Introduction to Programming

Lecture 1: Introduction

Clément Mazet-Sonilhac
clement.mazet@sciencespo.fr

Sciences Po Paris

What and why ?

Why are you here ? $\Rightarrow$ because programming is extremely useful in economics !

- dynamics models
- heterogeneous agent models
- machine learning (prediction, data collecting)
- econometrics

What and why ?

Why are you here ? $\Rightarrow$ because programming is extremely useful in economics !

- dynamics models
- heterogeneous agent models
- machine learning (prediction, data collecting)
- econometrics

What are you going to learn ?

- Understand how does a computer think and work
- How to use the terminal ? How to use Git ?
- How to correctly write programs (in R)

What you should know at the end of the class

- Everything about computers (CPU, GPU, RAM, ROM, etc.)
- Not using the mouse : UNIX-shell and the terminal

What you should know at the end of the class

- Everything about computers (CPU, GPU, RAM, ROM, etc.)
- Not using the mouse : UNIX-shell and the terminal
- Writing programs in team : Git + GitHub
- Basics of R

What you should know at the end of the class

- Everything about computers (CPU, GPU, RAM, ROM, etc.)
- Not using the mouse : UNIX-shell and the terminal
- Writing programs in team : Git + GitHub
- Basics of R

\+ Hugo Lhuillier (hugolhuillier.github.io) "Be Lazy Programming Rules"

How should you learn ?

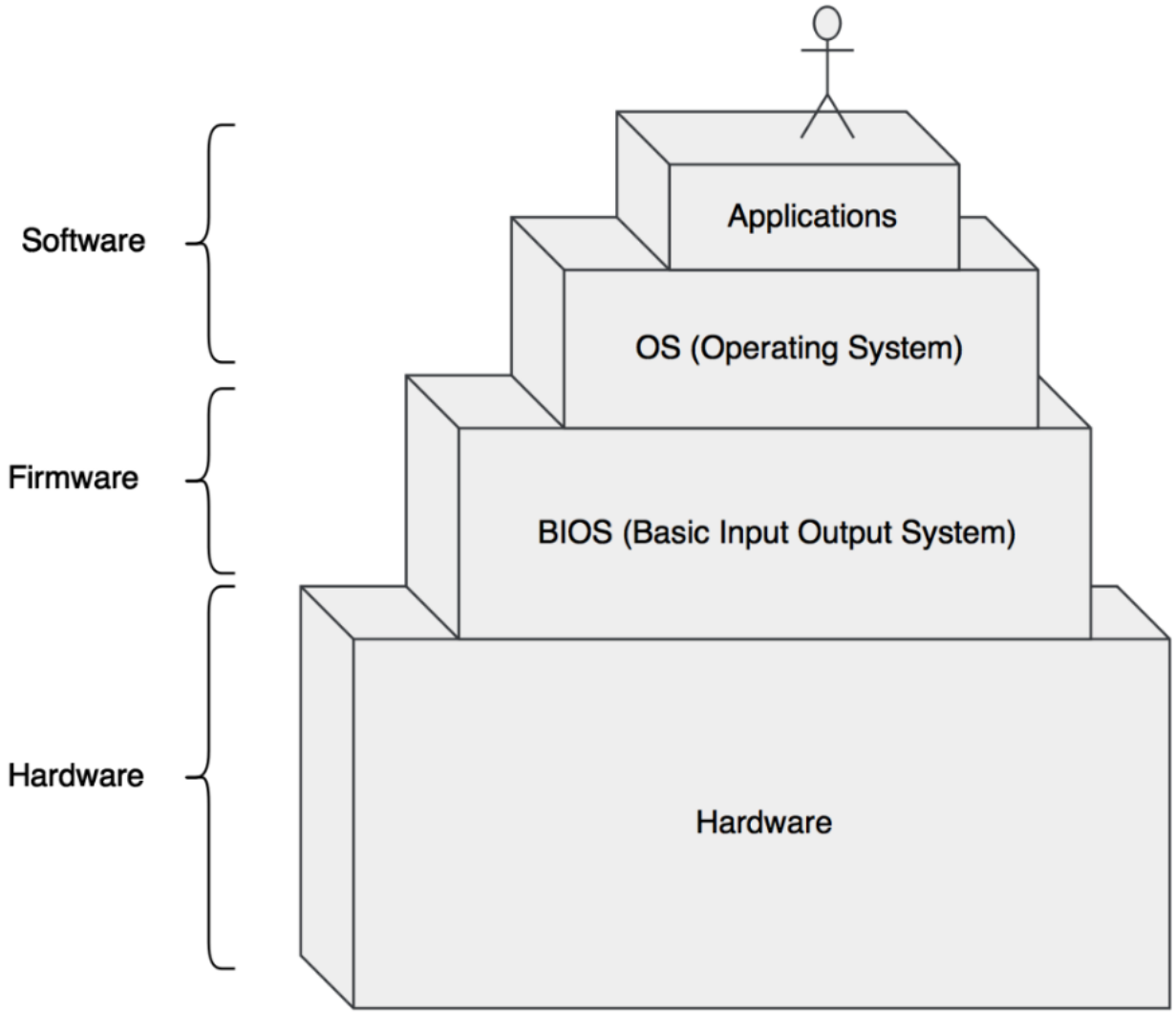
Doing programming is the only way to learn programming

How should you learn ?

Doing programming is the only way to learn programming

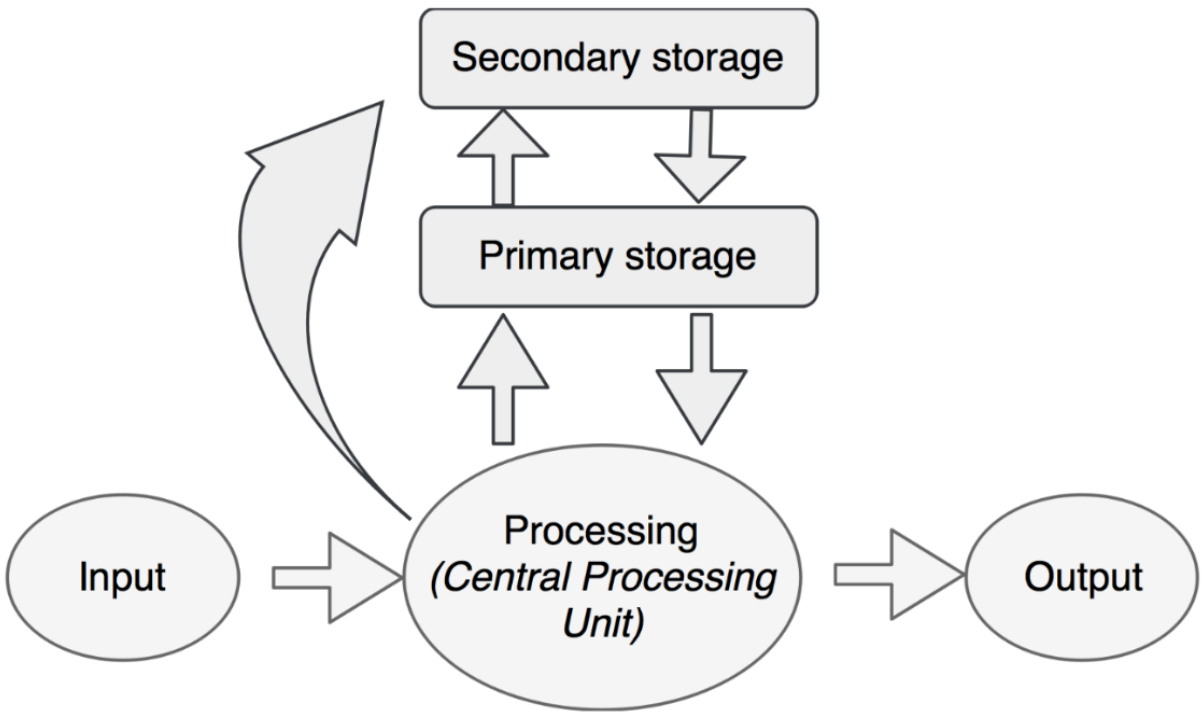
- We will try to make the class as interactive as possible with many exercices
- Weakly homeworks
- Final hackaton by teams of 2-3 students

Computers : Who are they ?



The Hardware (I)

- Definition : the machines, wiring, and other physical components of a computer



The Hardware (II)

The Central Processing Unit (CPU)

- Definition : the brain of a computer

- Made of i) control unit ii) CPU registers and iii) arithmetic and logic unit (ALU)
- Interpret and execute instructions

The Hardware (II)

The Central Processing Unit (CPU)

- The better your CPU, the faster your computer !
- Inside the CPU + some definitions :
  - Clock speed : a single-core CPU only carries one instruction at the time, speed of execution is measure as cycles per second. One cycle per second = 1 Hertz. A 4 GHz processor performs 4,000,000,000 clock cycles per second.

The Hardware (II)

The Central Processing Unit (CPU)

- The better your CPU, the faster your computer !
- Inside the CPU + some definitions :
  - Clock speed : a single-core CPU only carries one instruction at the time, speed of execution is measure as cycles per second. One cycle per second = 1 Hertz. A 4 GHz processor performs 4,000,000,000 clock cycles per second.
  - More speed ? More cores ! a dual core processor can fetch and execute two instructions simultaneously.

The Hardware (II)

The Central Processing Unit (CPU)

- The better your CPU, the faster your computer !
- Inside the CPU + some definitions :
  - Clock speed : a single-core CPU only carries one instruction at the time, speed of execution is measure as cycles per second. One cycle per second = 1 Hertz. A 4 GHz processor performs 4,000,000,000 clock cycles per second.
  - More speed ? More cores ! a dual core processor can fetch and execute two instructions simultaneously.
  - Cache : block memory build onto the processor, which stores the most commonly used instructions and data

Intel® Core™ i7-9700K Processor (12M Cache, up to 4.90 GHz)

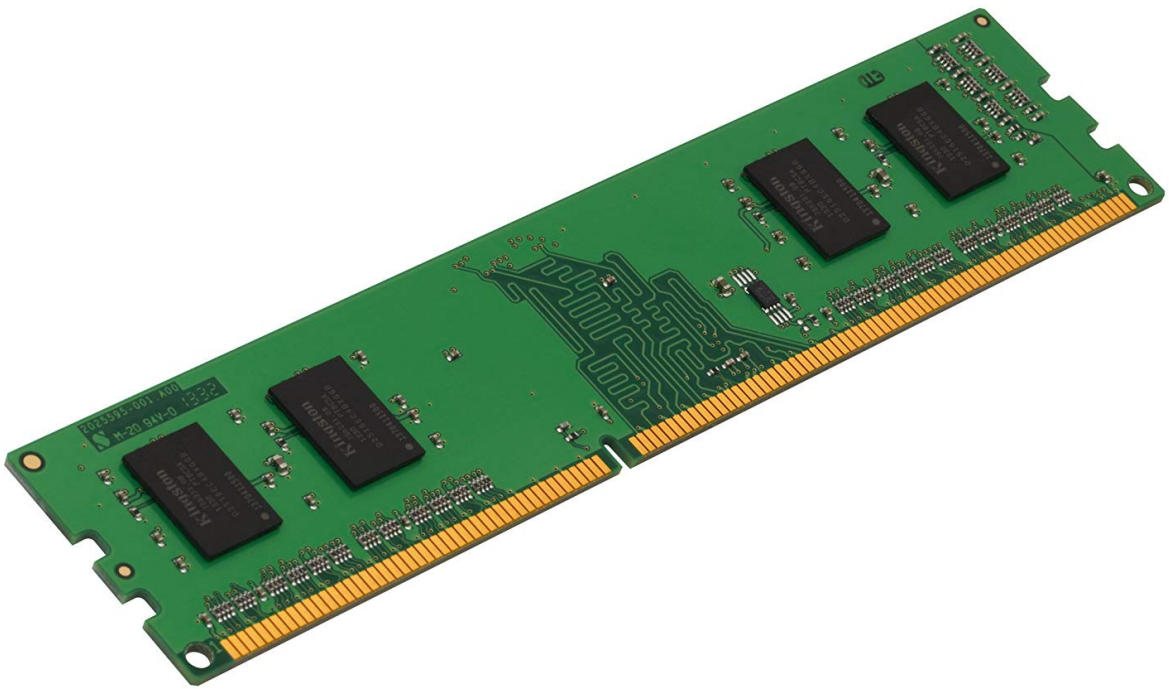
- 12 MB SmartCache Cache
- 8 Cores
- 8 Threads
- 4.90 GHz Max Turbo Frequency

Add to Compare

From $399.89

The Hardware (III)

Memory and storage



The Hardware (III)

Memory and storage

- Memory (RAM) $\approx$ short-term memory for a human $\approx$ volatile memory. Stores data before and after it is processed by the CPU. Data in RAM is NOT permanently written.

The Hardware (III)

Memory and storage

- Memory (RAM) $\approx$ short-term memory for a human $\approx$ volatile memory. Stores data before and after it is processed by the CPU. Data in RAM is NOT permanently written.

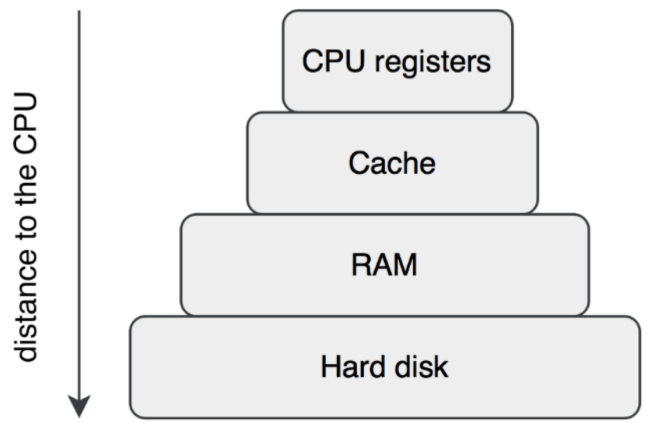
- Storage (ROM + Hard Drive) $\approx$ long-term memory for a human. Store large volume of data that need to persist after the computer is turned off.

The Hardware (III)

Memory and storage

- Memory (RAM) ≈ short-term memory for a human ≈ volatile memory. Stores data before and after it is processed by the CPU. Data in RAM is NOT permanently written.

- Storage (ROM + Hard Drive) ≈ long-term memory for a human. Store large volume of data that need to persist after the computer is turned off.



The Software (I)

Interact with your computer

- The software : programs and other operating information used by a computer
- A program : a list of instructions given to the computer
- Operating system (OS) : a collection of programs in charge of the most basic tasks :

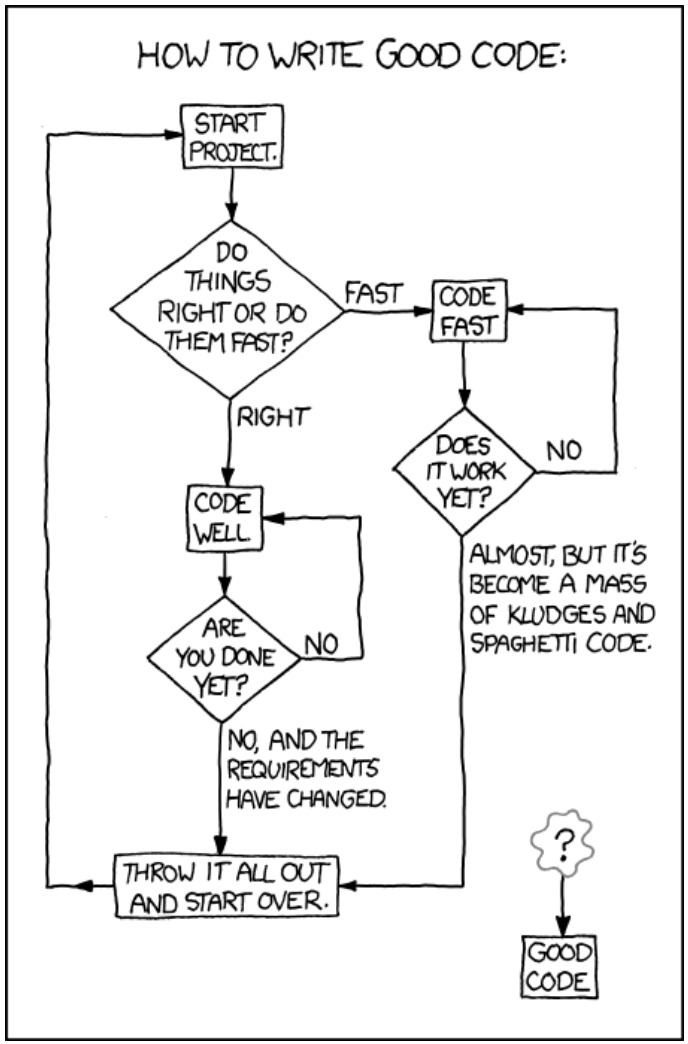
The Software (I)

Interact with your computer

- The software : programs and other operating information used by a computer
- A program : a list of instructions given to the computer
- Operating system (OS) : a collection of programs in charge of the most basic tasks :
  - provide an interface
  - manages the CPU
  - manages memory usage
  - provide security

The Software (II)

How to write a program ?



The Software (II)

Hugo Lhuillier BLPRs

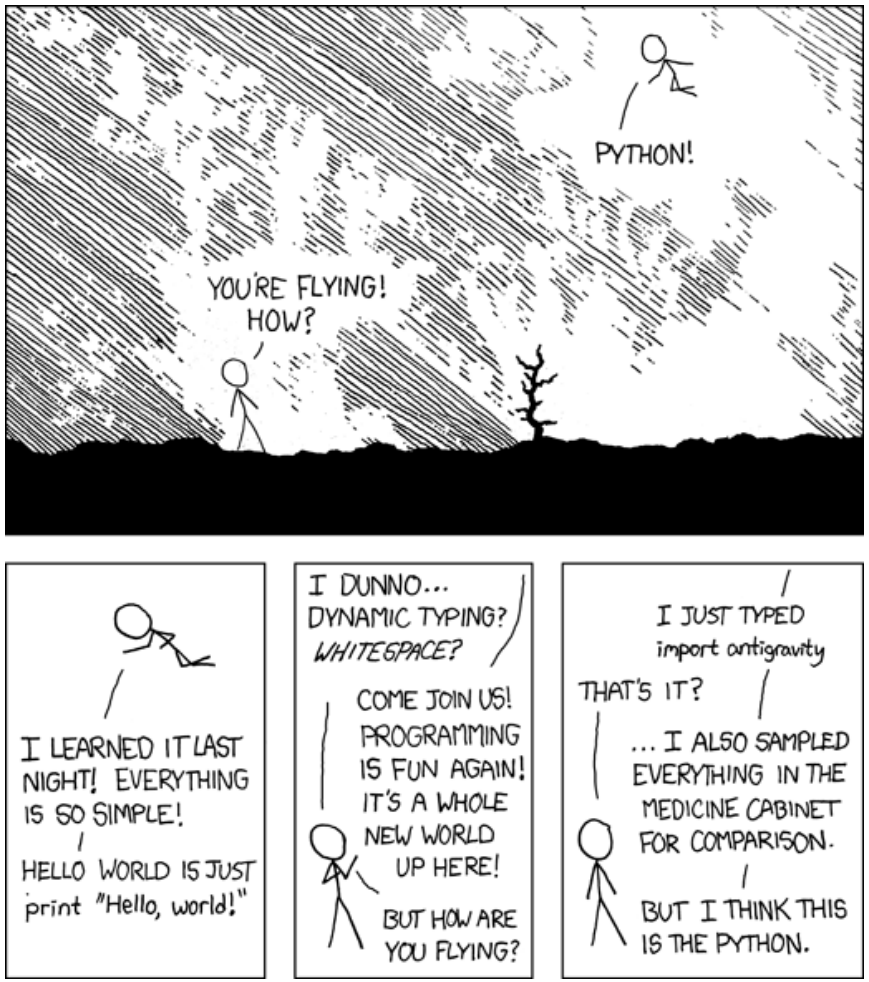
- Write re-usable code
- Write efficient code
- Write code with no bug
- Write nice and documented code

The Software (III)

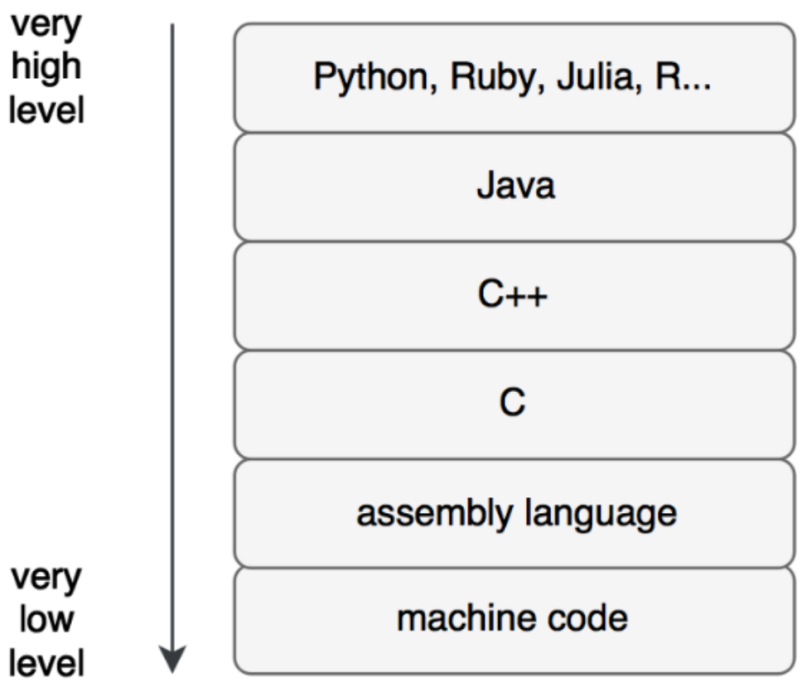
Best coding practices

1. Define CLEARLY what you want to do
2. Write a pseudo code i.e. a draft in mathematics or with words
3. Write AND comment the code
4. Test the code
5. Find the bugs (there always will be)
6. Re-start from 3.

Programming Languages



Programming Languages (I)



Programming Languages (II)

High vs. Low

- Language : source code that will be transformed into binary code that can be interpreted by the CPU
- Translation is operated by a compiler

Programming Languages (II)

High vs. Low

- Language : source code that will be transformed into binary code that can be interpreted by the CPU
- Translation is operated by a compiler
- High vs. Low level : how far are you from the computer language (the machine code) $\approx$ How much translation do you need ?

Programming Languages (II)

Compiled vs. Interpreted

- Compiled languages (Fortran, C, C++) : the source code is compiled into machine code ex ante (by yourself !)
- Interpreted languages (Python, R, Matlab) : the interpreter executes the instruction directly by translating the code into routines that are already compiled in machine code

Programming Languages (II)

Compiled vs. Interpreted

- Compiled languages (Fortran, C, C++) : the source code is compiled into machine code ex ante (by yourself !)
- Interpreted languages (Python, R, Matlab) : the interpreter executes the instruction directly by translating the code into routines that are already compiled in machine code
- Julia (Compiled Just In Time) : speed of compiled + easiness of interpreted ; But you'll have to wait for Florian Oswald next year course !

Programming Languages (III)

Open source vs. proprietary

- Open-source (C++, R, Python) : free to use, huge community, everybody contribute to the shared knowledge and can develop libraries (+)
- Proprietary (Stata, Matlab) : you must pay and use the entire software (language + interface), developers from the company are paid to sustain the source code

Programming Languages

Less is more ?

- Java : `protected int i; int i = 1;`

Programming Languages

Less is more ?

- Java : `protected int i; int i = 1;`
- R : `i = 1`

Programming Languages

Less is more ?

- Java : `protected int i; int i = 1;`
- R : `i = 1`
- Julia (and Python) : `i = 1`

... but Julia can be 250x faster than R !

Programming Languages

Aruoba and Fernandez-Villaverde (2015)

Table 1: Average and Relative Run Time (Seconds)

	Mac			Windows		
Language	Version/Compiler	Time	Rel. Time	Version/Compiler	Time	Rel. Time
C++	GCC-4.9.0	0.73	1.00	Visual C++ 2010	0.76	1.00
	Intel C++ 14.0.3	1.00	1.38	Intel C++ 14.0.2	0.90	1.19
	Clang 5.1	1.00	1.38	GCC-4.8.2	1.73	2.29
Fortran	GCC-4.9.0	0.76	1.05	GCC-4.8.1	1.73	2.29
	Intel Fortran 14.0.3	0.95	1.30	Intel Fortran 14.0.2	0.81	1.07
Java	JDK8u5	1.95	2.69	JDK8u5	1.59	2.10
Julia	0.2.1	1.92	2.64	0.2.1	2.04	2.70
Matlab	2014a	7.91	10.88	2014a	6.74	8.92
Python	Pypy 2.2.1	31.90	43.86	Pypy 2.2.1	34.14	45.16
	CPython 2.7.6	195.87	269.31	CPython 2.7.4	117.40	155.31
R	3.1.1, compiled	204.34	280.90	3.1.1, compiled	184.16	243.63
	3.1.1, script	345.55	475.10	3.1.1, script	371.40	491.33
Mathematica	9.0, base	588.57	809.22	9.0, base	473.34	626.19

Programming Languages

This course

- We will learn R : open-source, interpreted and user-friendly !
- Moreover, R is fantastic for people working with data.

Your first code (I)

Example : the Euler equation

- The Euler equation :

$$u'(C) = \beta \cdot \mathbb{E}_t[R_{t+1} u'(C)]$$

- Find $\hat{C}$ s.t. $F(\hat{C}) = 0$ where :

$$F(\hat{C}) = u'(\hat{C}) - \beta \cdot \mathbb{E}_t[R_{t+1} u'(\hat{C})]$$

Your first code (II)

Example : the Euler equation

- Find $\hat{C}$ s.t. $F(\hat{C}) = 0$ where :

$$F(\hat{C}) = u'(\hat{C}) - \beta \cdot \mathbb{E}_t[R_{t+1} u'(\hat{C})]$$

- Transpose $F(\hat{C})$ into a code line :

```
F(C) = u'(C) - beta * R * u'(C)
```

- and then, find the $\hat{C}$ s.t $F(\hat{C}) = 0$

```
find-root(F)
```

Your first code (II)

Example : the Euler equation

- Finally, comment your code :

```
F(C) = u'(C) - beta * R * u'(C) # the Euler equation
find-root(F) # solve for the consumption level s.t. F(C) = 0
```

Roadmap

- Next weeks (Part #1) : 1-2 lectures about UNIX-shell or *How to directly interact with your computer through the terminal ?* and 2-3 lectures about Git and GitHub *How to write programs in team and deal with version control ?*.

Roadmap

- Next weeks (Part #1) : 1-2 lectures about UNIX-shell or *How to directly interact with your computer through the terminal ?* and 2-3 lectures about Git and GitHub *How to write programs in team and deal with version control ?*.

- Part #2 : 4-6 lectures about programming in R, i.e. learning basics of programming by doing programming !

Roadmap

- Next weeks (Part #1) : 1-2 lectures about UNIX-shell or *How to directly interact with your computer through the terminal ?* and 2-3 lectures about Git and GitHub *How to write programs in team and deal with version control ?*.
- Part #2 : 4-6 lectures about programming in R, i.e. learning basics of programming by doing programming !
- Needed for next lecture : computer with a terminal (Windows users (?) should come talk to me)