# *Introduction to Programming*
# *# 1: introduction*

Hugo Lhuillier

February 1, 2018

Master in Economics, Sciences Po

## The why & what

Why are we here?

- Because programming is *extremely* useful in economics:
  - 99.9% of dynamic models
  - Heterogeneous agent models
  - ML & Bayesian estimators
  - Machine learning
  - ...

What are we going to learn?

- How to write programs according to the BLRs
- Warning: not a class on numerical methods

## The Be Lazy (Programming) Rules©

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

## The Be Lazy (Programming) Rules©

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

## The Be Lazy (Programming) Rules©

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

## The Be Lazy (Programming) Rules©

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

## What you should know at the end of the class

- Computer: the must knows (CPU, GPU, ALU, RAM, GHz etc.)
- Dropping the mouse: UNIX-shell
- Writing programs in team: Git & Github
- The fundamental introduction to R
- The way to go: Julia

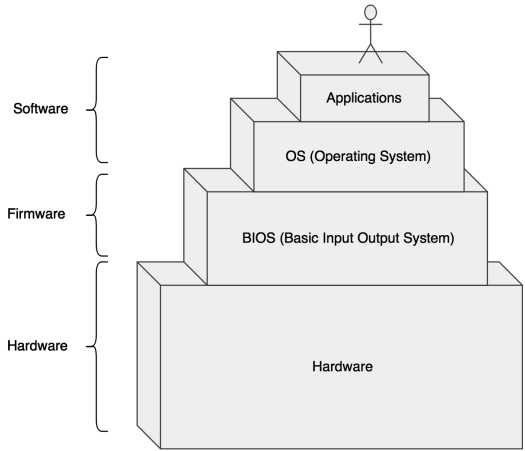
## Assignments

Will try to make the class as fun & as interactive as possible

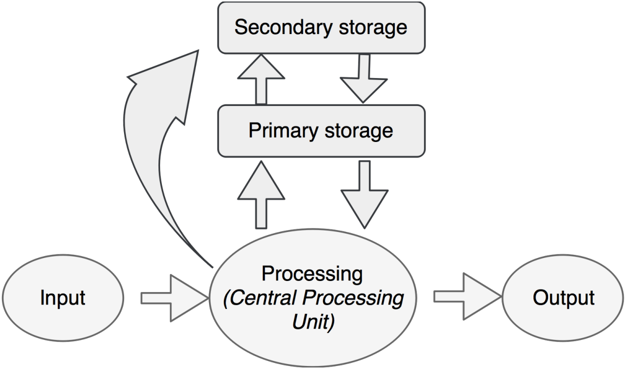
- Weekly homeworks
  - Heads-up: successful homeworks might give you bonuses during the hackathon.
- 2 to 4-hour hackathon by teams of 2-3

*Computers: who are they, and what do they do?*

**Figure 1:** A computer



# The hardware



## The hardware
### The CPU

- Execute and interpret instructions
- Made of
  1) Control unit
  2) CPU registers
  2) Arithmetic and logic unit (ALU)
- The better your CPU, the faster your computer
  - Clock speed: CPUs only carry one instruction at a time, with speed of execution measured as cycles per second. One cycle per second = 1 hertz
  - Cores: a dual core processor can fetch and execute two instructions simultaneously
  - Cache: a block of memory build onto the processor. Stores the most commonly used instructions and data


**Figure 2:** An Intel processor

# The hardware
## Memory & storage

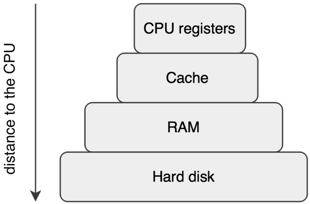



**Figure 3:** Different storages

1) Memory $\approx$ short term memory for a human
   - stores the data after it is input to the system and before it is processed
   - stores the data after it has been processed but before it has been released as output
   - stores the instructions needed by the CPU
2) Storage $\approx$ long term memory for a human: stores large volumes of data that need to persist after the computer is turned off

# Data

- Computers are electronic machines
- Binary coding system: computers only understand `on` (= 1) or `off` (= 0)
- One binary digit = 1 bit. 10010110 = 8 bits = 1 byte
- *Everything is turned into bytes*

## The software

- A program: a list of instructions given to the computer
- Operating system (OS): a collection of program in charge of the basic tasks
    - provides an interface (GUI and / or CLI)
    - manages the CPU
    - allows for multi-tasks
    - manages memory usage
    - provides security
    - ...

# The software

## How to write a program

0) Always define clearly what you want to do
1) Write the pseudo-code (plain English or mathematics)
2) Write the code
3) Test the code
4) Comment the code

# The software

## How to write a program: example

- THE EULER EQUATION (at the ss)

$$u'(C^{\star}) = \beta \mathbb{E}_t R_{t+1} u'(C^{\star}).$$

# The software

## How to write a program: example

- THE EULER EQUATION (at the ss)

$$u'(C^\star) = \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

- Find $C^\star$ such that $F(C^\star) = 0$, where

$$F(C^\star) = u'(C^\star) - \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

## The software

### How to write a program: example

- THE EULER EQUATION (at the ss)

$$u'(C^{\star}) = \beta \mathbb{E}_t R_{t+1} u'(C^{\star}).$$

- Find $C^{\star}$ such that $F(C^{\star}) = 0$, where

$$F(C^{\star}) = u'(C^{\star}) - \beta \mathbb{E}_t R_{t+1} u'(C^{\star}).$$

- Code
  - `F(C) = u'(C) - BETA * R * u'(C)`
  - `find-root(F)`

## The software

### How to write a program: example

- THE EULER EQUATION (at the ss)

$$u'(C^\star) = \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

- Find $C^\star$ such that $F(C^\star) = 0$, where

$$F(C^\star) = u'(C^\star) - \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

- Code
  - `F(C) = u'(C) - BETA * R * u'(C)`
  - `find-root(F)`
- Test: $F(C^\star) < \epsilon$, where $\epsilon \to 0$ (numerically) and $C^\star > 0$

## The software

### How to write a program: example

- THE EULER EQUATION (at the ss)

$$u'(C^\star) = \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

- Find $C^\star$ such that $F(C^\star) = 0$, where

$$F(C^\star) = u'(C^\star) - \beta \mathbb{E}_t R_{t+1} u'(C^\star).$$

- Code
  - `F(C) = u'(C) - BETA * R * u'(C)`
  - `find-root(F)`
- Test: $F(C^\star) < \epsilon$, where $\epsilon \to 0$ (numerically) and $C^\star > 0$
- Document 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`

*Computers: how to interact with them?*

## Programming languages

- Any source code is eventually transformed into binary code
- That translation operation is performed by a compiler
- Low vs. high programming languages: the closer you are to what the computer is actually doing, the "lowest" the language is
    - Assembly language

      ```
      mov r3, #1
      str r3, [r11, #-8]
      ```

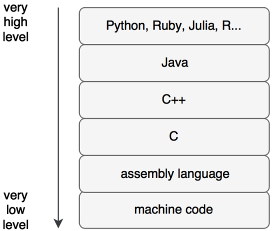



**Figure 4:** High vs. low programming languages

## Programming languages

- Any source code is eventually transformed into binary code
- That translation operation is performed by a compiler
- Low vs. high programming languages: the closer you are to what the computer is actually doing, the “lowest” the language is
    - Assembly language

      ```
      mov r3, #1
      str r3, [r11, #-8]
      ```

    - produces `i = 1`

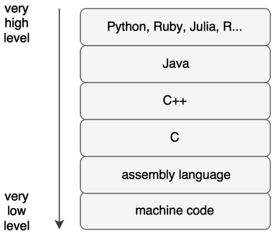



**Figure 4:** High vs. low programming languages

## Programming languages

### Interpreted vs. compiled

- Compiled languages (Fortran, C . . . ): the source code is compiled to machine code *ex ante*
  - Advantage: faster, because the executed code is tuned to the task
  - Disadvantage: longer & more complex code (static typing)
- Interpreted languages (Python, Matlab, R . . . ): the interpreter executes the program directly and translates each instruction into routines that already compiled in machine code
  - Easier to use and to read, but lose the speed of compiled languages

## Programming languages
### Interpreted vs. compiled

- Compiled languages (Fortran, C . . . ): the source code is compiled to machine code *ex ante*
  - Advantage: faster, because the executed code is tuned to the task
  - Disadvantage: longer & more complex code (static typing)
- Interpreted languages (Python, Matlab, R . . . ): the interpreter executes the program directly and translates each instruction into routines that already compiled in machine code
  - Easier to use and to read, but lose the speed of compiled languages
- ♡ Julia ♡: compiled just in time (JIT). The speed of compiled language with the facility of interpreted languages

# Programming languages

## Interpreted vs. compiled (example)

- Java: `protected int i; i = 1;`
  - If try `i = 1.0;` $\Rightarrow$ ERROR
- Python: `i = 1`
- Julia: `i = 1`, but will run code 5 to 100 times faster than Python

## Programming languages

### Interpreted vs. compiled (example)

- Java: `protected int i; i = 1;`
    - If try `i = 1.0;` $\Rightarrow$ ERROR
- Python: `i = 1`
- Julia: `i = 1`, but will run code 5 to 100 times faster than Python

# Programming languages
## Interpreted vs. compiled (example)

- Java: `protected int i; i = 1;`
  - If try `i = 1.0;` ⇒ ERROR
- Python: `i = 1`
- Julia: `i = 1`, but will run code 5 to 100 times faster than Python

# Programming languages
## Speed comparisons

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 |

**Figure 5:** From Aruoba & Fernández-Villaverde (2015)

# To go further

- Aruoba, S. B., & Fernández-Villaverde, J. (2015). A comparison of programming languages in macroeconomics. *Journal of Economic Dynamics and Control, 58*, 265-273
- Bitesize by BBC