Computational Economics with Python

Copenhagen Summer School
May 2018

John Stachurski

Lecture 1

Set up and Resources

Presented jointly with Natasha Watkins

Homepage:

- https:
  //quantecon.org/copenhagen-summer-school-2018

Steps

1. Install Anaconda Python
2. Download files from GitHub repo

Assumptions / Prerequisites

- Coding experience assumed

- But no Python required

Structure

Lecture 1: Introduction and Foundations

- Why Python?

- Environment (Jupyter / Anaconda / IPython)

- Python syntax and semantics

- Simple examples

Lecture 2: Standard Scientific Libraries

- NumPy

- SciPy

- Matplotlib

- Pandas

Lecture 3: JIT Compilation and Parallelization

- Numba

- Parallel execution

- Cloud computing

- Inventory dynamics

- Exercise: Option pricing

Lecture 4: Applications

- Wealth dynamics and inequality

- Optimal stopping (job search)

- Optimal savings

Programming Background — Software

A common classification:

- **low** level languages (assembly, C, Fortran)
- **high** level languages (Python, Ruby, Rust, Nim)

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

- interactivity

- readability, etc.

Example. Reading from a file in Python

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

Jane Street on readability:

> *There is no faster way for a trading firm to destroy itself than to deploy a piece of trading software that makes a bad decision over and over in a tight loop.*

> *Part of Jane Street's reaction to these technological risks was to put a very strong focus on building software that was easily understood—software that was readable.*
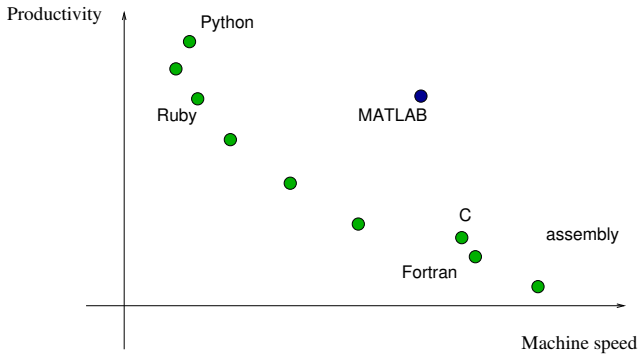
> – Yaron Minsky, Jane Street

Trade-Offs
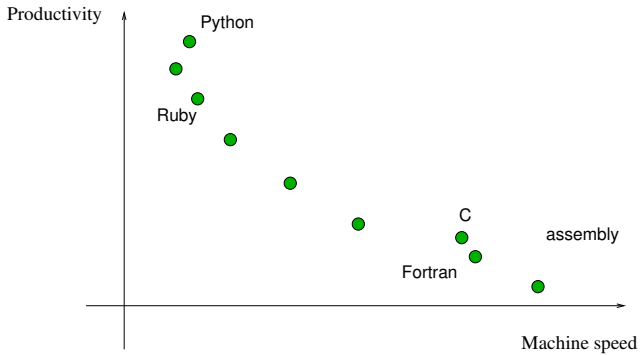
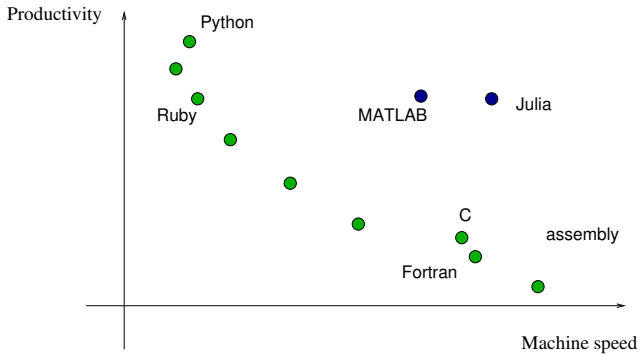# But what about scientific computing?

Requirements

- <u>Productive</u> — easy to read, write, debug, explore
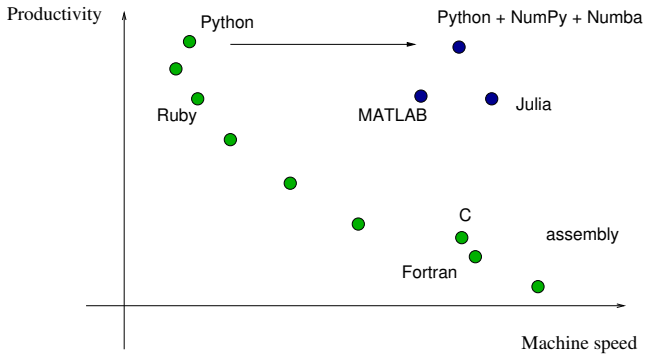
- <u>Fast</u> computations

# Trade-Offs

Trade-Offs

Trade-Offs

Summary: Why Python for Computational Economics?

- Elegant high level programming language

- High quality scientific libraries (NumPy, SciPy, pandas, Matplotlib)

- JIT compilation and parallelization

- Modern machine learning libraries (Tensorflow API, Scikit Learn)

- Vast array of free libraries in other fields (web, networks, natural language processing, etc.)

- Positive spillovers from popularity (Stack Overflow)
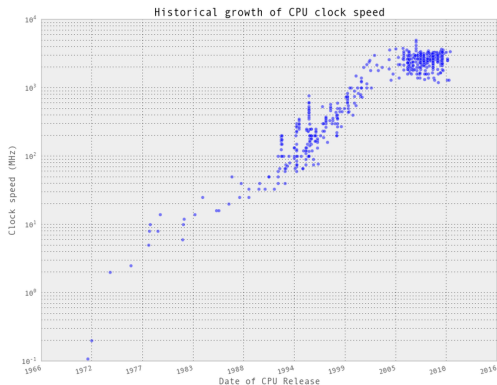
Python vs other popular high level languages
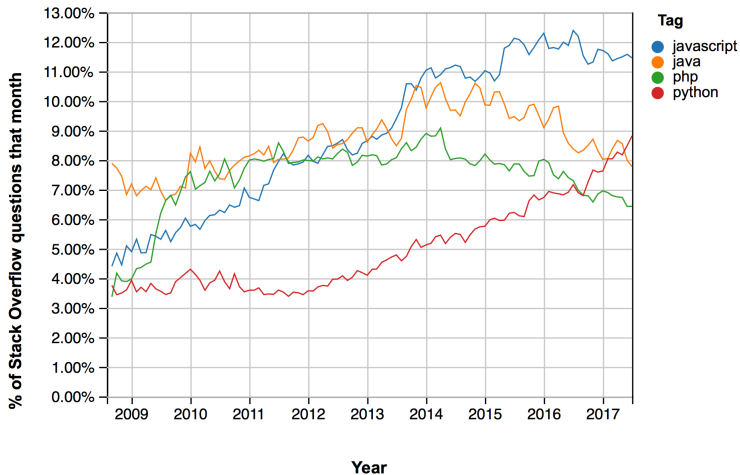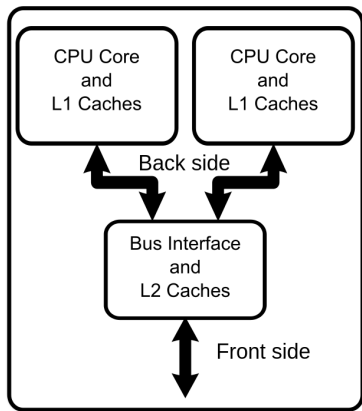
Disadvantages of Python

- Limited implementation of traditional econometric methods / tests

- Traditional scientific libraries are not JIT compiled

Programming Background — Hardware

CPU frequency (clock speed) growth is slowing

Historical growth of CPU clock speed

Chip makers have responded by developing multi-core processors

Source: Wikipedia

Exploiting multiple cores / threads is nontrivial

Sometimes we need to redesign algorithms

Sometimes we can use tools that **automate** exploitation of multiple cores

Distributed/Cloud Computing

Advantages: run computationally intensive code on big machines we didn't have to buy

Options:

- University machines
- AWS and other commercial services

Jupyter notebooks

A browser based interface to Python / Julia / R / etc.

Can be opened through `Anaconda navigator`

Or via a terminal:

Step 1: Open a terminal

- on Windows, use `Anaconda Command Prompt`

Step 2: type `jupyter notebook`

- opening a notebook

- executing code

- edit / command mode

- everything's an object (lists, strings)

- installing `quantecon`

- getting help

- introspection

- math and rich text

- Jupyter lab

Notebooks

- `day1/python_by_example.ipynb`

- `day1/python_essentials.ipynb`

- `day1/python_foundations.ipynb`