Computational Modeling for Economists

An Introduction using Python

A QuantEcon–RSE Collaboration

February 2020

Introduction

Lecturers

- Matt McKay
- John Stachurski

Plan

- Review introductory QuantEcon lectures

Thanks

- Alfred P. Sloan Foundation, Research School of Economics

Downloads / Installation

Install Python + Scientific Libs

- Install Anaconda from `https://www.anaconda.com/downloads`
  - Select latest Python version (3.7)
  - For your OS!
- Not plain vanilla Python

Remote options

- `https://colab.research.google.com`
- etc.

Prereqs / Aims / Outcomes

Assumptions:

- econ/computer/maths/stats literate
- no familiarity with Python

Aims:

- Overview of scientific computing and Python
- Review some simple economic models
- Show how to solve such models with Python
- Prep for remainder of the course

Background — Language Types

Proprietary

- Excel
- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Language Types

Low level

- C/C++
- Fortran
- Java

High level

- Python
- Ruby
- Javascript

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

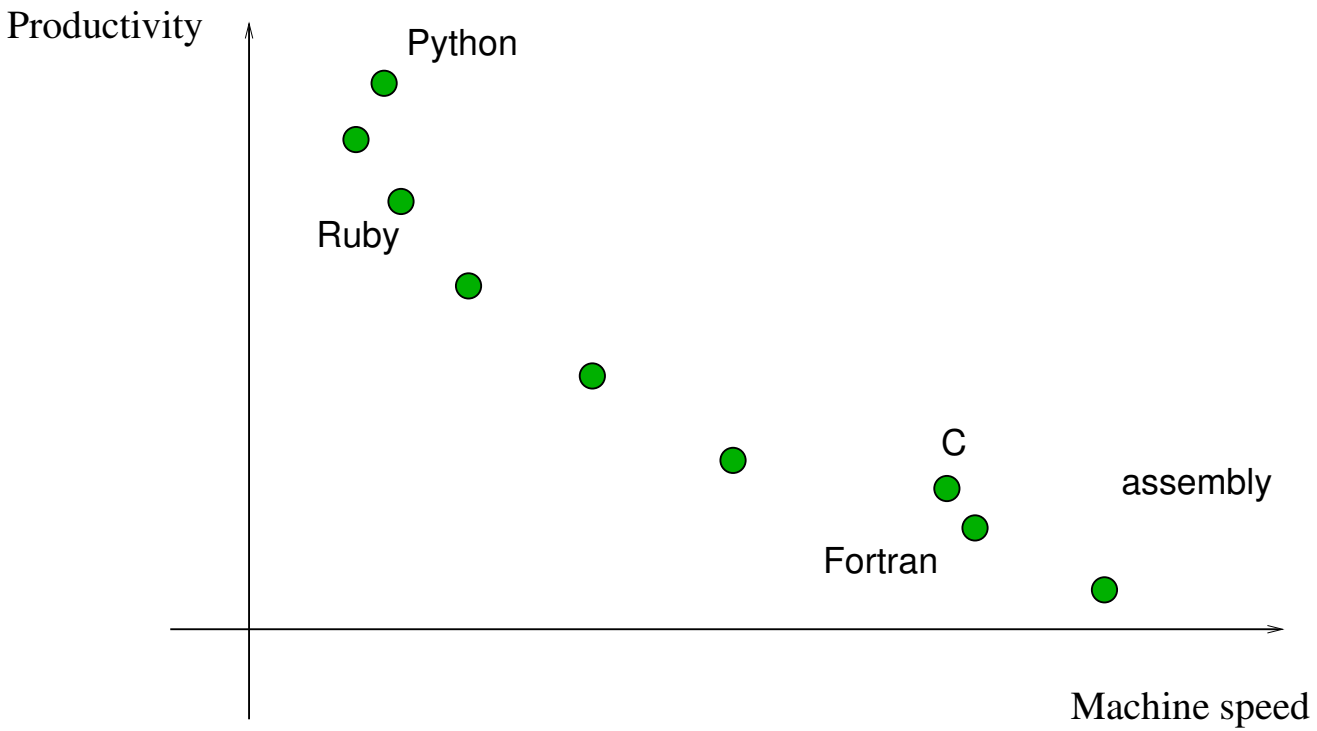
```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

High level languages give us abstraction, automation, etc.

Example. Reading from a file in Python

```
data_file = open("data.txt")
for line in data_file:
    print(line.capitalize())
data_file.close()
```

Trade-Offs

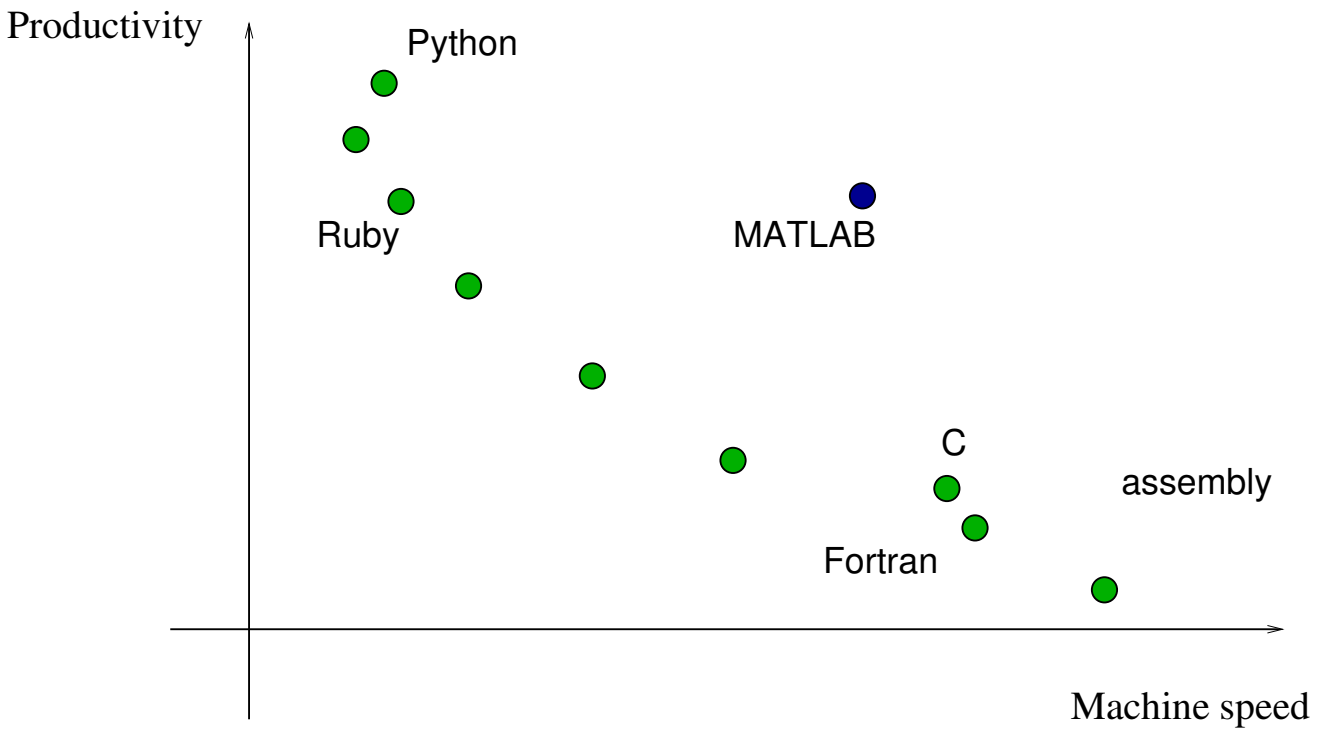

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

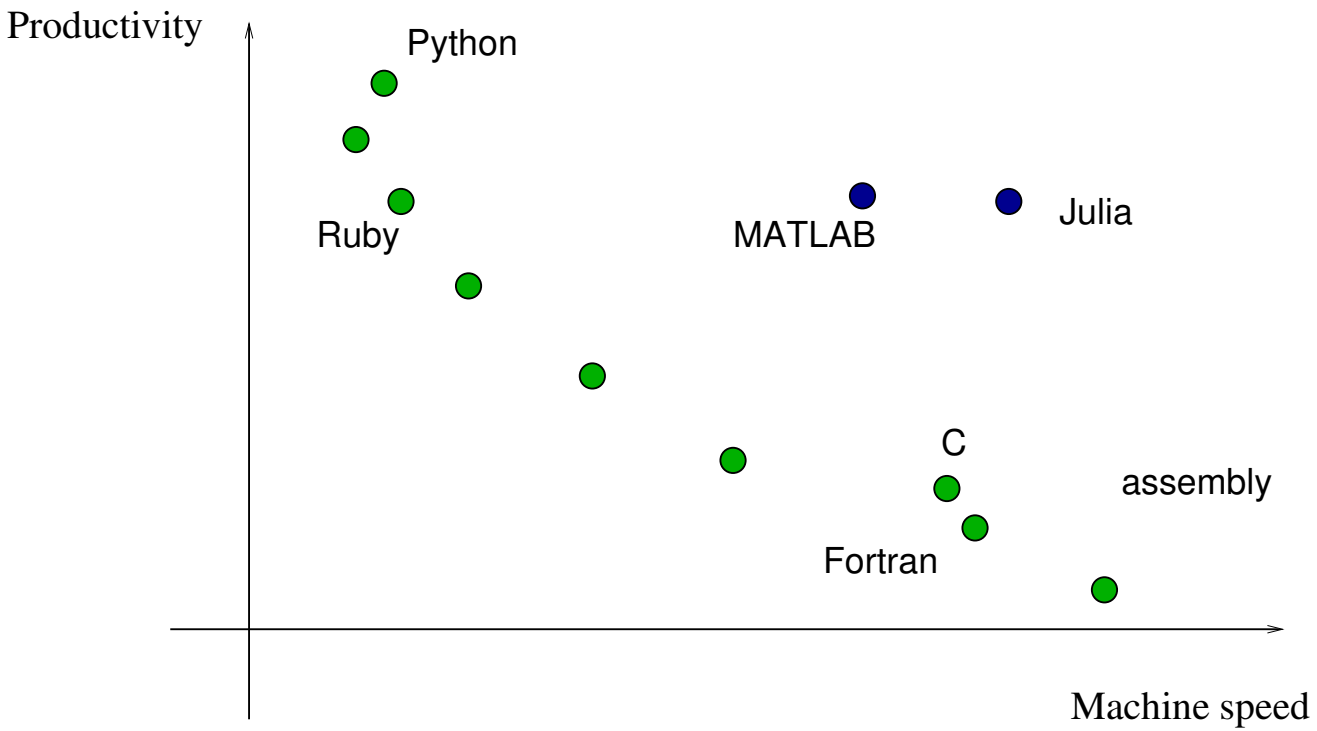
Requirements

- Productive — easy to read, write, debug, explore
- Fast computations

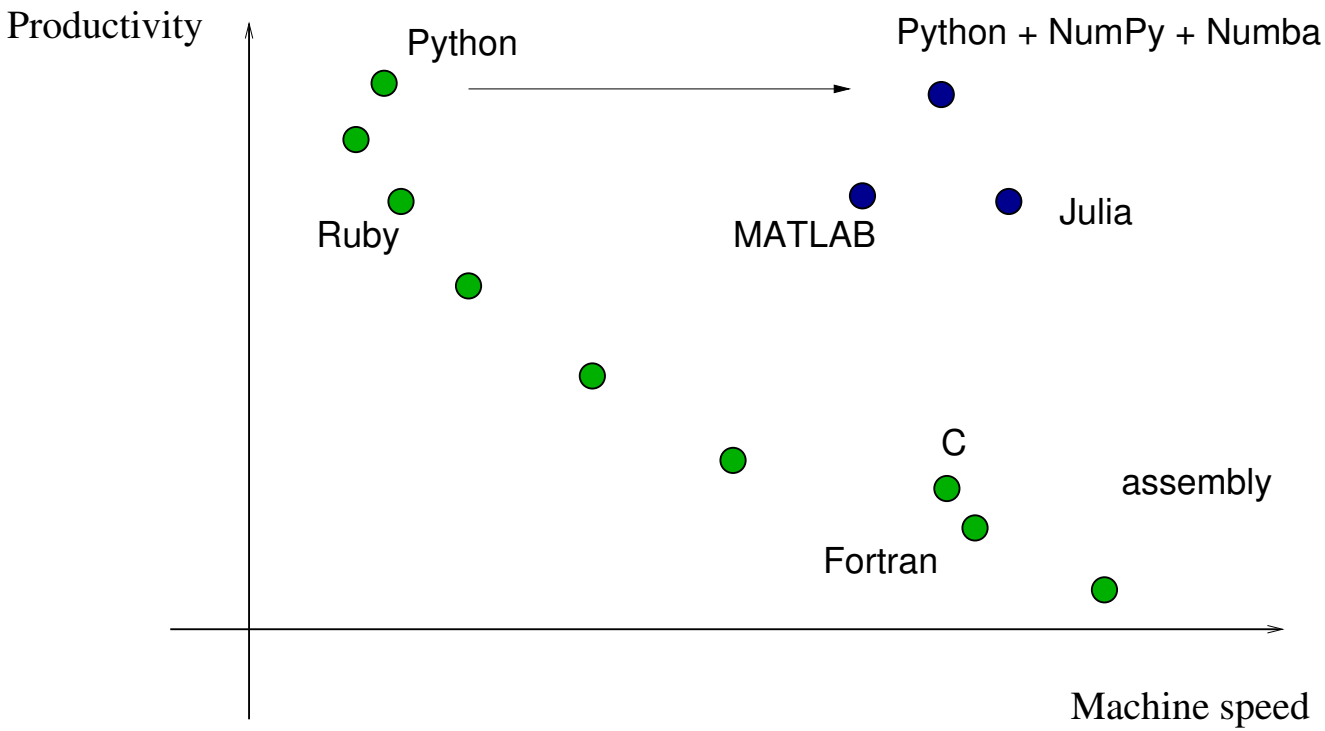
Trade-Offs


Productivity
Python
Ruby
MATLAB
C
assembly
Fortran
Machine speed

Trade-Offs

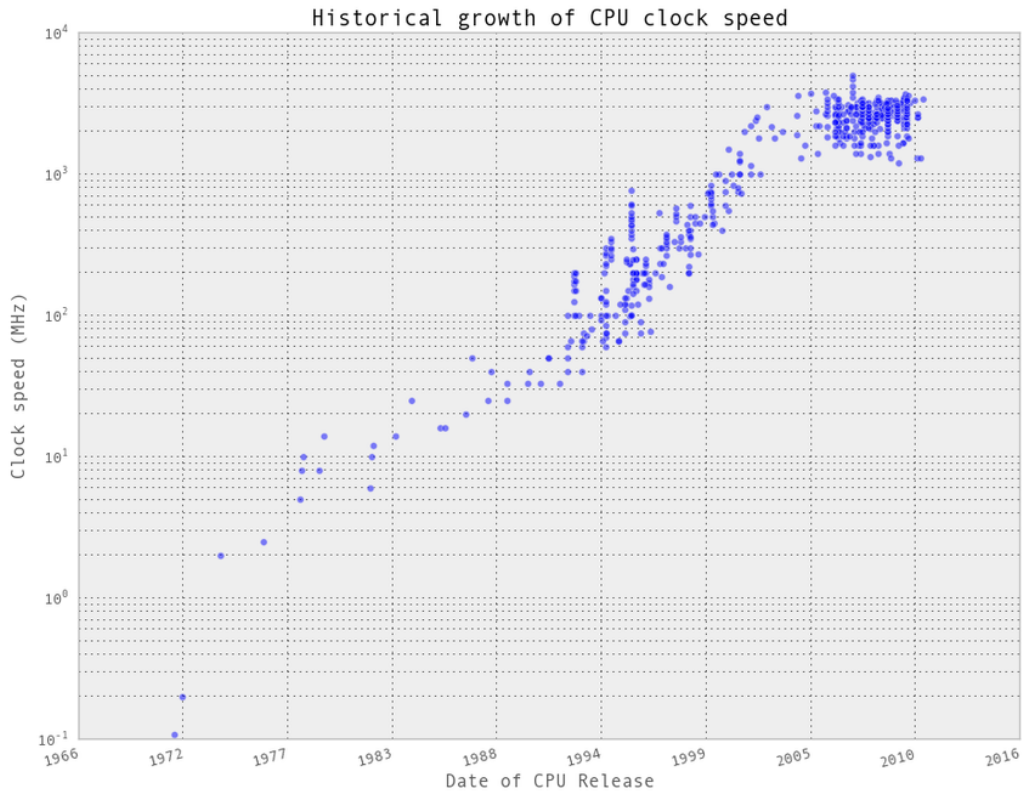

Productivity
Python
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Trade-Offs

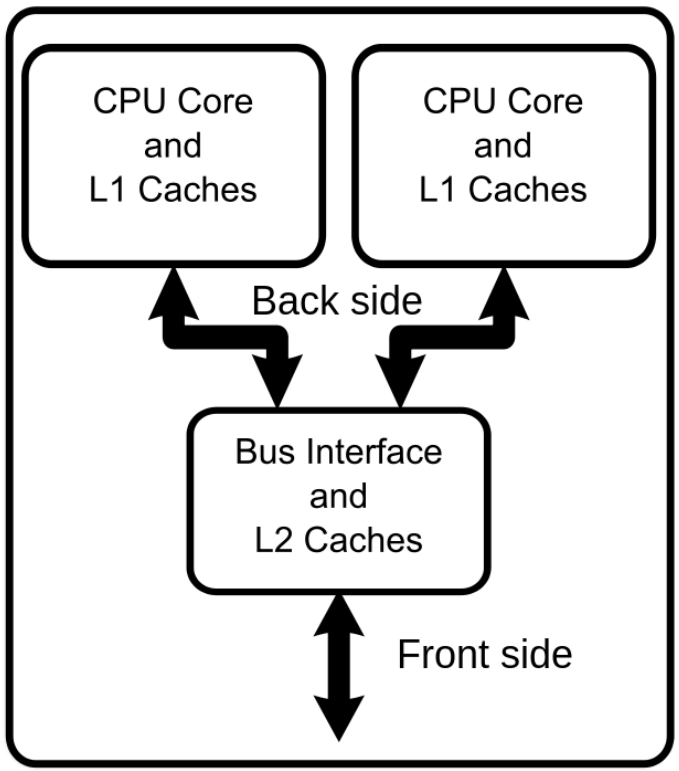

Productivity
Python
Python + NumPy + Numba
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Trend 1: Parallelization

CPU frequency (clock speed) growth is slowing

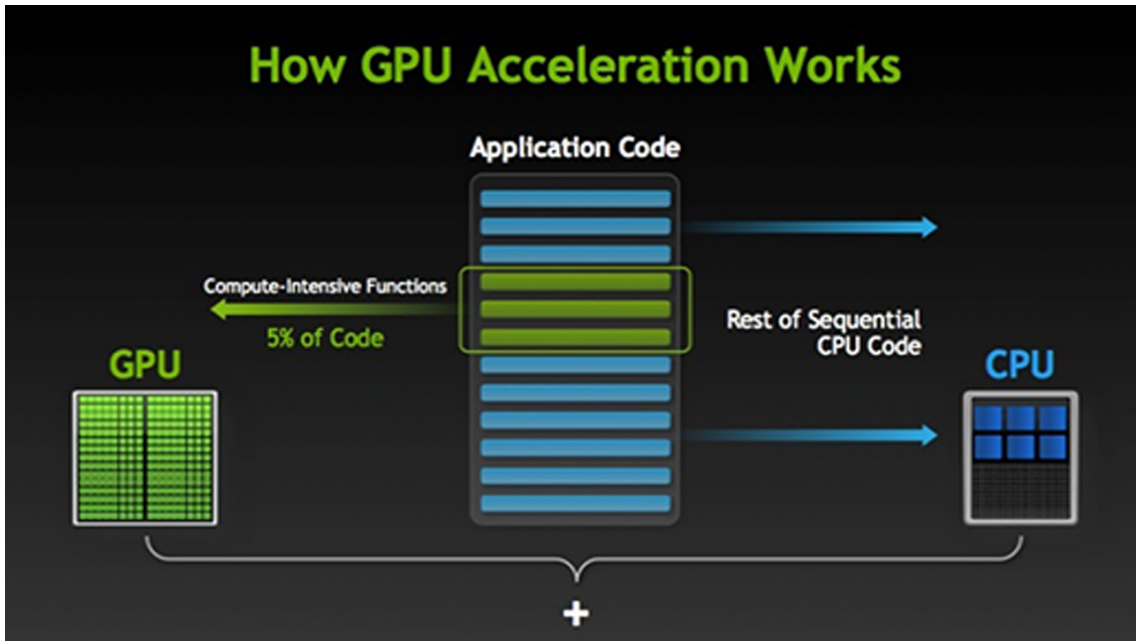


Chip makers have responded by developing multi-core processors



Source: Wikipedia

GPUs / ASICs are also becoming increasingly important



Applications: machine learning, deep learning, etc.

Trend 2: Distributed Computing

Advantages:

- run code on big machines we don't have to buy
- customized execution environments
- circumvent internal IT departments

Options:

- University machines
- AWS
- Google Colab, etc.

Why Python?

- Easy to learn, well designed
- Massive scientific ecosystem
- Open source
- Huge demand for tech-savvy Python programmers

Scientific Computing

Python has strong tools in vectorization / JIT compilation / parallelization / visualization / etc.

Examples:

- SciPy, NumPy, Matplotlib, pandas
- Numba (JIT compilation, multithreading)
- Tensorflow, PyTorch (machine learning, AI)
- JAX, NetworkX, etc., etc.

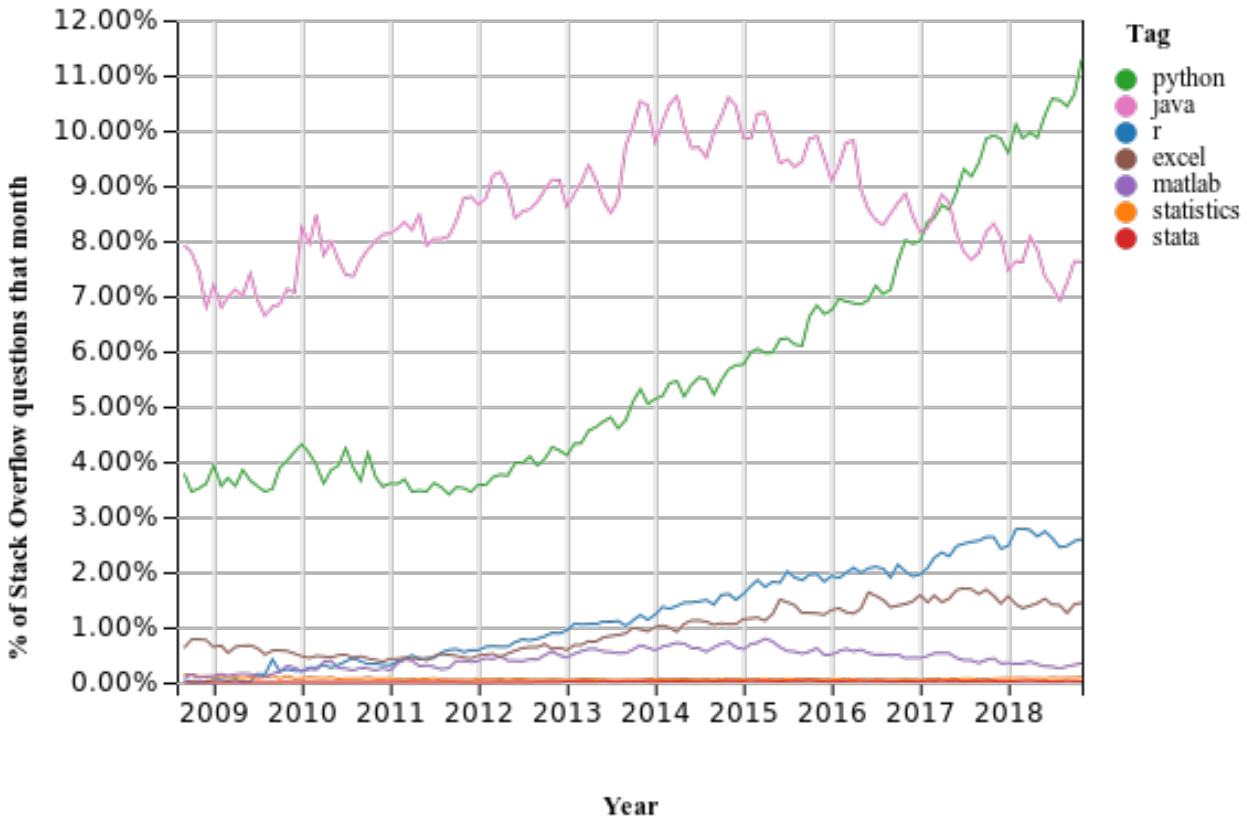
Python is convenient because it covers so many bases

- web dev, databases, system admin, GUIs

Chris Wiggins, Chief Data Scientist at The New York Times:

Python has gotten sufficiently weapons grade that we don't descend into R anymore. Sorry, R people. I used to be one of you but we no longer descend into R.

As a result of these advantages:



Using Jupyter Notebooks

- Dashboard and notebooks
- Modal editing
- Arithmetic
- Strings
- Variables (including unicode)
- Built in functions
- Getting help
- Repetition: Introduction to loops
- Documentation and rich text (markdown, LaTeX)