# Python for Data Scientists

Patrick Varilly - dataminded
Thu 17 Sep 2015

We're hiring data scientists!!
Come talk to me
in the pizza break

# Python for Data Sci

Patrick Varilly - dataminded
Thu 17 Sep 2015

python™

# Goals for today

- For people who've never seen Python:

  **Learn enough Python to want more**

- For people who use a bit of Python:

  **Learn new tricks useful in data science**

- For people who know Python inside out:

  **Showcase of data-science libraries**

# Overview

- About me
- Quick history of Python
- Whirlwind tour of ecosystem & language
- Pros and cons for Python in data science
- ***Pizza break***
- Showcase of key data science packages (NumPy, SciPy, …)
- Efficient Python
- Where to learn more

# About Me

- Previous life: trained as physicist, researcher in theoretical chemistry
- Started using Python in 2012 for research projects, contributed cKDTrees to SciPy.
- Changed careers in Sep 2013: online advertising / big data until Jan 2015, now data scientist / engineer at Data Minded. Used Python for data wrangling, data pipeline management and dynamic query generation.
- Coached Scalable Machine Learning MOOC (pySpark) at DIHub

patrick.varilly@dataminded.be
http://www.dataminded.be
http://github.com/patvarilly

# (Abridged) History of Python


Guido van Rossum
Python's Benevolent Dictator For Life (BDFL)

- Created in 1989 by Guido van Rossum at CWI in Amsterdam.
- Currently, two major, **incompatible** branches of Python: Python 2 and Python 3.
  - forked in 2008 to fix how strings & unicode are handled.
  - main development in Python 3 (now: 3.4.3)
  - Python 2 is frozen and supported until 2020. Good features from Python 3 are back-ported.
  - most scientific computing & data science written for Python 2 (but can use them in Python 3)
  - **we'll stick to Python 2 today**

**Much more in depth: http://python-history.blogspot.com**

# (Abridged) History of Python

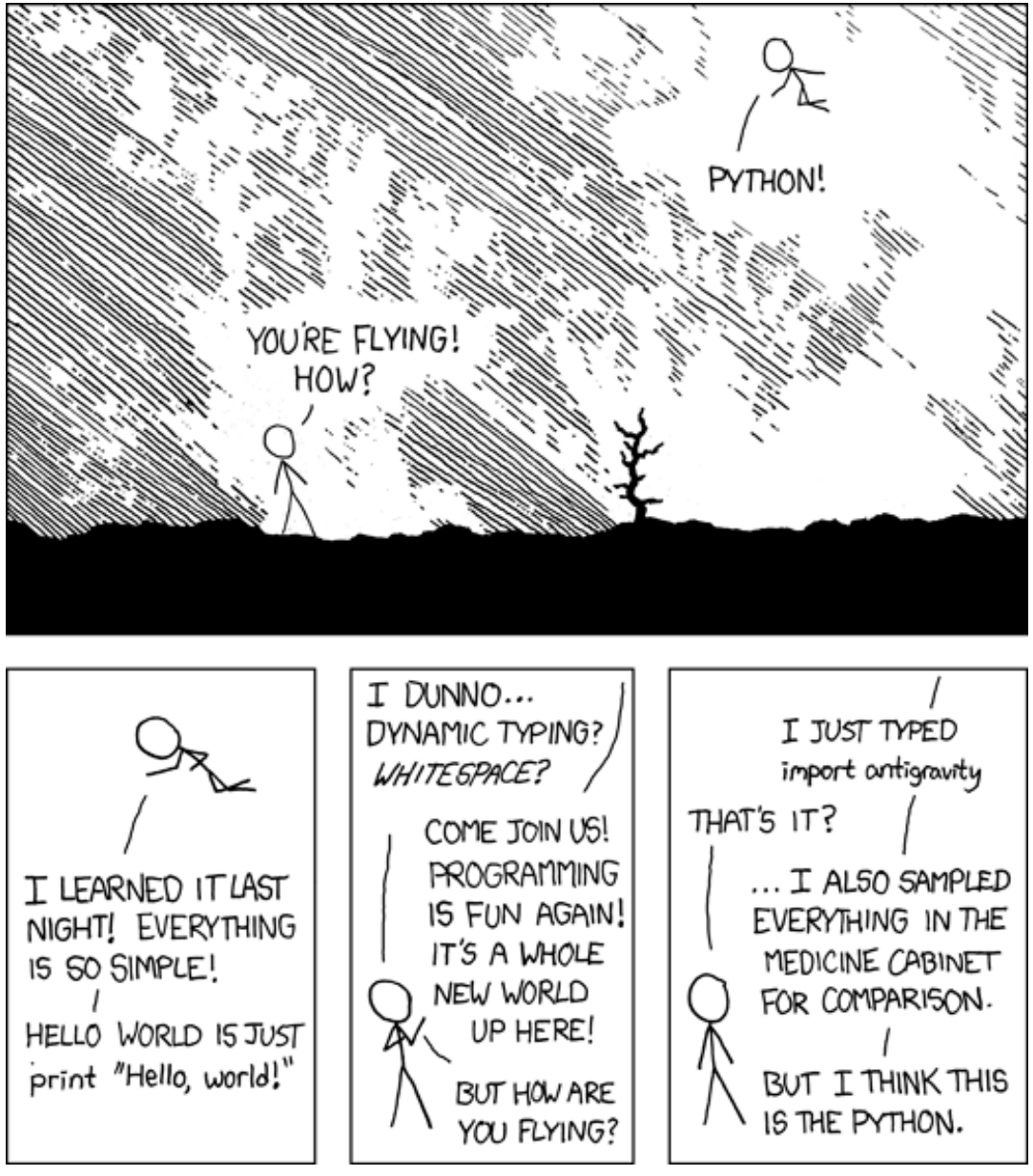
- Vast numbers of libraries & tools make Python **useful**:
  - 1995 – NumPy: fast vector and matrix algebra
  - 2001 – SciPy: optimization, integration, FFT, ODEs, …
  - 2007 – Matplotlib: MATLAB-style plotting
  - 2007 – Scikit-Learn: machine learning
  - 2007 – Cython: static typing for fast, compiled code
  - 2007 – IPython: much-improved REPL, notebooks for sharing & presenting
  - 2008 – Pandas: data frames for data wrangling
  - 2012 – Numba: near-transparent JIT compilation
  - 2012 – pySpark: big data / distributed computing

# Typical Python use cases

- Web development — e.g., Google's first crawler, youtube.com, bit.ly, SurveyMonkey, OpenERP, …
- System management — yum for packages, ansible for infrastructure
- In-app scripting / glue code – Blender, Industrial Light & Magic
- Science – e.g., data analysis at LHC, astrophysics at ALMA
- Engineering — e.g., systems control at Siemens
- Finance — e.g., risk management at Swisscom
- Outreach – e.g., One Laptop Per Child, MIT's Intro to Programming

Sources: http://brochure.getpython.info, https://www.python.org/about/success, https://docs.python.org/3/faq/general.html

# Atypical Python use cases



# Zen of Python

**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!

https://www.python.org/dev/peps/pep-0020/

# How to write and run Python

- Text Editors + Scripts
- Python Command line
- IPython Command line
- IPython Notebook
- PyCharm IDE

# Whirlwind tutorial of Python

# Pros of Python in data science

- Quick to write, interactive: encourages experimentation
- Huge library of packages: interact with any data source, lots of algorithm available with little effort
- Full & sensible programming language: not crippled, scales well (compare to SQL or Matlab)
- Emphasis on communication with others: “executable pseudo-code” (Zen of Python), IPython notebook

# Cons of Python in data science

- Interpreted: errors show up mostly at runtime, not compile time
- Naïve, pure Python can be very slow (300-500x slower than C)
- Data usually must fit in memory (unless you use pySpark)
- R probably has more stats & machine learning libraries
- Python 2 vs Python 3 split still not fully resolved