BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

Introduction to Scientific Programming and Machine Learning with Julia

Kick-off meeting

Antonello Lobianco

!! Please enable subtitles !!

"A language that doesn't affect the way you think about programming is not worth knowing."

-- Alan Perlis(American computer scientist)

Today's objectives

- **What this course is about**
- **How this course is organised**
- **Who am I ? (Who are you ?)**
- **What is Julia ? (Why to bother with yet-an-other language ?)**
- **What is Artificial Intelligence, Machine Learning, Deep learning.. ? Will robots rule the world ?**
- **Let's set up the environment for the course !**
  - **diff, git, github**
  - **computational environment**
- **Let's start by learning to work with packages**
- **A quick view of simple ML applications**

This course

- Two parts:
  - A brief but solid introduction to the Julia Programming Language
    - General syntax and programming constructs (packages, data types, control flows, macros, I/O,...)
    - "scientific" programming: wrangling and visualising data, optimisation problems, symbolic computation, statistics, documentation...
  - An introduction to ML and ML algorithms
    - Scope, objectives, principles and terminology
    - Main algorithms for classification and regression (Perceptron, Neural Networks, Random Forests, KMeans, GGM,...)
    - Set up a ML workflow and model evaluation
- The course follows the following references :
  - The book "Julia Quick Syntax Reference" (Apress 2019) for the Julia part (PDF available on the course site – to not redistribute, only for you !)
  - Some very "dirty" notes available here: https://raw.githubusercontent.com/sylvaticus/MITx_6.86x/master/MITx_6.86x_notes.md.pdf for the ML part
- Course on CoursEnLigne: https://coursenligne.parisnanterre.fr/course/view.php?id=7728
- Course on GitHub: https://github.com/sylvaticus/IntroSPMLJuliaCourse

Course organisation

- This kick-off meeting
- 6 additional recorded video lessons:
  - Basic Julia programming
  - Scientific programming with Julia
  - Introduction to Machine Learning 1
  - Introduction to Machine Learning 2
  - Neural Networks
  - Random forests and clustering
- ~~One Lesson == One week~~ One Lesson != One week
  - The video, Jupyter notebook and related exercises open each week and are due for the ending of the week (see calendar for details
  - You can either watch the video or just play by yourself with the associated Julia script or study them in the "compiled form" online
- The computational environment
  - The best option: you install the computational environment on your own PC (one reason why we have a kick-off meeting)
  - The fallback: a server is available for you for this course
    - could be slow
    - documents will not remain available after the course end
  - Other online computational environments with Julia support: https://cocalc.com, https://nextjournal.com, https://notebooks.gesis.org, https://codeocean.com, https://visualstudio.microsoft.com/services/github-codespaces, https://replit.com/languages/julia
- The exam:
  - 10% on-line exercises during the course
  - 30% project (last week)
  - 60% in-presence exam

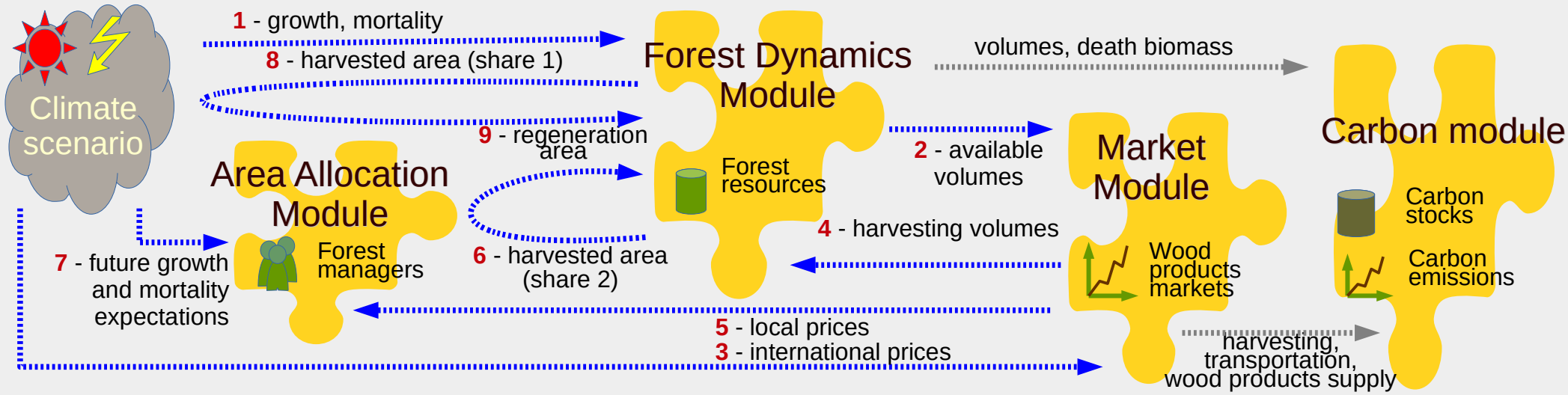
Let's present us

- I'm a research engineer in Forest and Environmental Economics at BETA (Bureau d'économie théorique et appliquée) Nancy (UMR AgroParisTech, INRAE, University of Lorraine, University of Strasbourg, CNRS)
- My work involves a lot of modelling and simulations
  - agent-based farm-level to regional simulations → RegMAS (Regional Multi Agent Simulator)
  - bio-economic models of the forest sector (forest dynamics, timber markets equilibrium, carbon cycle) → FFSM (French Forest Sector Model)
- My pathway with programming:
  - Commodore 16, Commodore64 Basic, DOS, VBA, Pascal, Perl, PHP, C++, Python, Julia
  - I have experience also with setting LAMP servers/middleware
  - What I used this for? Everything from management applications (including the students management office of my university), web sites, numerical utilities.. research
- The ML part
  - started with a MOOC from MIT "Machine Learning with Python: from Linear Models to Deep Learning"
  - implemented a ML library: BetaML

- And you ?
  - Which programming languages do you use (let me guess.. R ? Python? Maybe javascript or Matlab ?)
  - What's your background ?
  - Have you already used ML algorithms ?
  - What are you looking / expect from this course ?

BETA Bureau d'économie théorique et appliquée
AgroParisTech
Talents d'une planète soutenable

My main work: the FFSM model



Recursive bio-economic model with coupled biological forest growth model and partial equilibrium market of forest products.

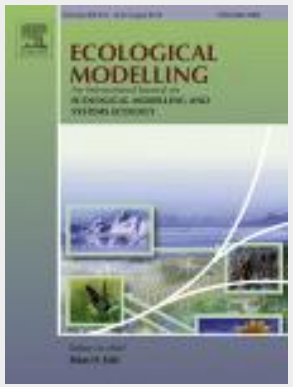
Major strength in the capacity to catch the interaction between drivers. A **framework** with five interconnected modules:

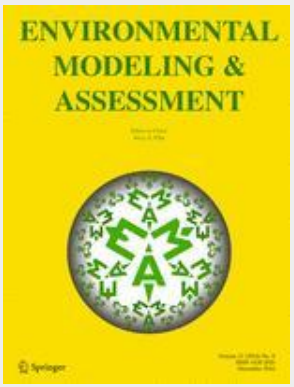
Forest dynamics module:	model forest resources -
Pathogen module:	simulation of a spatially explicit pathogen
Market module:	HWP markets (supply, demand, trade..) - partial eq.
Area allocation module:	forest investment choices – agent based
Carbon module:	mitigation potentials of the forest sector

Use of FFSM

FFSM has been used for a wide range of analysis, of both positive and normative nature:

- Should policies subsides targeted to sequestrated carbon or fuelwood ?
- Economic and resource impact of fuelwood subsidies
- Carbon tax and mitigation policies: impact on the French forest sector
- Is it better to store or export the excess of timber following large windfalls ?
- How much the future of French forests depends from human management and forest managers risk-aversion
- What is the climate change mitigation potential of the French forest sector ?
- Which is the ecological and economic effect of the introduction of the ash pathogen ?





BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

JULIA

The Julia Programming Language

- Julia is one of the three main open-source languages used in data science (the other two being R and Python – JuPyteR)
- The newest of the three (Python: 1991, R: 1993, Julia: 2012)
- Key principles:
  - Interactive
  - Dynamically-typed but Just In Time compiled (using LLVM - 2003) with type inference
  - Garbage-collection of the memory
  - Most numerical functions in core or "standard library" ("batteries included")
  - Reflective (meta-programming)
  - Multiple-dispatch
  - C and Fortran libraries directly callable
  - High-level parallelisation
  - Unicode supported in variable names
  - Default git-based package manager with light environments
- Effect:
  - High-level but "tweakable" to the lowest bit
  - Efficient from both the programmer prospective ("expressive") and the hardware one ("fast")
    - solves the trade-off that has long existed in programming: fast coding vs. fast execution
    - much faster library development: libraries are wrote in the same language as the one they are employed
  - Simplified libraries API (e.g. JuMP vs Pyomo)
  - Independently-developed libraries work well together
  - Play well with older languages (PyCall, RCall, ...)
  - Scalable from a single laptop to High Performance Computing (HPC) clusters
  - Code closer to math representation: `α = 2; β² = 2α^2`
  - Reproducible research

In which languages libraries are developed ?

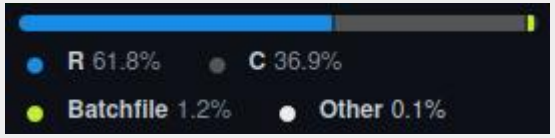
- Julia libraries
  - DataFrames:

  - Flux:

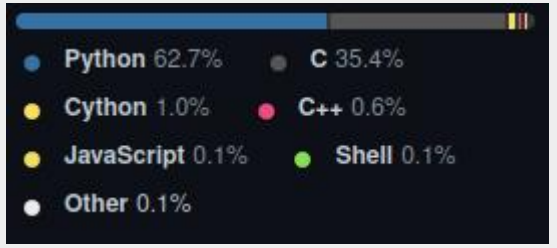
  - BetaML:

- R libraries
  - Dplyr:

  - data.table:

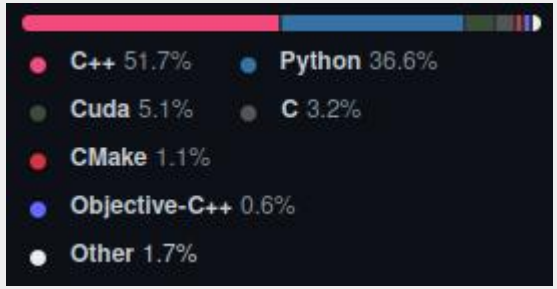


- Python libraries:
  - Numpy:



  - Pandas:

  - Pytorch:



BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

ML

A bit of terminology

- Does a pocket calculator use Artificial Intelligence ?

A bit of terminology

Artificial Intelligence

- Definitions
  - "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages." (Oxford)
  - "1 : a branch of computer science dealing with the simulation of intelligent behavior in computers
    2 : the capability of a machine to imitate intelligent human behavior" (Merriam-Webster)
  - "The study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn" (Cambridge)

- Described in relation to human intelligence: an "intelligent agent" has a goal, senses its environment and interacts with it, learns from the result of its actions and act toward achieving its goal, given perceptual limitations and finite computation (Poole, David; Mackworth, Alan; Goebel, Randy (1998). *Computational Intelligence: A Logical Approach*)
- What is "normally requiring human intelligence" changes with time !
- Still pretty vague (so much that definitions is provided giving specific elements that are considered part of AI rather than characteristics of the set)
- Conied in the '50s, then almost abandoned, finally "resurgence" in specific fields
- I personally prefer not to use it
- We don't discuss ethical issues in this course

A bit of terminology

Machine Learning

- Definitions
  - "The use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data." (Oxford)
  - "The process by which a computer is able to improve its own performance (as in analyzing image files) by continuously incorporating new data into an existing statistical model" (Merriam-Webster)
  - "The process of computers changing the way they carry out tasks by learning from new data, without a human being needing to give instructions in the form of a program" (Cambridge)
- key concepts:
  - "learn" the relations between I/O of a system ("black box") or its structure E.g. image image recognition, clustering
    - most assumptions of a statistical model aren't needed
    - emphasis on online algorithms in these definitions
  - "prediction" a key scope → most jobs are actually predictions (wait next slide)
- Why ML now ?
    - Algorithm foundations
    - Big data availability
    - Computational Power

BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

Example: estimating bike rental demand

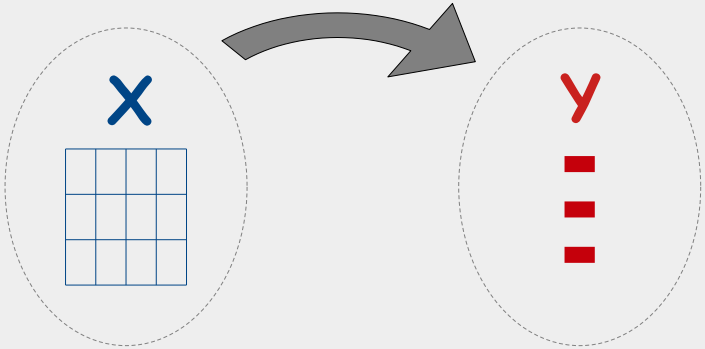
- Obj: estimate the demand of bike rental for the day/hours in different rental locations so that the management company can optimise the allocation of the resources
- Input data: temperature/rain/wind, day of the week, (hour), season, working day flag, school holiday, next day working day flag, next day school holiday, ...
- Output: bike rented at the specific location/day(hour)

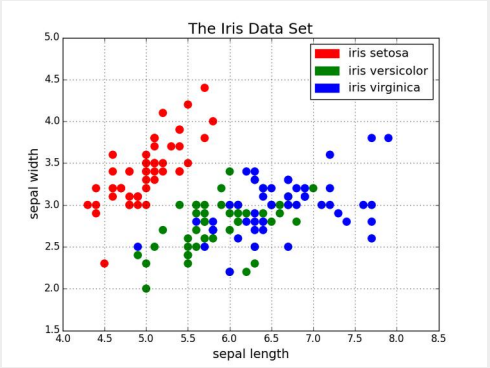
- **Statistical model**: Bike rented = $\alpha + \beta_1$*temp+β_2 * temp*wday+ ....
  - the modeller needs to make assumptions about the data and their relations, the variable to include, the covariance/independence between variables,..
  - the goodness of fit (significance) of the estimated parameters depends on the above assumptions
  - the modeller needs to provide the decision maker just the above equation and the α, β_1, β_2,... parameters
- **ML "model"**: a "generic" regression algorithm
  - the goodness of the estimated output obtained by splitting the data in the train/validation/test sets
  - the modeller need to provide the decision maker with the whole "trained" algorithm for the decision maker to be able to make predictions

BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

Three areas of ML

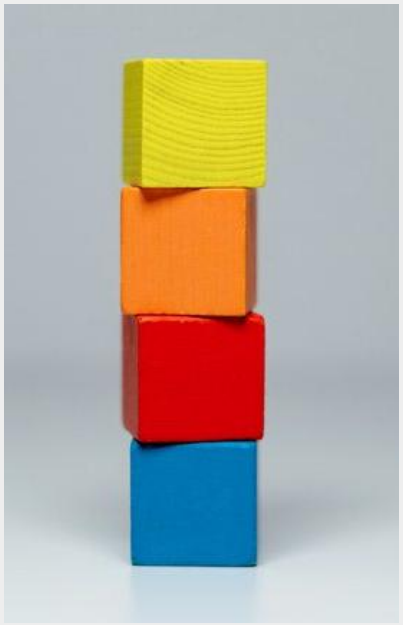
- **Supervised learning**
  - obj: predict Y ("labels" or "target": continuous → regression or categorical → classification) based on X ("features")
  - given a certain number of (X,Y) couples provided to the algorithms
  - learn solution from examples
- **Unsupervised learning**
  - exploit the "hidden" structure of some data (normally highly dimensional) without having any label provided
  - es. PCA, clustering
- **Reinforcement learning**
  - learn the "best" action an agent can make toward a certain goal given unknown "rewards" of the actions
  - e.g. board games (chess, go), self-driving robots, control systems
  - often in a physic world
  - need training with many repeated "experiments" (including many failures!)
  - exploration vs exploitation trade off

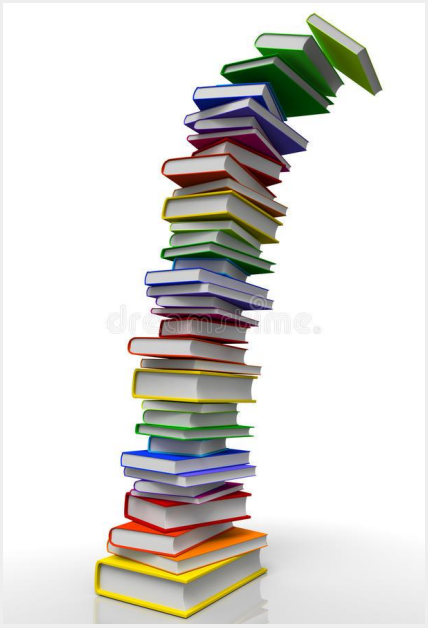




ML now

- ML algorithms differentiate for the predictive capacity, the computational efficiency (both memory and CPU), the necessity of input preprocessing (both simple data transformation and feature-engineering), the possibility to "reverse-engineer" the machinery and "understand" the relation between the X and the Y,...
- requirement for lot of data, next step is transfer knowledge from a system (problem) to an other

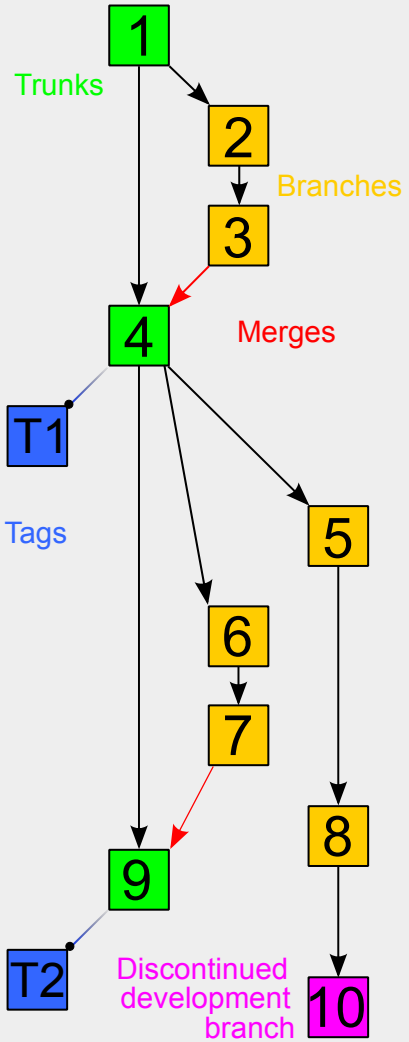




HANDS ON

Version control software

- Used to keep track of the development of a project and allow easy integration of multiple sub-projects or team contributions
- Typically they allow assigning a name/log entry to a given "screenshot in time" of project development, visualise diffs between them, create and merge branches of different development pathways, automatically manage or facilitate resolution of conflicts between versions,...
- First generation (CVS, SVN,...):
  - client-server model → the full project (history) is kept in a centralised server and individual users connect and keep in their pc one specific version
- Second generation (Git, Mercurial, Bazaar,...):
  - distributed model → each user has the whole copy (including history) of the project and exchanges its contribution with the other "remote" nodes
  - increases redundancy
  - decreases network load
  - faster operations (many operations – e.g. diff – can be fully done in local)



Set up the environment

- Create our first git repository
  - register or sign in on github.com
  - add new repository "testGit" (attention capital letters! no spaces!)
  - add readme, gitignore (Julia) and licence (MIT? GPL?)
- Install Julia (terminal): https://julialang.org/ → download → 1.7 64 bit installer → run with the option "add julia to path"
- Install Visual Studio Code: https://code.visualstudio.com/ → download → run with the options additional icons/other
- Install Julia extension:
  - search in extensions: "julia" (not "julia insider")
  - try it is working: new file -> select a language -> julia -> println("Hello World!") SHIFT+ENTER

In case of impossibility to set up the environment on your own pc an account on the lab server at https://lef-nancy.org has been set for you (username: your surname all in lower letters, psw [XXXXXX]

Help: this course forum (for those with access to it), https://discourse.julialang.org/ , StackOverflow (using the "julia" tag)

Essential git workflow

- Clone an existing repositories or start a new one: `git clone [url]` or `git init`
- ..work on your project..
- Add files: `git add [file]`
- Commit your work: `git commit -a -m ‘‘[message]’’`
- See branches: `git branch`
- Create branch: `git branch [branchname]`
- Switch to branch: `git checkout [branchname]`
- Merge a branch: `git merge [branchname]`
- Delete a local branch: `git branch -d [branchname]`
- Fetch a remote repository (without attempt to update the local repository): `git fetch`
- Fetch and try to merge from a remote repository: `git pull`
- Push your data to a remote repository: `git push`
- List all the remote repositories: `git remote`
- Get current repository status: `git status`
- Get git logs: `git log`
- See diff with HEAD: `git diff`
- See diff between any commit(s): `git diff [commithash1] [commithash2]`

Further resources
Git tutorials (shorts):
- https://git-scm.com/docs/gittutorial
- https://thenewstack.io/tutorial-git-for-absolutely-everyone

Git book (long):
- https://git-scm.com/book

A first look on git(hub)
- retrieve the repository locally

- Clone the repository locally using Visual Code Studio
  - Automatic way
    - CTRL+SHIFT+P to open the command palette and search for git:clone
    - Clone from github
    - Authorise
    - Choose the repository to clone
    - Open in new file
  - Manual way
    - open a new "git bash" terminal using "new terminal" and then the "plus"
    - git clone https://github.com/[YOUR GITHUB USERNAME]/testGit.git
    - open folder This PC > OS (C:) > Users > Your name > testGit

A first look on git(hub)

- work, commit & push

- Add a file "test.jl" → `println("Hello World!")` → ALT+ENTER → save
- Edit the README.md file
- `git status`
- `git add test.jl`
- `git status`
- `git config --global user.email "you@example.com"`
- `git config --global user.name "Your Name"`
- `git commit -a` → add a message → CTR+X → Y
- `git push` → follow online instruction to authorise
- => changes are back to github

A first look on git(hub)

- working with branches

- Edit file "test.jl" to add text on lines 4, 7 and 9, save and commit
- Open a git bash terminal
- git branch temp
- git checkout temp
- Edit test.jl on line 9, save and commit
- git checkout main
- Edit test.jl on line 5, save and commit
- git diff main temp
- git branch
- git merge temp
- git log
- git push
- git branch -d temp

Modules and packages

Structure of a module:

```
module ModuleName
export myObjects # functions, structs, and other objects that will be directly available once `using ModuleName` is typed
[...module code...]
end
```

- `using pkgOrModule`
- `import pkgOrModule: X,Y,Z`
- Module names are customary starting with a capital letter and the module content is usually not indented. Modules can be entered in the REPL as normal Julia code or in a script that is imported with `include("file.jl")`.
- There is no connection between a given file and a given module as in other languages, so the logical structure of a program can be decoupled from its actual division in files. For example, one file could contain multiple modules.
- A more common way to use modules is by loading a package that will consists, at least, of a module with the same name of the package.
- All modules are children of the module `Main`, the default module for global objects in Julia, and each module defines its own set of global names.
- Note: using and import, when they are followed with either Main.x or .x, look for a module already loaded and bring it and its exported objects into scope (for `import` only those explicitly specified). Otherwise, they do a completely different job: they expects a *package*, and the package system lookups the correct version of the module x embedded inside package x, it loads it, and it bring it and its exported objects into scope (again, for `import x` only those explicitly specified).

https://syl1.gitbook.io/julia-language-a-concise-tutorial/language-core/11-developing-julia-packages

The integrated Pkg manager

```
julia> using Pkg

julia> Pkg.add("DataFrames")
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`

julia>
```

```
(@v1.6) pkg> add DataFrames
    Updating registry at `~/.julia/registries/General`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
  Progress [========================================>]  3/3
3 dependencies successfully precompiled in 44 seconds (366 already precompiled)

(@v1.6) pkg>
```

Some useful package commands:

- `status` Retrieves a list with name and versions of locally installed packages
- `update` Updates your local index of packages and all your local packages to the latest version
- `add myPkg` Automatically downloads and installs a package
- `rm myPkg` Removes a package and all its dependent packages that has been installed automatically only for it
- `add pkgName#master` Checkouts the master branch of a package (and `free pkgName` returns to the released version)
- `add pkgName#branchName` Checkout a specific branch
- `add git@github.com:userName/pkgName.jl.git` Checkout a non registered pkg
- `free pkgName` Return to the "standard" latest compatible released version of a package

https://syl1.gitbook.io/julia-language-a-concise-tutorial/language-core/11-developing-julia-packages

Julia environments

- Packages themselves are downloaded, "installed" (precompiled) and stored in a user common directory (e.g. `/home/[username]/.julia` in Linux)
- Each project can (should!) have its own "environment"
- This is really simple and "light", as it is a simple, automatically managed file ("Manifest.toml") that list all the packages and sub-packages used in the project and their versions
  - providing this file to the "customer"/book, together with the source code, the set of inputs and using a fixed seed for the random numbers at the beginning of the script(s) guarantee reproducible analysis/research

- `cd(@__DIR__)`
- `using Pkg`
- `Pkg.activate(".")`
- `Pkg.instantiate()`

An Example of Manifest file

```
# This file is machine-generated - editing it directly is not advised

[[ArgTools]]
uuid = "0dad84c5-d112-42e6-8d28-ef12dabb789f"

[[Artifacts]]
uuid = "56f22d72-fd6d-98f1-02f0-08ddc0907c33"

[[Base64]]
uuid = "2a0f44e3-6c83-55bd-87e4-b1978d98bd5f"

[[Compat]]
deps = ["Base64", "Dates", "DelimitedFiles", "Distributed", "InteractiveUtils",
"LibGit2", "Libdl", "LinearAlgebra", "Markdown", "Mmap", "Pkg", "Printf", "REPL",
"Random", "SHA", "Serialization", "SharedArrays", "Sockets", "SparseArrays",
"Statistics", "Test", "UUIDs", "Unicode"]
git-tree-sha1 = "727e463cfebd0c7b999bbf3e9e7e16f254b94193"
uuid = "34da2185-b29b-5c13-b0c7-acf172513d20"
version = "3.34.0"

[[Crayons]]
git-tree-sha1 = "3f71217b538d7aaee0b69ab47d9b7724ca8afa0d"
uuid = "a8cc5b0e-0ffa-5ad4-8c14-923d3ee1735f"
version = "4.0.4"

[[DataAPI]]
git-tree-sha1 = "ee400abb2298bd13bfc3df1c412ed228061a2385"
uuid = "9a962f9c-6df0-11e9-0e5d-c546b8b5ee8a"
version = "1.7.0"

[[DataFrames]]
deps = ["Compat", "DataAPI", "Future", "InvertedIndices",
"IteratorInterfaceExtensions", "LinearAlgebra", "Markdown", "Missings", "PooledArrays",
"PrettyTables", "Printf", "REPL", "Reexport", "SortingAlgorithms", "Statistics",
"TableTraits", "Tables", "Unicode"]
git-tree-sha1 = "d785f42445b63fc86caa08bb9a9351008be9b765"
uuid = "a93c6f00-e57d-5684-b7b6-d8193f3e46c0"
version = "1.2.2"
```

BETA Bureau d'économie théorique et appliquée
AgroParisTech Talents d'une planète soutenable

ML EXAMPLES

ML examples

- Digits recognition
  - will the algorithm recognise your hand-written digits ? (i.e. why captchas are a matter of the past)
- A **regression** task: the prediction of bike sharing demand
  - The task is to estimate the influence of several variables (like the weather, the season, the day of the week..) on the demand of shared bicycles, so that the authority in charge of the service can organise the service in the best way.
- A **classification** task when labels are known - determining the country of origin of cars given the cars characteristics
  - In this exercise we have some car technical characteristics (mpg, horsepower,weight, model year...) and the country of origin and we would like to create a model such that the country of origin can be accurately predicted given the technical characteristics. As the information to predict is a multi-class one, this is a classification task. It is a challenging exercise due to the simultaneous presence of three factors: (1) presence of missing data; (2) unbalanced data - 254 out of 406 cars are US made; (3) small dataset.
- A **clustering** task: the prediction of plant species from floral measures (the iris dataset)
  - The task is to estimate the species of a plant given some floral measurements. It use the classical "Iris" dataset. Note that in this example we are using clustering approaches, so we try to understand the "structure" of our data, without relying to actually knowing the true labels ("classes" or "factors"). However we have chosen a dataset for which the true labels are actually known, so to compare the accuracy of the algorithms we use, but these labels will not be used during the algorithms training.

https://sylvaticus.github.io/BetaML.jl/stable/ → Tutorials