Computational Economics with Python

Tinbergen June 2018

John Stachurski and Natasha Watkins

Lecture 1

Set up and Resources

Follow the instructions here

- https://github.com/jstac/tinbergen_mini_course

Steps

1. Install Anaconda Python
2. Update Numba via `conda install numba=0.38`
3. Download files from GitHub repo

Assumptions / Prerequisites

- Coding experience is assumed
- But no Python required

Course Structure

Day 1: The Core Language

- Python syntax and semantics
- Functions, flow control, etc.
- OOP design
- Classes — DYI objects
- Afternoon exercises

Day 2: Scientific Computing

- NumPy / SciPy / Matplotlib
- JIT compilation with Numba
- Multithreading techniques
- Distributed / cloud computing
- Application: Inventory dynamics
- Afternoon exercises

Day 3: Applications

- Wealth dynamics and measures of inequality
- Job search
- Optimal savings
- Afternoon exercises

Day 4: Working with Data

- Data manipulation
- Popular libraries for statistics / econometrics / machine learning
- Estimation methods
- Afternoon exercises — start final assignment

Programming Background — Software

A common classification:

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

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.

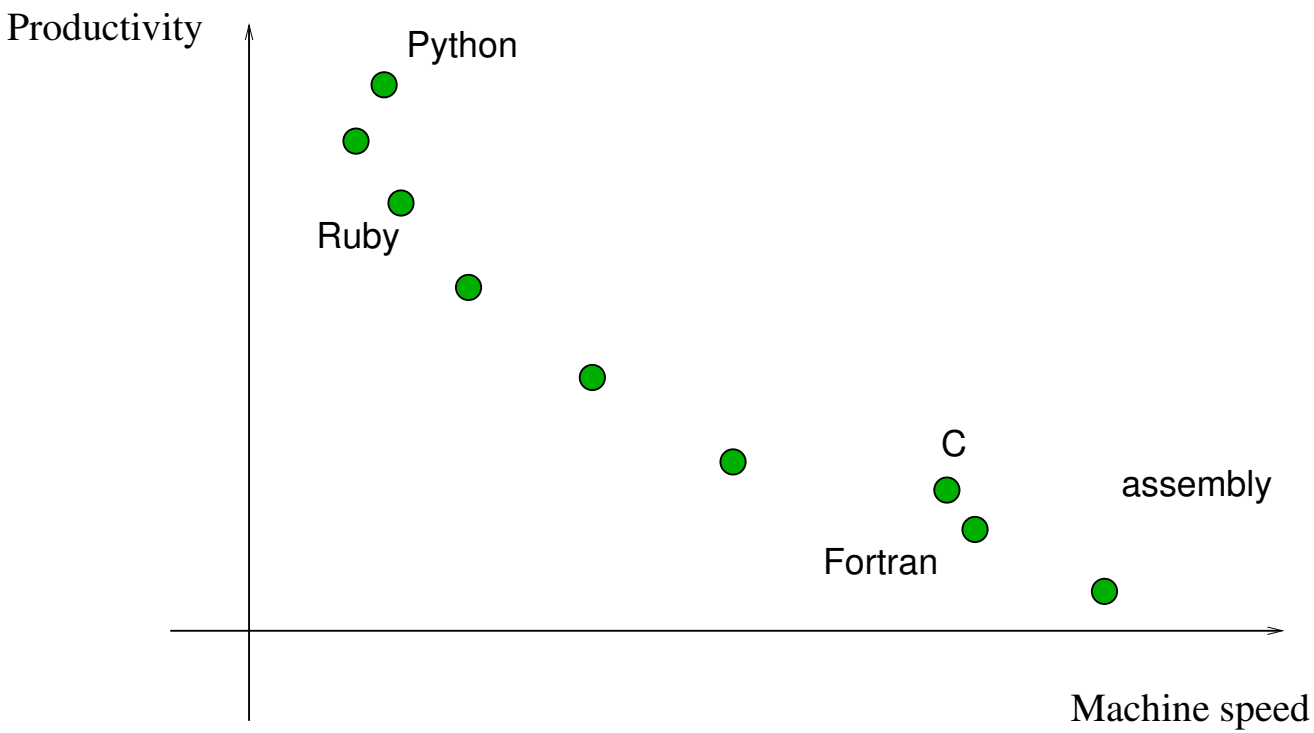
Example. Reading from a file in Python

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

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

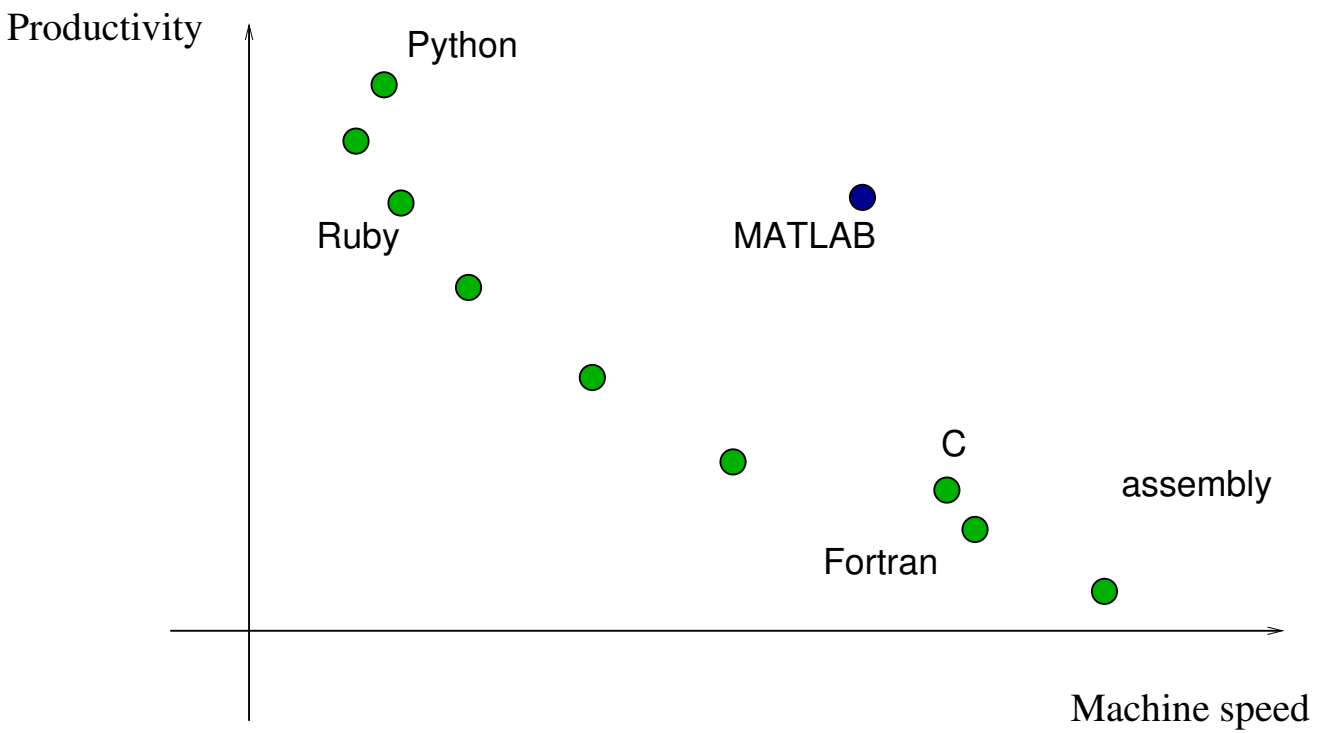

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

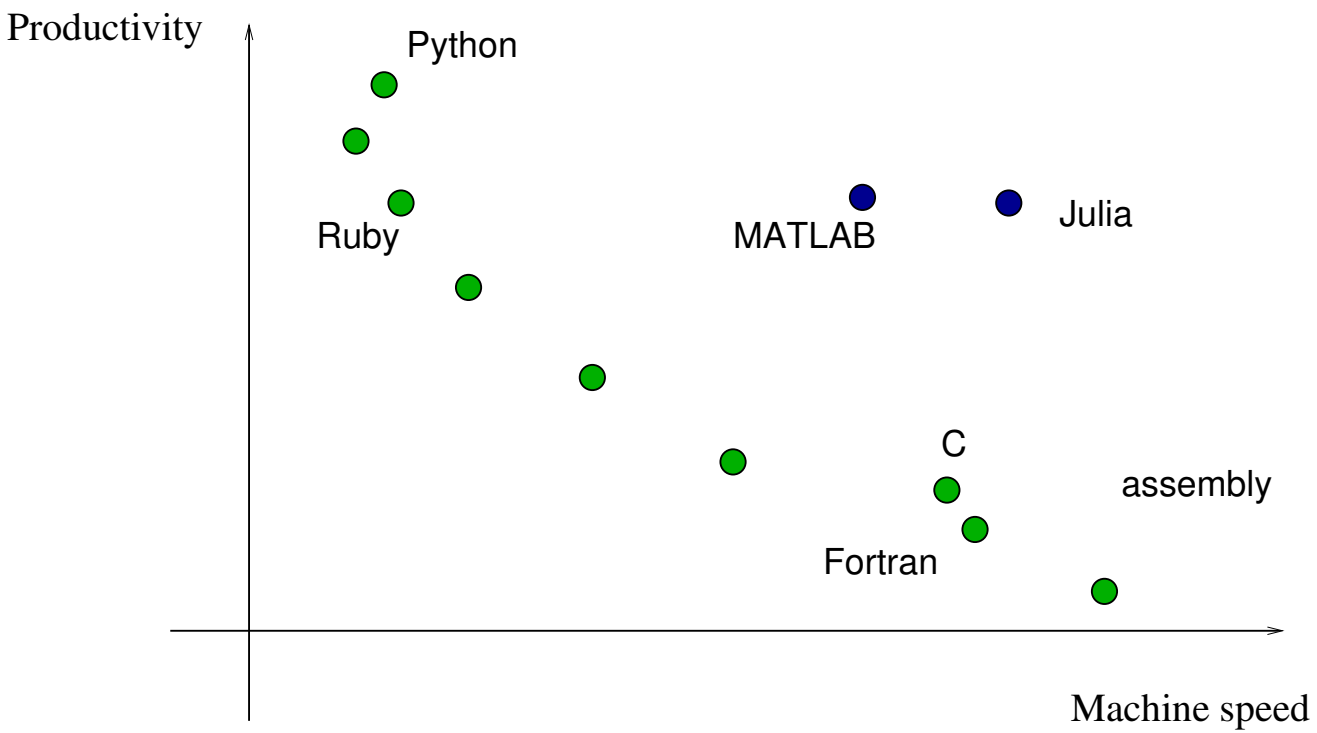
Requirements

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

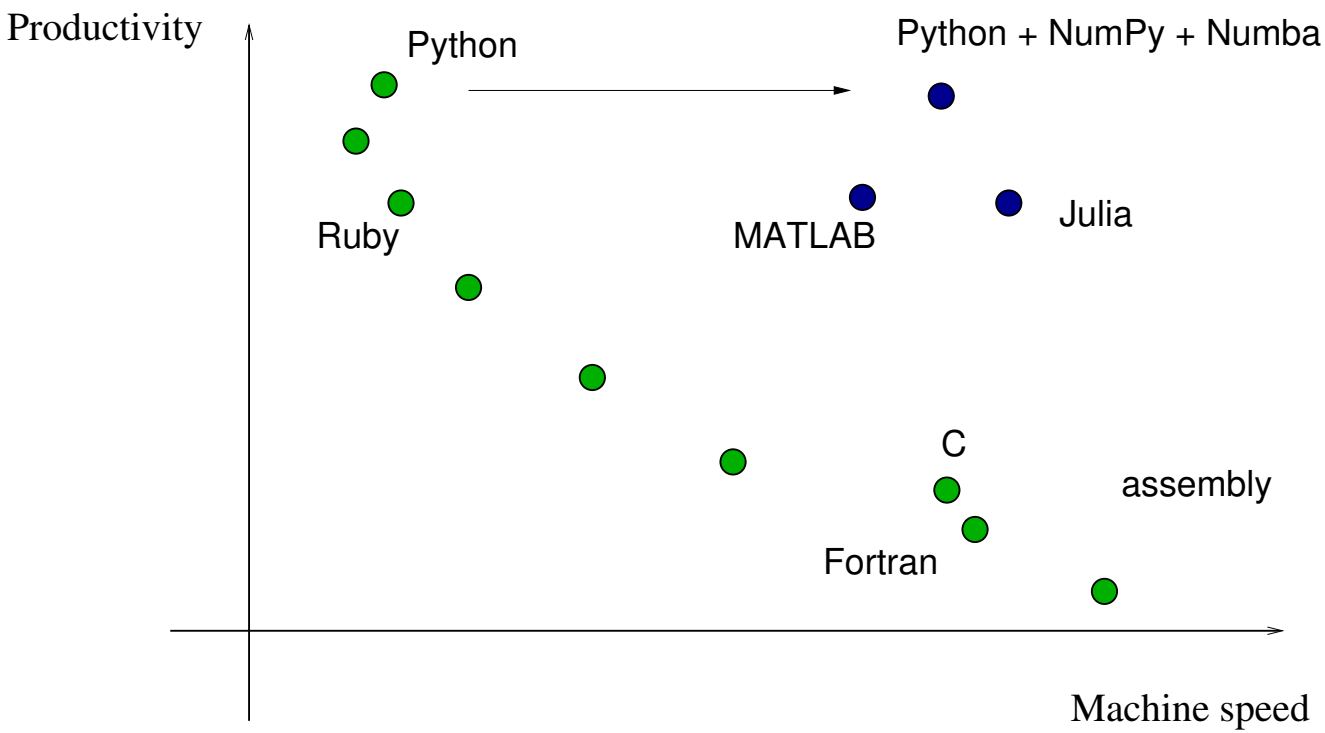
Trade-Offs


Productivity
Python
Ruby
MATLAB
C
assembly
Fortran
Machine speed

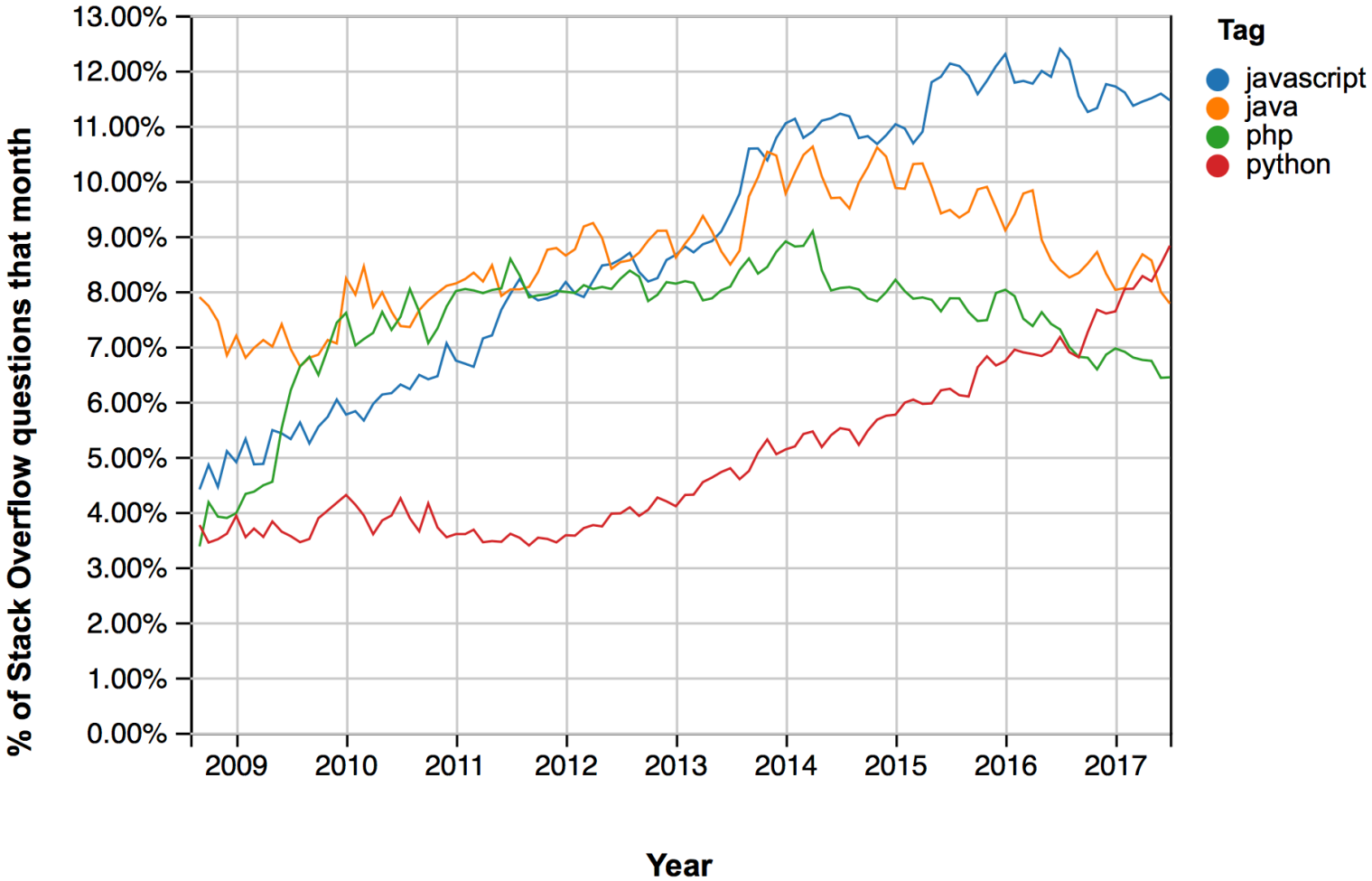
Trade-Offs


Productivity
Python
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Trade-Offs

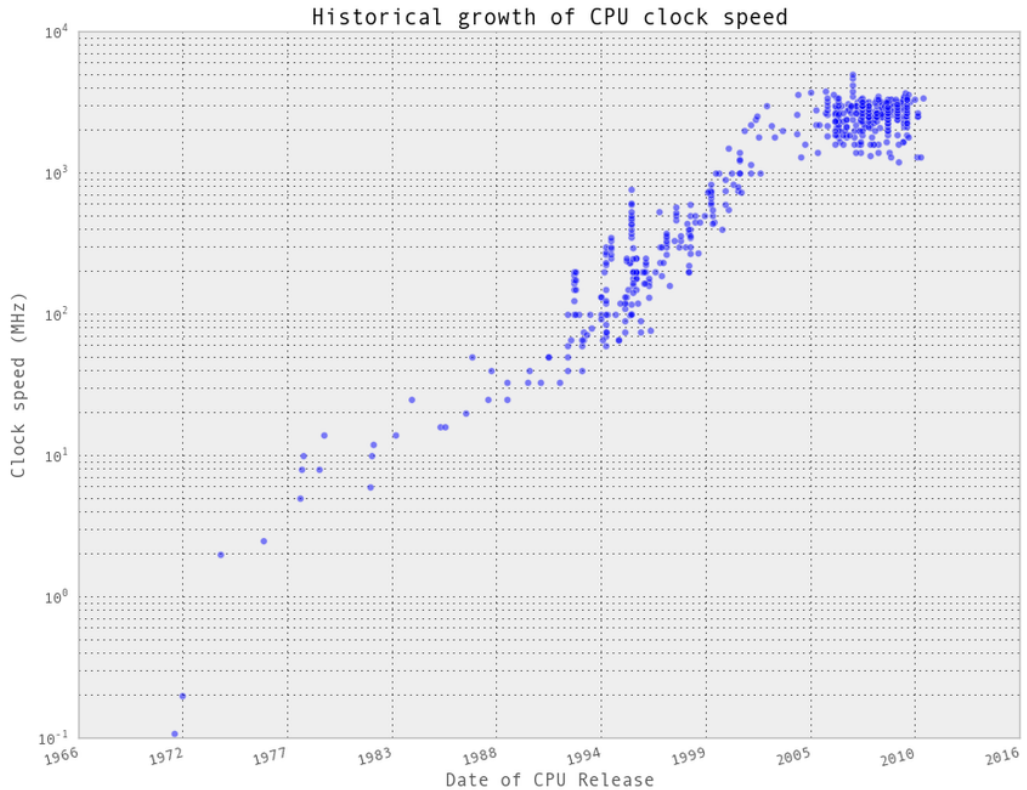

Productivity
Python
Python + NumPy + Numba
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Python vs other popular high level languages

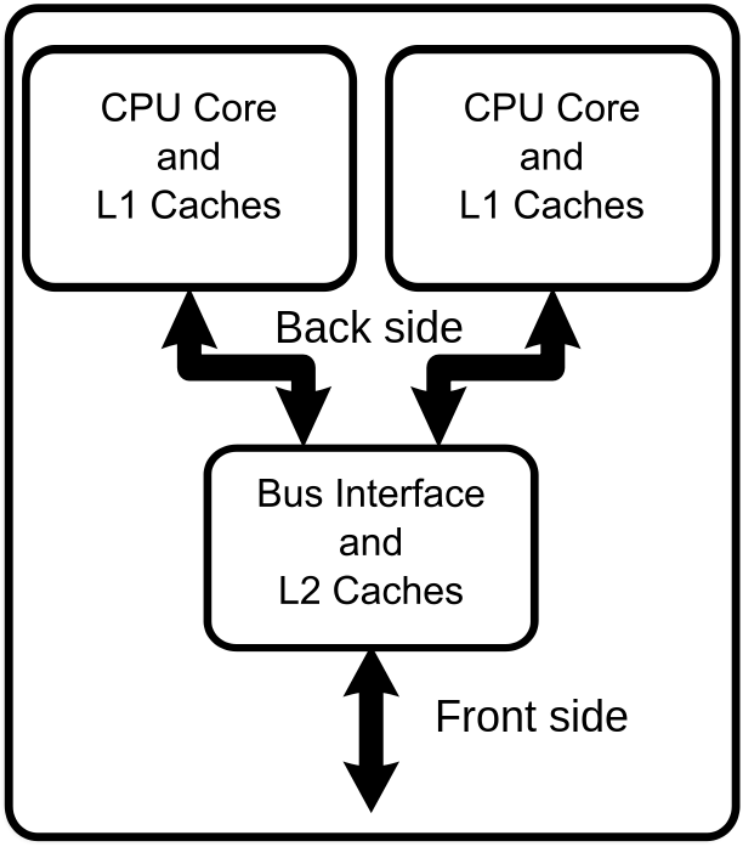


Programming Background — Hardware

CPU frequency (clock speed) growth is slowing



Chip makers have responded by developing multi-core processors



Source: Wikipedia

Exploiting multiple cores / threads is nontrivial

Sometimes we need to redesign algorithms

But the implementation itself is getting easier

- Numba + `@vectorize`
- Numba + `@jit + prange`
- etc.

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`