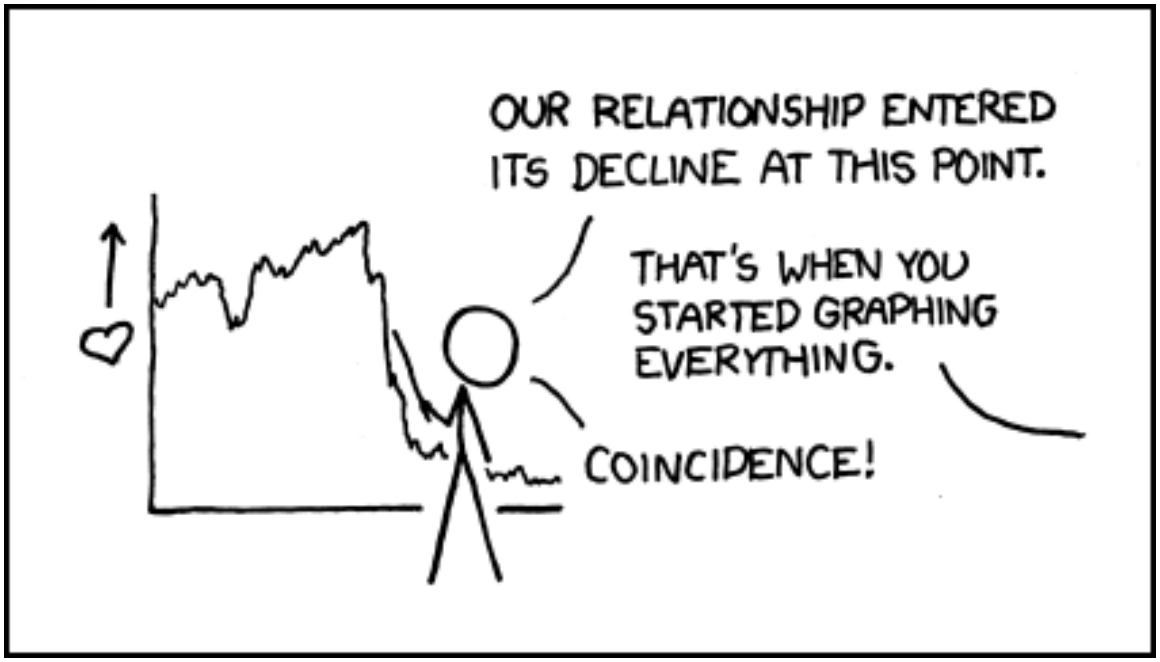
- Basic structure: DataFrame
  - Columns with names, expressive queries
- Like R, but in Python!
- Tutorial given at SciPy 2015
- 10 minutes to pandas: http://pandas.pydata.org/pandas-docs/version/0.18.1/10min.html

I'M NOT AN ASTRONOMER
¯_(ツ)_/¯

Astropy

- Astropy consists of a core package with common functionality:
  - `units` and `constants`
  - Juggling coordinate systems with multiple frames, units, etc. Seems to be mostly done with `SkyCoord`
  - Working with FITS files (`astropy.io.fits`)
  - Fitting/modeling routines (looks WIP)

Plotting in Python



A brief rant on colormaps

- Colormap matters! A poor choice can…
  - …create perceptual artifacts in data
  - …be a nightmare to print (→ grayscale conv!)
  - …screw with colorblind readers
  - …most importantly, be ugly

A “bad” choice: Jet

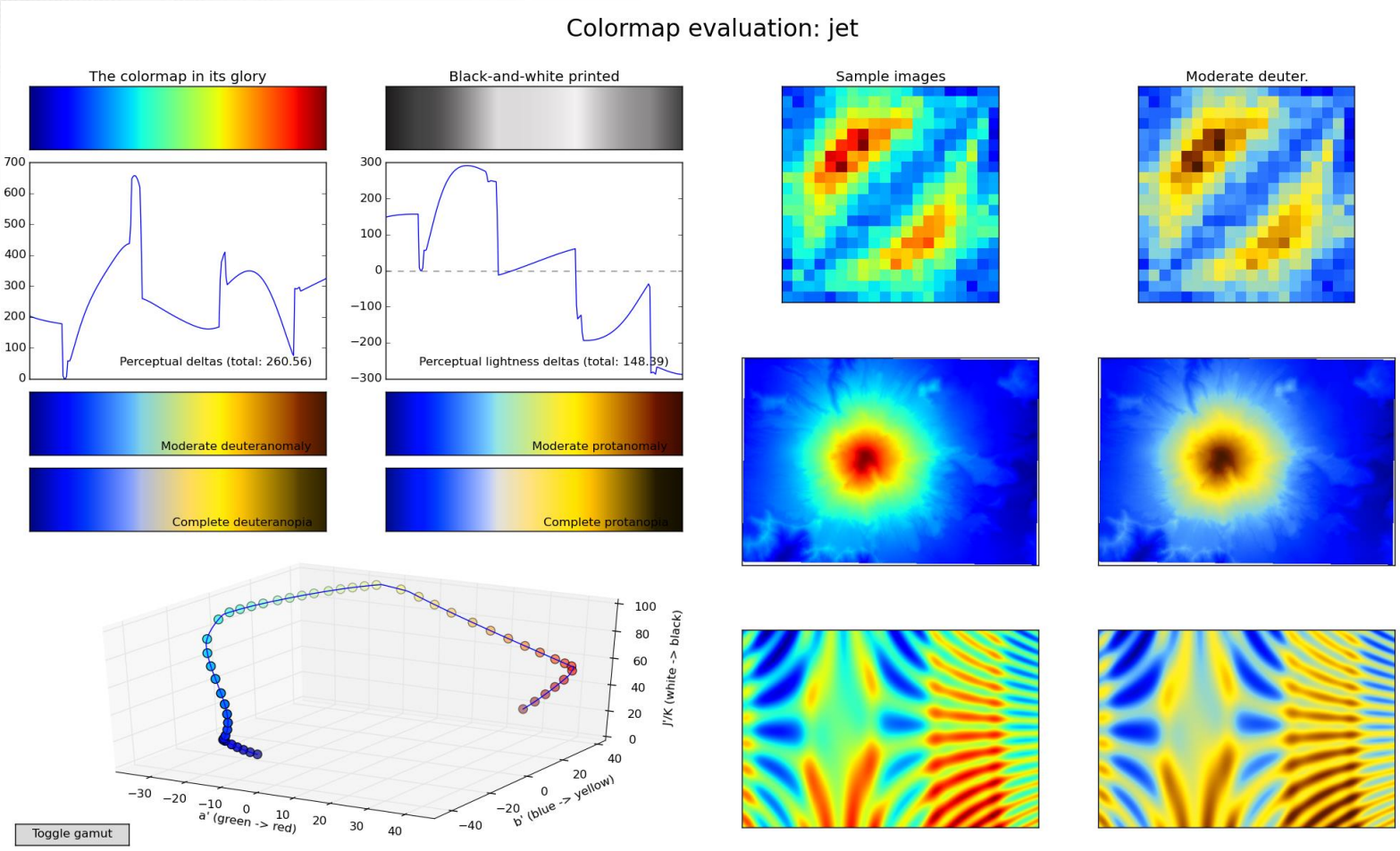


Figure from https://bids.github.io/colormap/

A “good” choice: Viridis

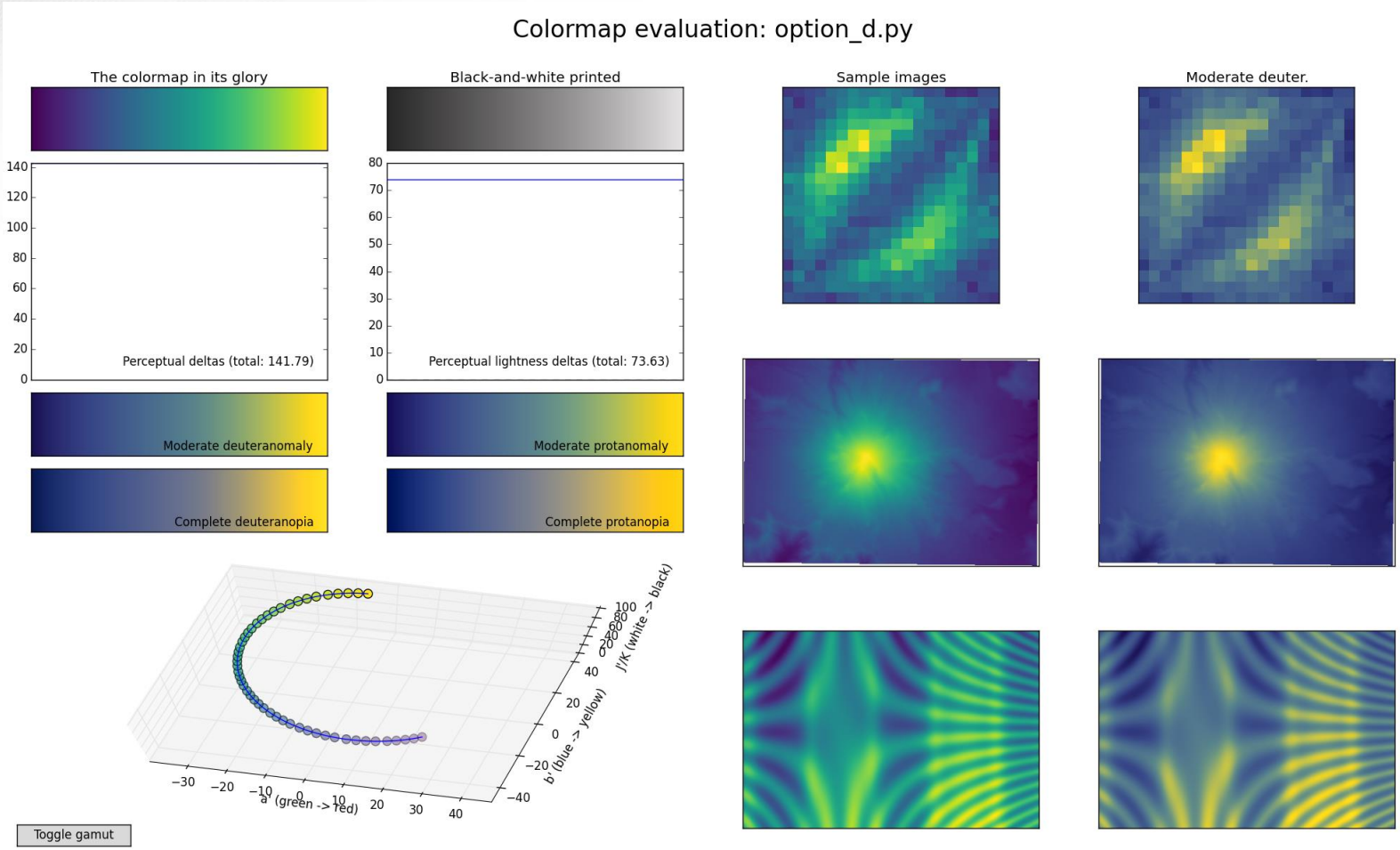


Figure from https://bids.github.io/colormap/

matplotlib

- The principal visualization library for Python is matplotlib
  - matplotlib.pyplot: MATLAB-like plotting framework and probably already your BFF
  - matplotlib.cm: predefined colormaps and tools for making your own
  - …and lots of minutiae

Anatomy of a matplotlib figure

- …in the form of a matplotlib figure!
  - See matplotlib/anatomy_of_matplotlib_figure.ipynb

matplotlib objects

- Although the `pyplot` interface is useful, it is often more reliable to interact directly with the `matplotlib` API.
  - An example of a very creative plot that uses the GridSpec system to produce subplots with different sizes: http://gregj.net/JHEPC/

Writing code that doesn't suck

- Python makes it easier to write code that is simple to read
  - Enforced indentation and lack of braces helps
- It doesn't make writing bad code impossible!
- Python's PEP 8 is long, but has good guidelines
  - https://www.python.org/dev/peps/pep-0008/

Useful resources

- **CodeAcademy**
  https://www.codecademy.com/learn/python
- **Scipy lectures**
  https://scipy-lectures.github.io/
- A long list of other resources available on GitHub, along with this presentation:
  https://github.com/jgerity/talks

The Zen of Python

- When in doubt…
  - **import this**

Thank you!