Programming for Economists

Week 1: The Basics

Tyler Abbot

Department of Economics
Sciences Po

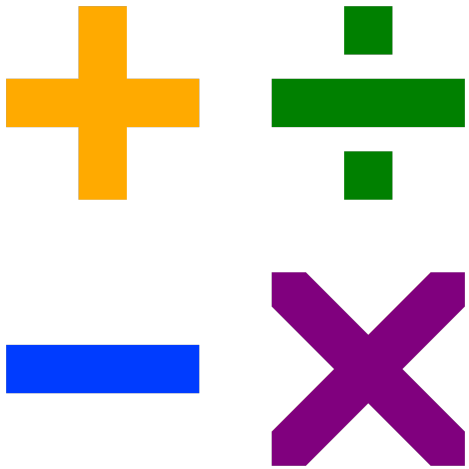
Fall 2016

Outline

About the Course

Why am I here?

- Programming and economics go hand in hand.
- In theory courses, no time to cover the basics.
- In applied courses, assume you know the basics!
- This course will seek to cover *programming* in the most fundamental sense.
- The most complex math in this course will be convex optimization and basic linear algebra.
- The final will be based on a simple sum!



Source: wikipedia

About the Course

Is this worth my time?

Source: life

- Grad school is hard and you should budget your time wisely.
- This course will not be mathematically challenging, but will seek to make your life easier.
- I will not be offended if you drop this class.
- I will make a huge effort to make this course useful (I was once in your position!).

About the Course

What will we do?

- Week 1: Introduction to programming for Economists.
- Week 2: The Gambler's Ruin and basic programming in R.
- Week 3: Monte Carlo and functions.
- Week 4: Least Squares part 1 and linear algebra in R.
- Week 5: Hello World and packages.
- Week 6: Financial Data and the R dataframe.
- Week 7: Least Squares part 2, fitting linear models.
- Week 8: Convex Optimization.
- Week 9: An introduction ot Python.
- Week 10: A whirlwind tour of the big 4 Python modules.
- Week 11: Either multiprocessing in Python or workshop to finish semester project.
- Week 12: Project presentations or multiprocessing in Python.

About the Course

Grades: will I love or hate ths course?

- Pass/fail! Hooray!
- If you submit a strict majority of the problem sets and both the project and take home exam, you will pass. **I repeat: submit at least 6 out of 10 problem sets as well as the project and take home exam.**
- Programming is about practice and the structure is meant to encourage you to try and fail. Only through debugging does one gain true insight.
- I will try to give you time *in class* to work on the problem sets.
- "Do or do not. There is no try." Source: Yoda.

About the Course

Assignments

- Weekly problem sets. Do these! They are designed to take less than 5 hours, so if they take longer tell me and I'll tone it down! They are due by the start of class. Any submissions after this will not count towards the required 6 problem sets.
- Semester project (see next slide).
- Take home exam. This will test your understanding of programming in general. You will be given two weeks to complete the assignment. It's really easy, so just do it.

About the Course I

Semester Project

- Either alone or in a group.
- Create a fully functioning package and push it to Github (if this is mumbo jumbo to you, good, you might learn something in this class).
- Important deadlines:
  - 14/9/2016: Send me an email containing the name of your group and your group members.
  - 30/9/2016: Send me an email containing a proposed project for your package. I will respond with whether I think it is feasible or not.
  - 21/10/2016: Send me an email containing a proposed structure for your package, including functions and their purpose (note: this will probably change).
  - 10/11/2016: Make a pull request to ask me to incorporate your package into the course repository.

- 2/12/2016: Make all changes I requested via the pull request.

- Some ideas for your project:
  - A package to predict outcomes in game of thrones.
  - A package that takes in a person's name and outputs random tweets based on past tweetings (eg Donald Drumpf).
  - A package to predict airline crash based on origin and destination.
  - A package to estimate stock market stuff.
  - A package to write random poems based on some input poem.
  - A package to randomly generate homework assignments from some question pool.
  - A package that uses google images to match a statement to some image or gif and email it to an unwitting target.

About the Course III

Semester Project

- The only requirements are:
  - Your package contains at least 3 functions.
  - Your functions are correctly documented.
  - The package contains at least one test and passes that test and passes the ‘–as-cran‘ check.

About Me

Why am I listening to you?

- Second year Phd student.
- Research heterogeneous agents.
- Broad interest in computational finance and macroeconomic theory.
- My knowledge is practical and not formal...
- I use Python, R, Julia, HTML, Markup, etc. and feel that there is not enough focus on "Programming"!

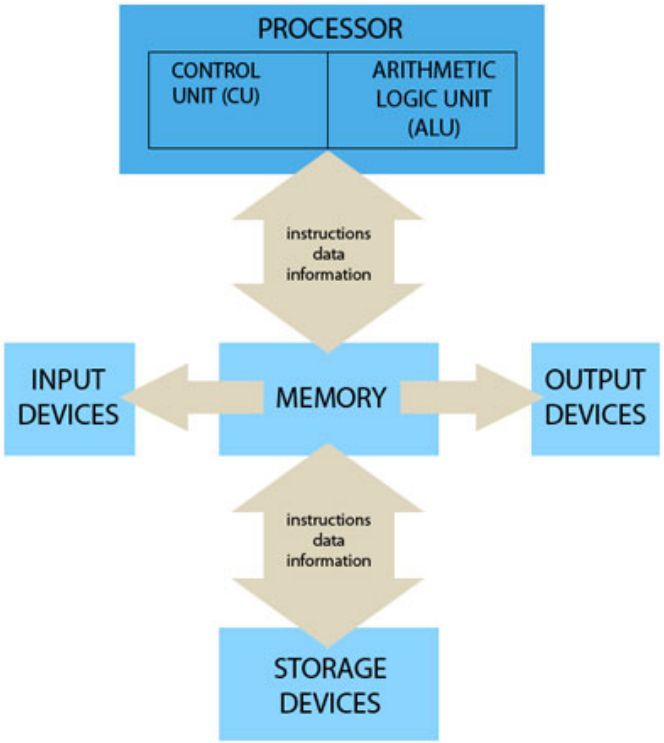
A computer is your dumbest friend.

Bit	Single Binary Digit (1 or 0)
Byte	8 bits
Kilobyte (KB)	1,024 Bytes
Megabyte (MB)	1,024 Kilobytes
Gigabyte (GB)	1,024 Megabytes
Terabyte (TB)	1,024 Gigabytes
Petabyte (PB)	1,024 Terabytes
Exabyte (EB)	1,024 Petabytes

Source: provo.org

- A computer can only do what you tell it.
- Essentially can only understand a binary signal (on and off).
- Two necessary ideas: data storage and calculation.
- Data is stored in "bits" or a series of zeros and ones.
- 8 bits grouped together make a byte.
- Bits and bytes are run through a complicated set of circuits, but the basics are just ones and zeros.

The parts of a computer.

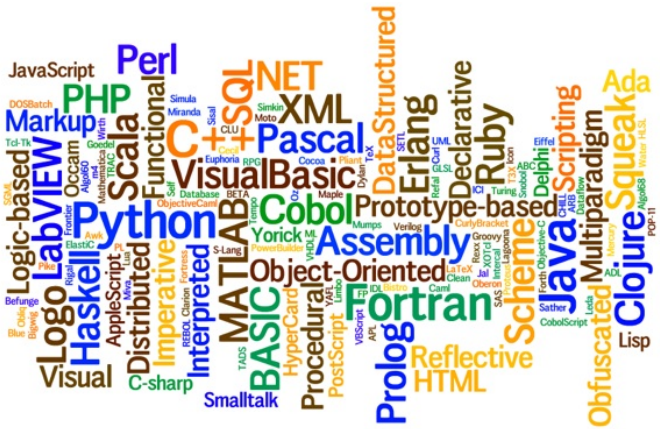


Source: diagramme.info

- Memory: RAM. On my computer: 16 gigabytes.
- Storage: disk or solid state. On my computer: 128 gigabytes SSD.
- CPU: Calculations. On my computer: dual core, 2.8 ghz.
- GPU: Graphics processing unit, used for high performance computing. On my computer: 512 cores.
- One core can calculate one operation at any time, so more cores can give more speed.
- 3Ghz $\Rightarrow$ 3 billion fetch/execute operations per second.

Talking to a computer.

- It speaks many languages, but how are they different?
- Interpreted versus compiled.
- Languages for economists: Python.
- "Languages" for economists: Matlab, Stata, R, Julia.
- Languages you should be able to at least read: Fortran, C++, HTML, Markdown.



Source: cs.tufts.edu

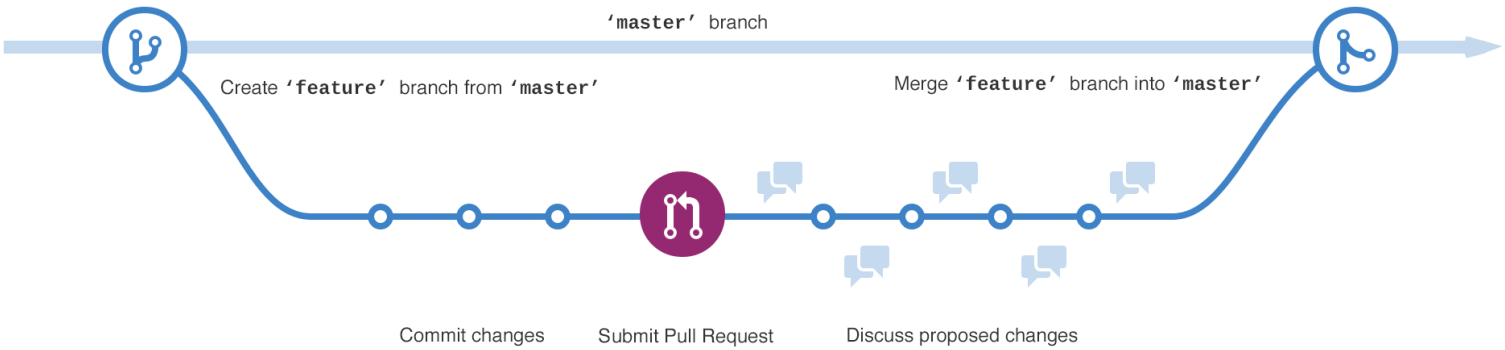
Telling a computer what to do.

- No matter the language, the same ideas in writing code apply.
- One has to keep in mind how a computer thinks, what it can and cannot do, how it tracks types, and how it deals with transforming words to binary.
- For example if you want to leverage multiprocessing, YOU have to do it!
- The reason we work with interpretted languages is their simplicity, but they suffer from speed issues.
- For now, we will focus on simply *writing logical programs*, maybe at the end we will deal with these speed issues.

Big problems.

- Simulating continuous time, stochastic volatility models (a T-dimensional integral).
- Solving heterogeneous agent models in both discrete and continuous time.
- Estimating econometric models about smooth, non-smooth, continuous, discontinuous problems.
- Using machine-learning to find the best predictors in a large set of explanatory variables.

Collaboration: Github



Source: github.com

- Github is an open source development tool.
- Allows version control in a public forum.
- Easily review changes and host code.
- Even get a free website!

How to navigate: Bash

- Using the command line.
- Show on the computer how to `cd` `sudo` `mkdir` `cp` etc.
- Do this by example by creating a local github repository for the course.

How to write a program: algorithm design

- What is an algorithm? Just step by step instructions!
- What is important in algorithm design? The goal, the memory use, the speed, the convergence, etc. (most of these are completely unimportant for this course)
- You (meaning students in this class) should focus on the first part: the goal.
- To design an algorithm you need to break the problem down into the smallest parts your computer can handle.
- Example: calculate $\sum_i x_i$ for a series $\{x_i\}$.

How to write a program: pseudo-code

- Write in plain english.
- Only for you, so use whatever notation makes sense.
- Begin from the largest task and work down to the smallest task.
- As an example, use the simple looped summation.

How to write a program: Documentation

- Always document your code.
- Always update your documentation!
- Always follow the standards for the language (in R this means including an Rd file in any public package).
- Always include a `read_me`, for your own sake as well as others.
- Things to include: author, filename, date of creation, date of modification...

```
"""
Origin: A simple example.
Filename: example_scatter.py
Author: Tyler Abbot
Last modified: 15 September, 2015
"""
```

How to write a program: Actually coding

- Wait a tic, we haven't talked about the language!

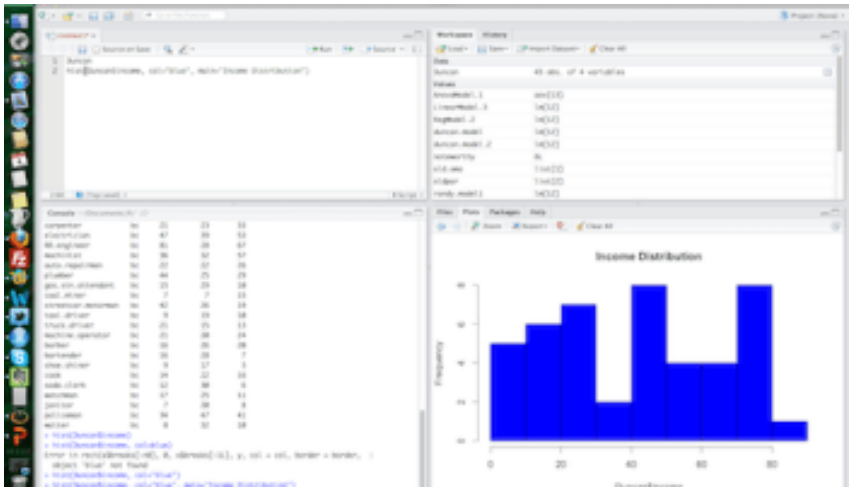
An introduction to R

- R is a software environmnent... not a language!
- Despite this "drawback", it is widely used for statistics.
- Function oriented (vs. object oriented).
- Can interact with other software (matlab, C++ programs, etc.)
- Simple use, but syntax can often be heavy...
- Interpreted.

Source: r-project.org

Ways to interact with R.

- Interactive Development Environment (IDE): suggest RStudio.
- Text editor and command line: suggest Atom.
- Also the Jupyter notebook! Since this is nice for teaching and notetaking, we will see this a lot.



Source: r-bloggers.com

Conclusion I

- Homework:
  1. Create a Github account.
  2. Fork the repo for this course, create a local clone, and configure it to sync with my repository as the upstream.
  3. Create your own repository for this course (different from the one above), clone it on your local machine, and make me a contributor!
  4. Set up an email service to notify me every time you make a commit. You can follow this link: https://help.github.com/articles/managing-notifications-for-pushes-to-a-repository/.
  5. Install RStudio or whatever other method you prefer for working in R.
  6. Install Anaconda so you can run the Jupyter Notebook (https://www.continuum.io/downloads).
  7. Install the Notebook and the R kernel by running `conda install -c r ipython-notebook r-irkernel`

Conclusion II

8. Write an R script which prints "Hello world!" Don't forget the documentation!
9. Open a notebook session by typing `jupyter notebook` at the command line. Create a new R notebook that does the same thing as your R script.
10. Add, commit, and push your script and notebook to a folder titled "Session1_HW" in your repository.
11. Go enjoy happy hour, because you my friend just wrote a computer program!