Programming

optimisation and operations research algorithms with Julia

for Business Tasks

Prof. Dr. Xavier Gandibleux

Université de Nantes
Département Informatique – Faculté des Sciences et Techniques
France

Lesson 1 – May-June 2022

version 1.7.x and later

Why Julia?

Discovered in 2016 (v0.4.5), it became my favourite language:

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   ⇒ High-level and high-performance!

   ⇒ Extremely good ratio between coding time and running time.

   ⇒ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple M-series -experimental-, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   ⇒ Ideal for the development of efficient open-source app

Why Julia?

Discovered in 2016 (v0.4.5), it became my favourite language:

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   ⇒ High-level and high-performance!

   ⇒ Extremely good ratio between coding time and running time.

   ⇒ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple M-series -experimental-, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   ⇒ Ideal for the development of efficient open-source app

Why Julia?

Discovered in 2016 (v0.4.5), it became my favourite language:

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   $\Rightarrow$ High-level and high-performance!

   $\Rightarrow$ Extremely good ratio between coding time and running time.

   $\Rightarrow$ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple M-series -experimental-, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   $\Rightarrow$ Ideal for the development of efficient open-source app

Why Julia?

Discovered in 2016 (v0.4.5), it became my favourite language:

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   $\Rightarrow$ High-level and high-performance!

   $\Rightarrow$ Extremely good ratio between coding time and running time.

   $\Rightarrow$ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple M-series -experimental-, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   $\Rightarrow$ Ideal for the development of efficient open-source app

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.7.2 (February 6, 2022)
- Communities:
  - Over 7,400 Julia packages https://juliahub.com/ui/Packages
  - Optimization https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 https://juliacon.org/
  - 27th to 29th of July, 2022: free, online and everywhere
- Structures:
  - JuliaComputing http://juliacomputing.com
  - JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.7.2 (February 6, 2022)
- Communities:
  - Over 7,400 Julia packages https://juliahub.com/ui/Packages
  - Optimization https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 https://juliacon.org/
  - 27th to 29th of July, 2022: free, online and everywhere
- Structures:
  - JuliaComputing http://juliacomputing.com
  - JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.7.2 (February 6, 2022)
- Communities:
  - Over 7,400 Julia packages https://juliahub.com/ui/Packages
  - Optimization https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 https://juliacon.org/
  - 27th to 29th of July, 2022: free, online and everywhere
- Structures:
  - JuliaComputing http://juliacomputing.com
  - JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.7.2 (February 6, 2022)
- Communities:
  - Over 7,400 Julia packages https://juliahub.com/ui/Packages
  - Optimization https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 https://juliacon.org/
  - 27th to 29th of July, 2022: free, online and everywhere
- Structures:
  - JuliaComputing http://juliacomputing.com
  - JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

Key concepts

- Julia has an LLVM-based Just-in-time compilation (JIT)
- Compilation down to native code (running on CPU and GPU, ...)
- Julia is dynamically typed, provides multiple dispatch
- Julia has many built-in mathematical functions
- A built-in package manager
- Designed for parallelism and distributed computation

Resources

Some URL to know:

- Official website
  `https://julialang.org/`
- Official documentation (online and pdf)
  `https://docs.julialang.org/`
- Julia channel on *youtube*
  `https://www.youtube.com/channel/UC9IuUwwE2xdjQUT_LMLONoA/featured`
- Questions and discussion on *discourse*
  `https://discourse.julialang.org/`
- Book online « Think Julia: How to Think Like a Computer Scientist »
  `https://benlauwens.github.io/ThinkJulia.jl/latest/book.html`
- Julia (v1.6.3) on *replit*
  `https://replit.com/languages/julia`

Resources

Some URL to know:

- Official website
  `https://julialang.org/`
- Official documentation (online and pdf)
  `https://docs.julialang.org/`
- Julia channel on *youtube*
  `https://www.youtube.com/channel/UC9IuUwwE2xdjQUT_LMLONoA/featured`
- Questions and discussion on *discourse*
  `https://discourse.julialang.org/`
- Book online « Think Julia: How to Think Like a Computer Scientist »
  `https://benlauwens.github.io/ThinkJulia.jl/latest/book.html`
- Julia (v1.6.3) on *replit*
  `https://replit.com/languages/julia`

Getting started

Software environment

Software environment: check list

Base:

- Operating system: **MacOS**, Windows, Linux (ubuntu), etc.
- Browser: **Firefox**, Safari, Chrome, etc.

Apps to install on your computer:

- Language: **Julia 1.7.2**
- Editor: **Visual Studio Code (VSCode)**, Emacs, etc.

Apps to install for optimisation needs:

- Open-source MILP solver: **GLPK**, COIN-OR, etc.
- Commercial MILP solver: **Gurobi**, Cplex, etc.

Apps used (installation not mandatory but recommanded):

- interacting with repository: **GitHub Desktop**

Software environment: check list

Base:

- Operating system: **MacOS**, Windows, Linux (ubuntu), etc.
- Browser: **Firefox**, Safari, Chrome, etc.

Apps to install on your computer:

- Language: **Julia 1.7.2**
- Editor: **Visual Studio Code (VSCode)**, Emacs, etc.

Apps to install for optimisation needs:

- Open-source MILP solver: **GLPK**, COIN-OR, etc.
- Commercial MILP solver: **Gurobi**, Cplex, etc.

Apps used (installation not mandatory but recommanded):

- interacting with repository: **GitHub Desktop**

Software environment: check list

Base:

- Operating system: **MacOS**, Windows, Linux (ubuntu), etc.
- Browser: **Firefox**, Safari, Chrome, etc.

Apps to install on your computer:

- Language: **Julia 1.7.2**
- Editor: **Visual Studio Code (VSCode)**, Emacs, etc.

Apps to install for optimisation needs:

- Open-source MILP solver: **GLPK**, COIN-OR, etc.
- Commercial MILP solver: **Gurobi**, Cplex, etc.

Apps used (installation not mandatory but recommanded):

- interacting with repository: **GitHub Desktop**

Software environment: check list

Base:

- Operating system: **MacOS**, Windows, Linux (ubuntu), etc.
- Browser: **Firefox**, Safari, Chrome, etc.

Apps to install on your computer:

- Language: **Julia 1.7.2**
- Editor: **Visual Studio Code (VSCode)**, Emacs, etc.

Apps to install for optimisation needs:

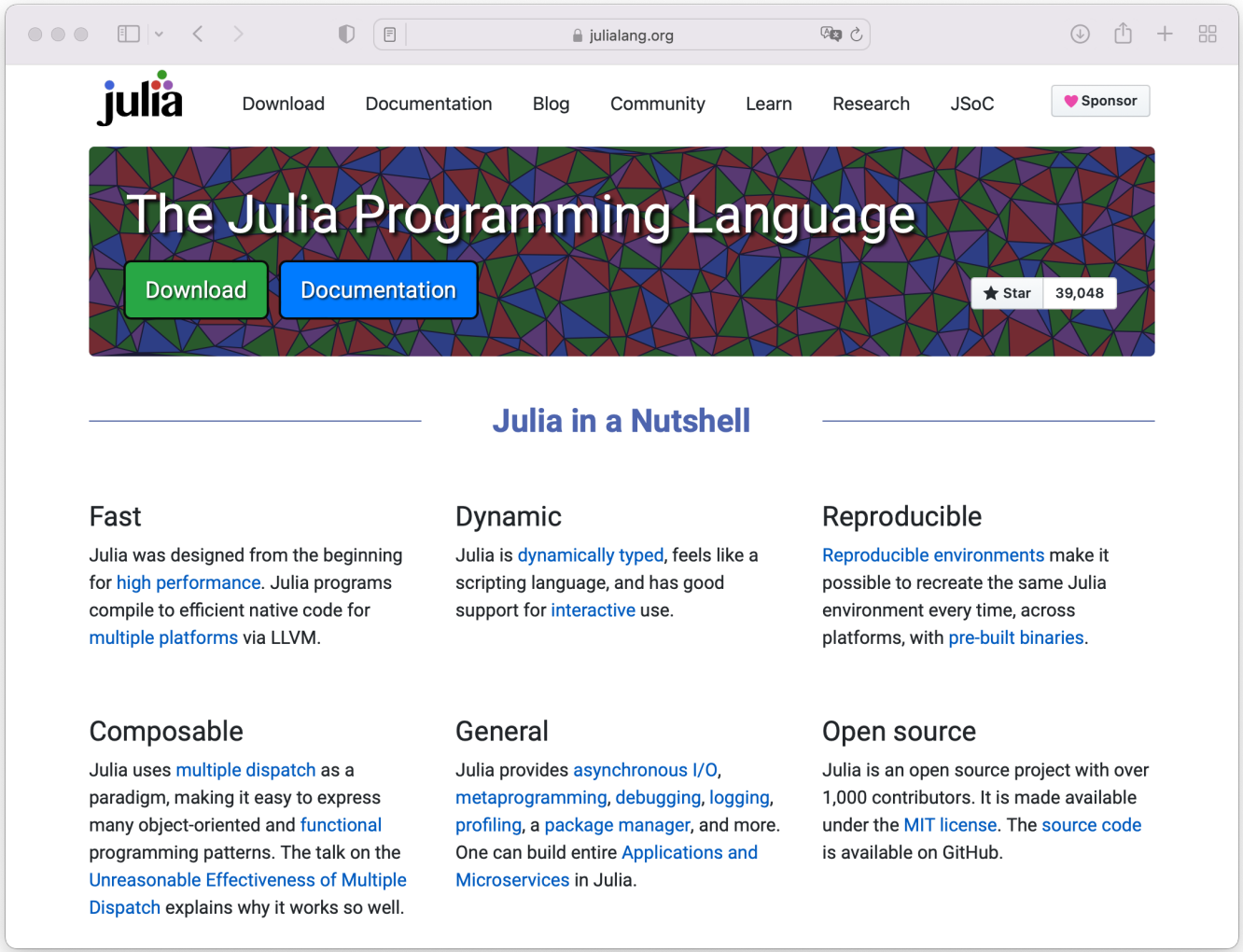
- Open-source MILP solver: **GLPK**, COIN-OR, etc.
- Commercial MILP solver: **Gurobi**, Cplex, etc.

Apps used (installation not mandatory but recommanded):

- interacting with repository: **GitHub Desktop**

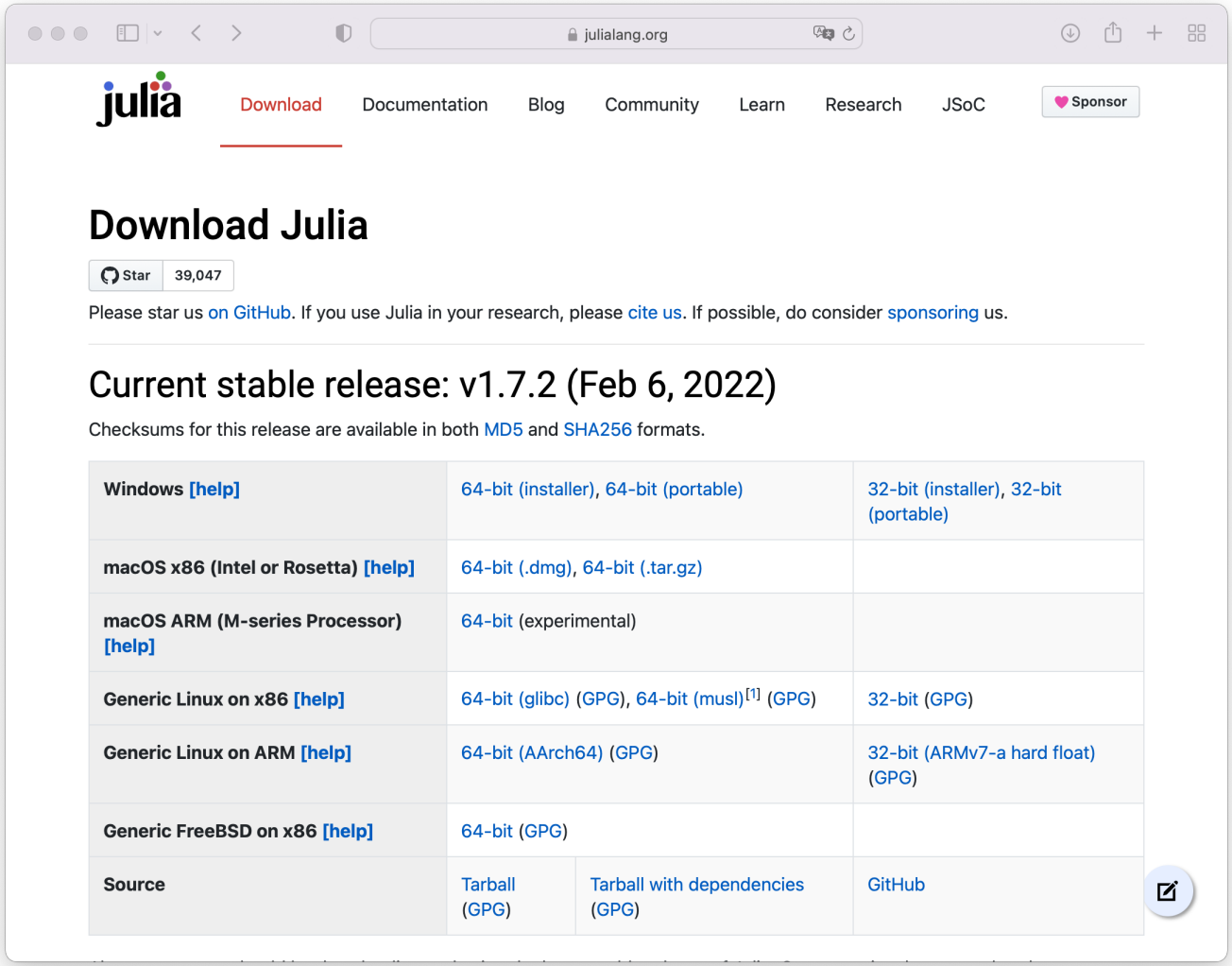
Language: Julia 1.7.2

`https://julialang.org/`



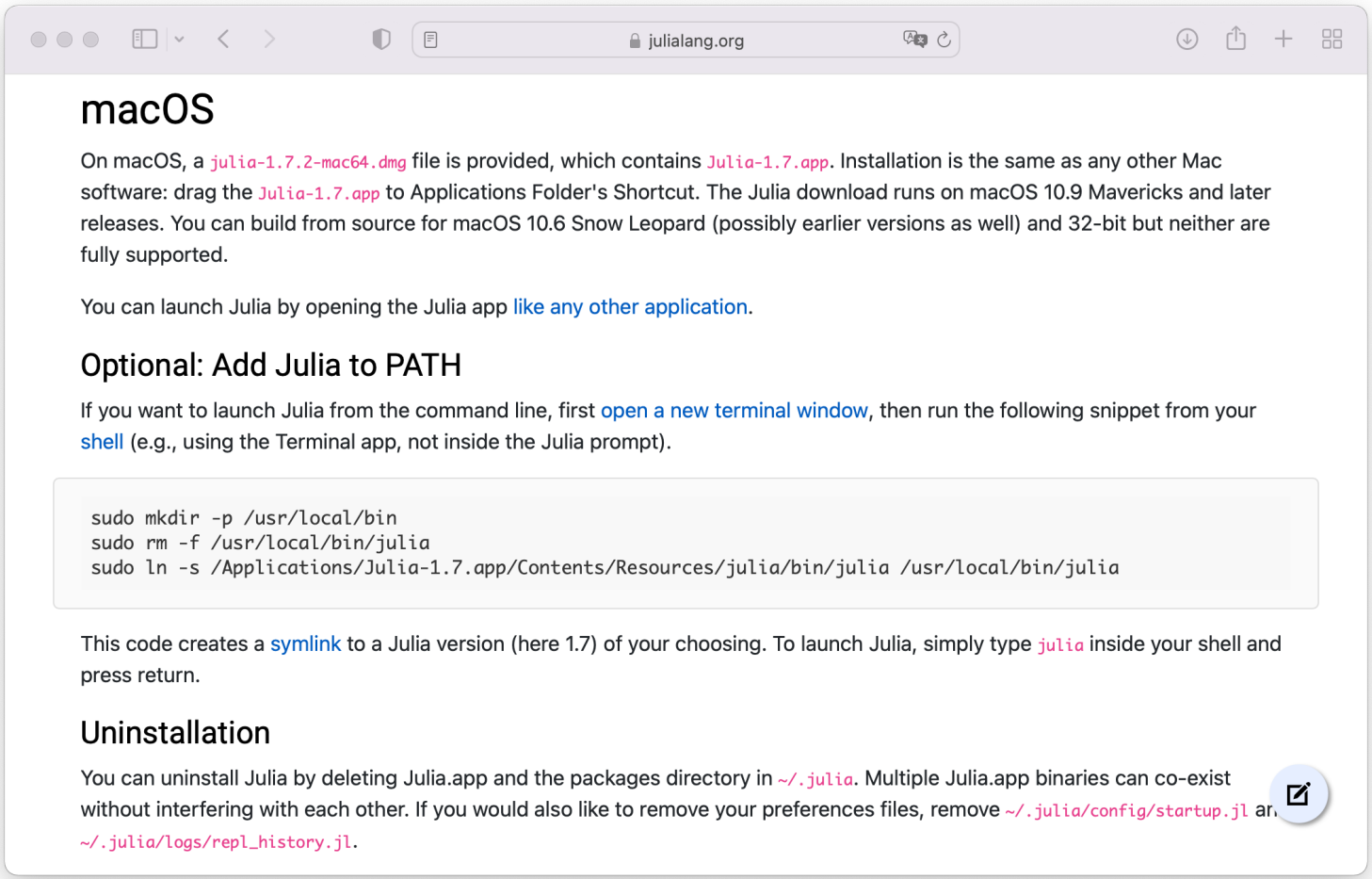
Language: Julia 1.7.2

`https://julialang.org/downloads/`



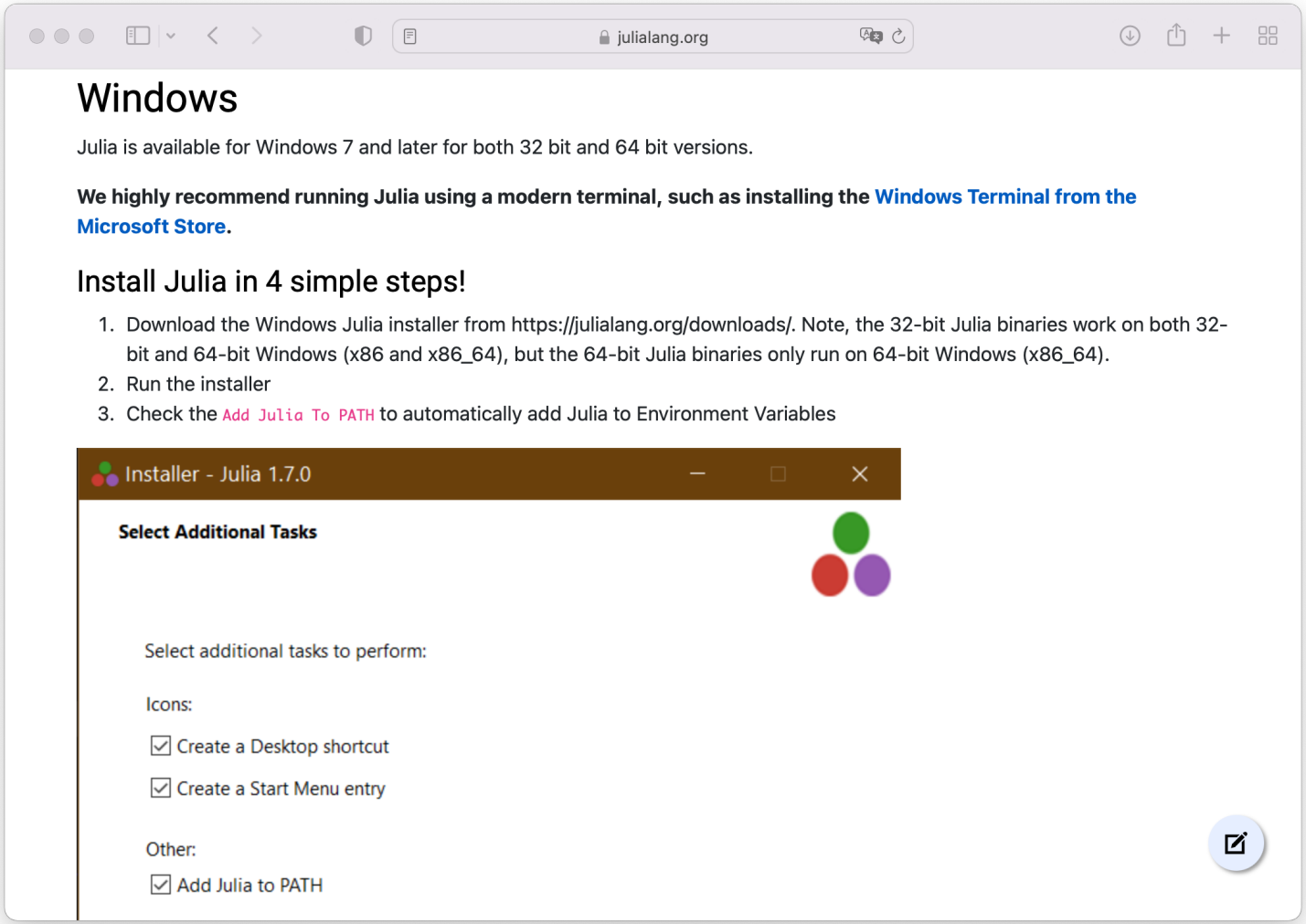
Language: Julia 1.7.2

`https://julialang.org/downloads/`



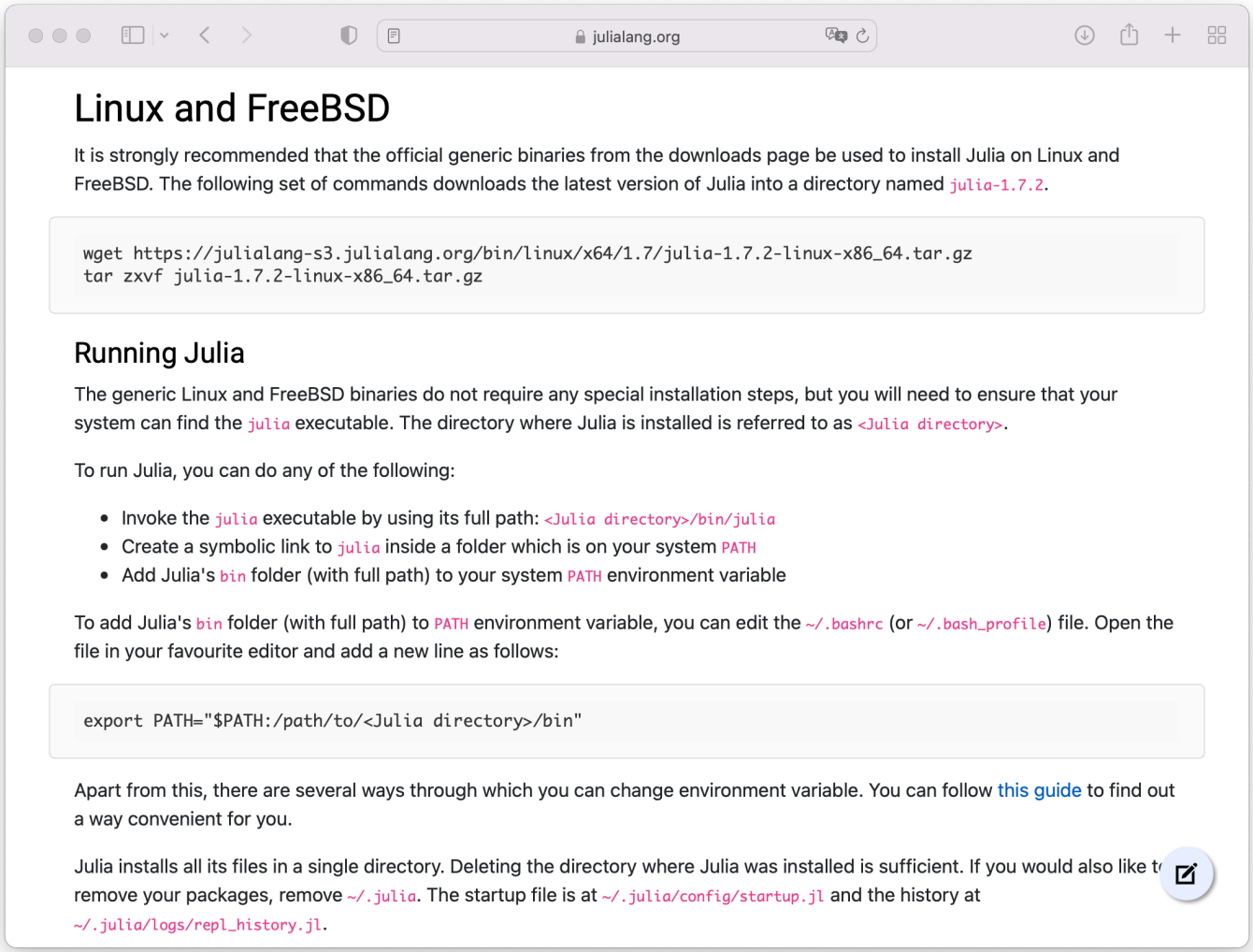
Language: Julia 1.7.2

`https://julialang.org/downloads/`



Language: Julia 1.7.2

`https://julialang.org/downloads/`

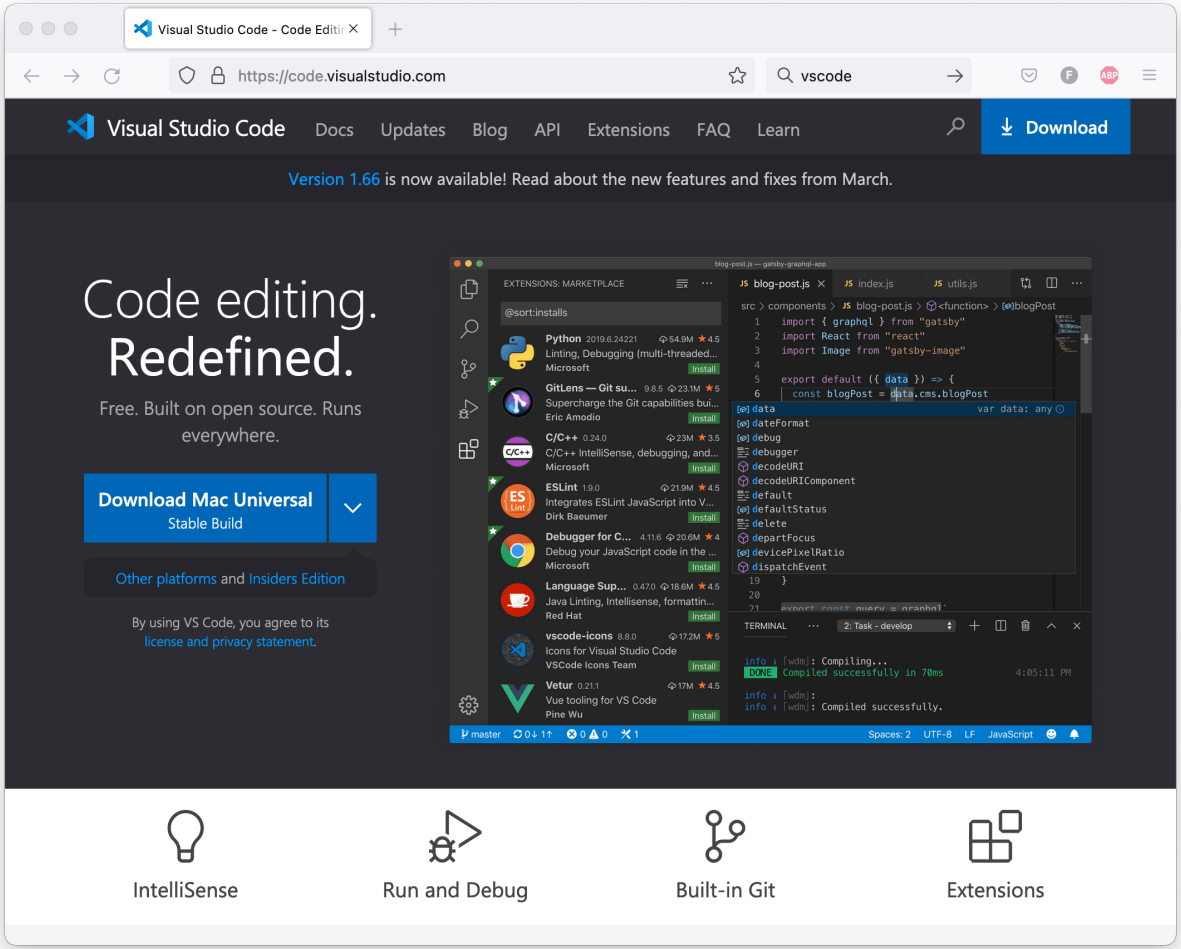


Language: Julia 1.7.2

Just click on the app icon

Editor: Visual Studio Code

`https://code.visualstudio.com/`

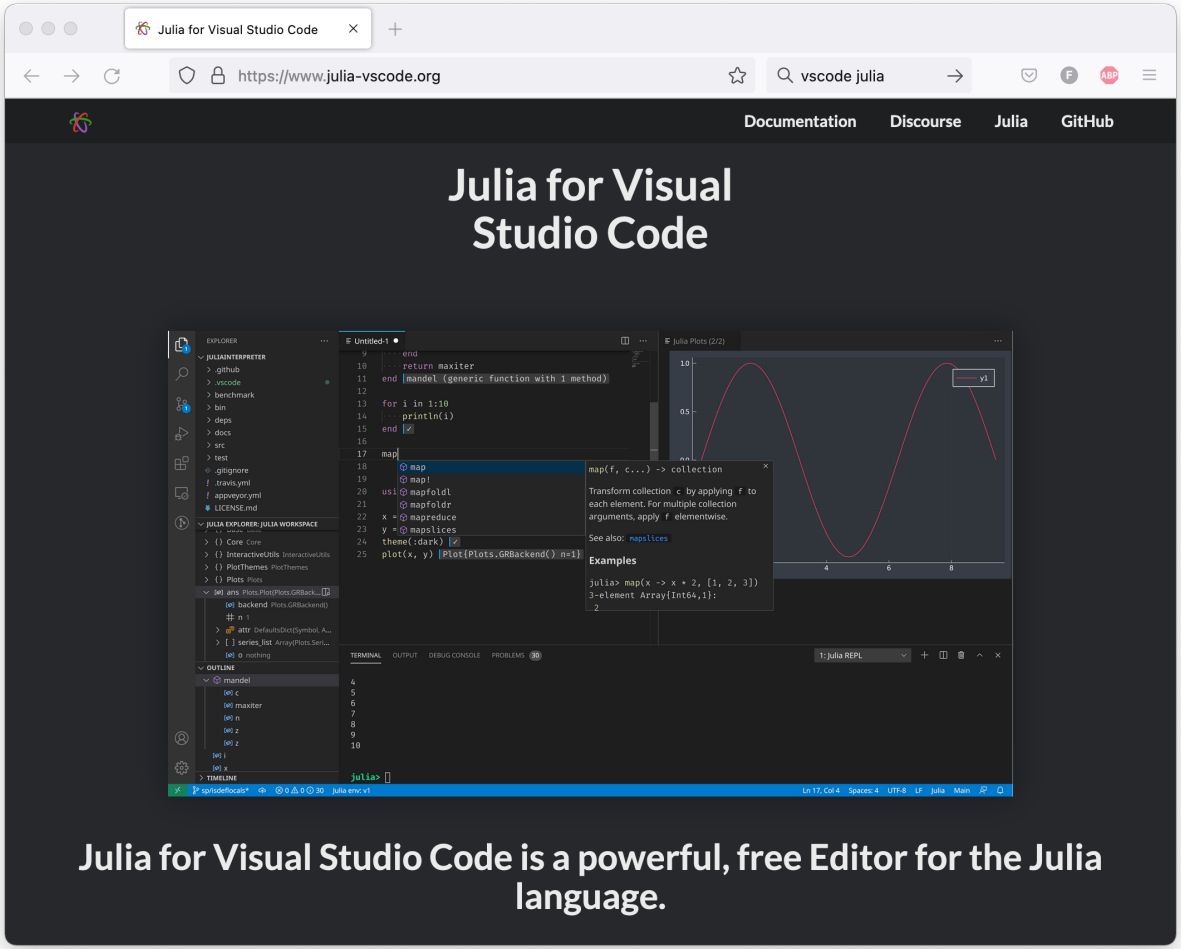


extended for Julia

`https://www.julia-vscode.org/`

Editor: Visual Studio Code

`https://code.visualstudio.com/`



extended for Julia

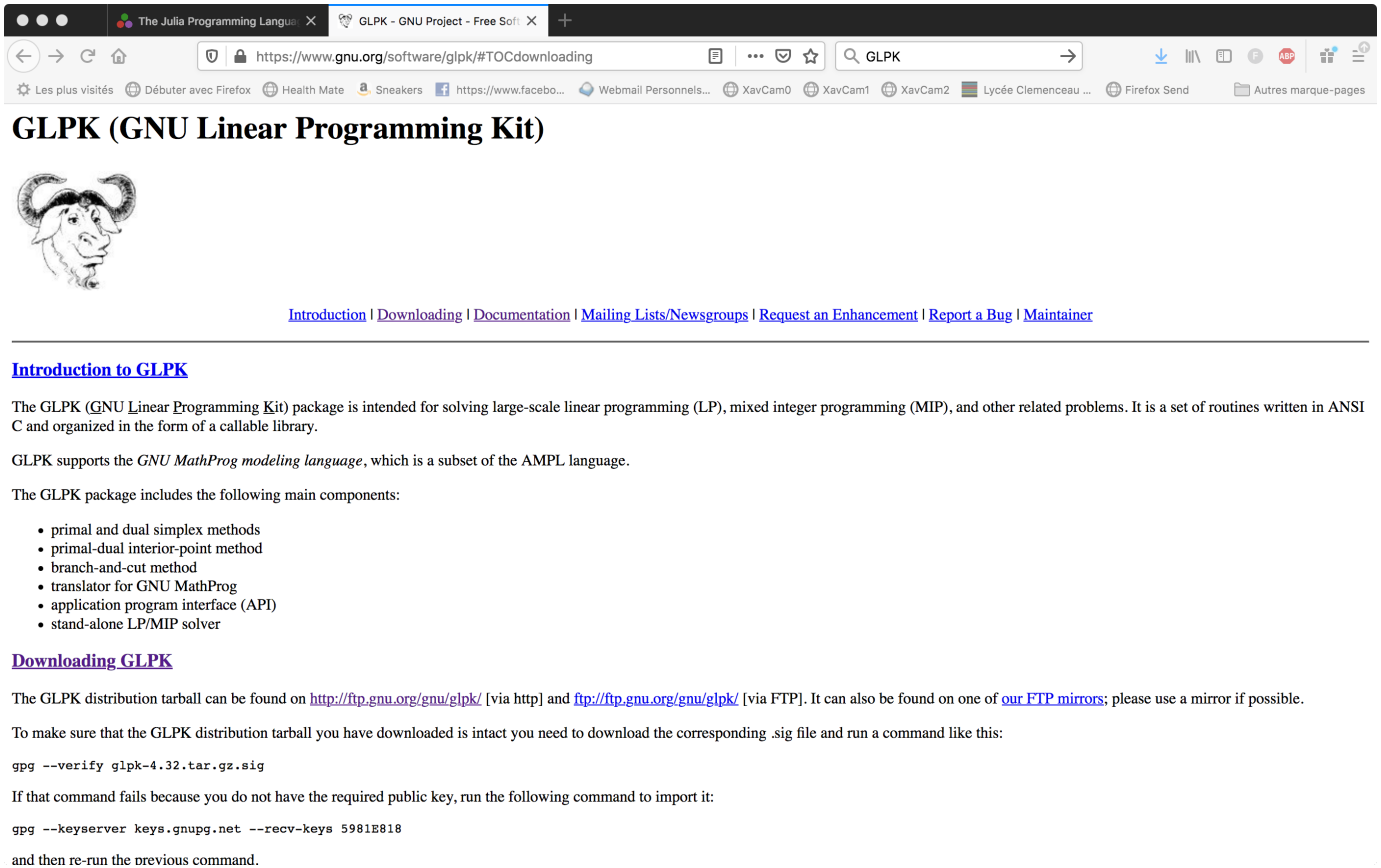
`https://www.julia-vscode.org/`

Editor: Visual Studio Code

`https://code.visualstudio.com/`

Open Source MILP solver: GLPK

https://www.gnu.org/software/glpk/



GLPK (GNU Linear Programming Kit)

Introduction | Downloading | Documentation | Mailing Lists/Newsgroups | Request an Enhancement | Report a Bug | Maintainer

Introduction to GLPK

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

GLPK supports the *GNU MathProg modeling language*, which is a subset of the AMPL language.

The GLPK package includes the following main components:

- primal and dual simplex methods
- primal-dual interior-point method
- branch-and-cut method
- translator for GNU MathProg
- application program interface (API)
- stand-alone LP/MIP solver

Downloading GLPK

The GLPK distribution tarball can be found on http://ftp.gnu.org/gnu/glpk/ [via http] and ftp://ftp.gnu.org/gnu/glpk/ [via FTP]. It can also be found on one of our FTP mirrors; please use a mirror if possible.

To make sure that the GLPK distribution tarball you have downloaded is intact you need to download the corresponding .sig file and run a command like this:

```
gpg --verify glpk-4.32.tar.gz.sig
```

If that command fails because you do not have the required public key, run the following command to import it:

```
gpg --keyserver keys.gnupg.net --recv-keys 5981E818
```

and then re-run the previous command.

Installing GLPK

On macOS

The easy way to install GLPK with one of these two package manager:

- homebrew
  - install homebrew: `https://brew.sh/`
  - install GLPK: `https://formulae.brew.sh/formula/glpk`
- macport
  - install macport: `https://www.macports.org/`
  - install GLPK: `https://ports.macports.org/port/glpk/`

On windows

Follow the instructions given here:

- `http://winglpk.sourceforge.net/`

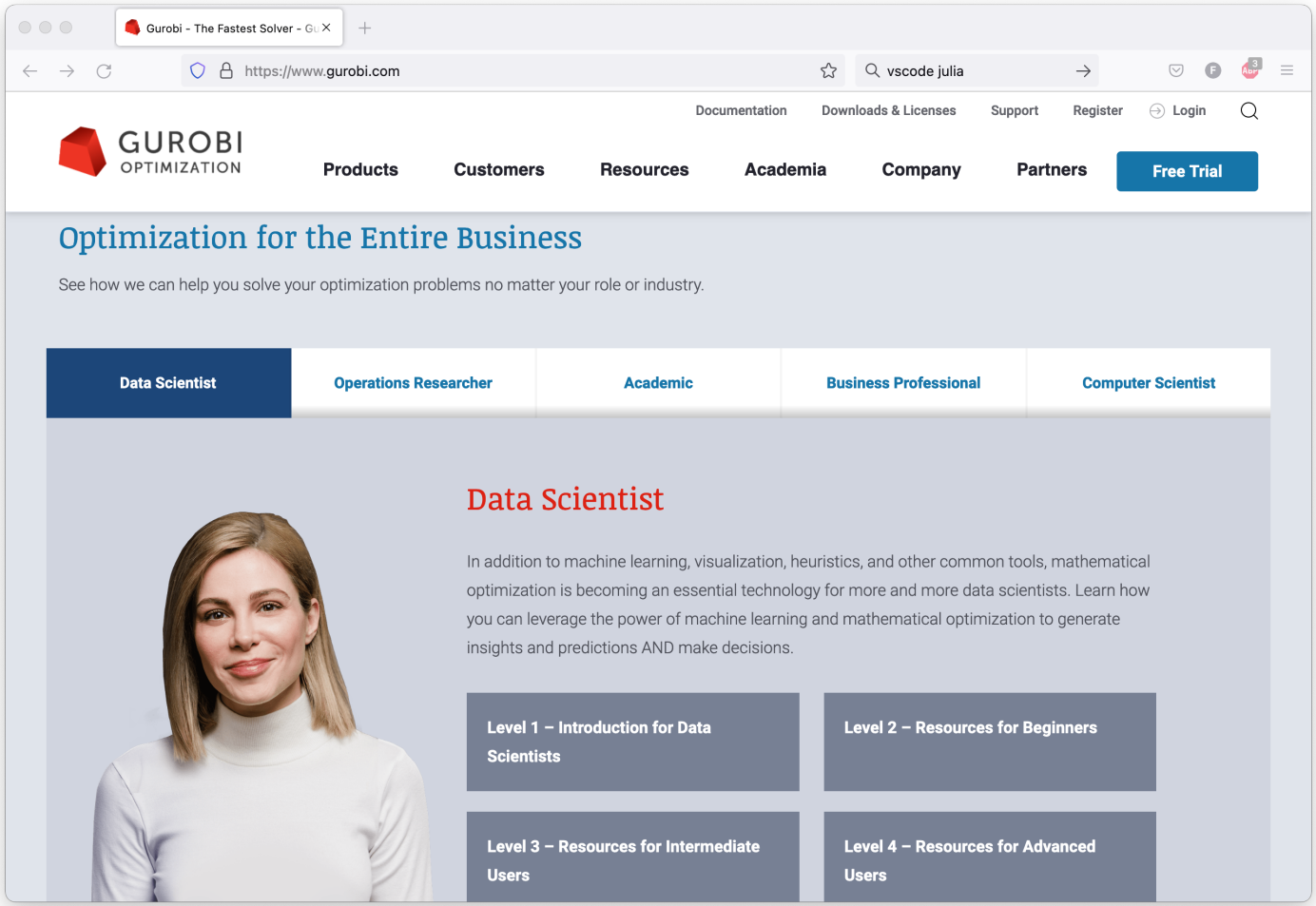
On linux (ubuntu)

Follow the instructions given here:

- `https://howtoinstall.co/en/glpk`

Commercial MILP solver: Gurobi

`https://www.gurobi.com/`



Installing Gurobi (academic license)

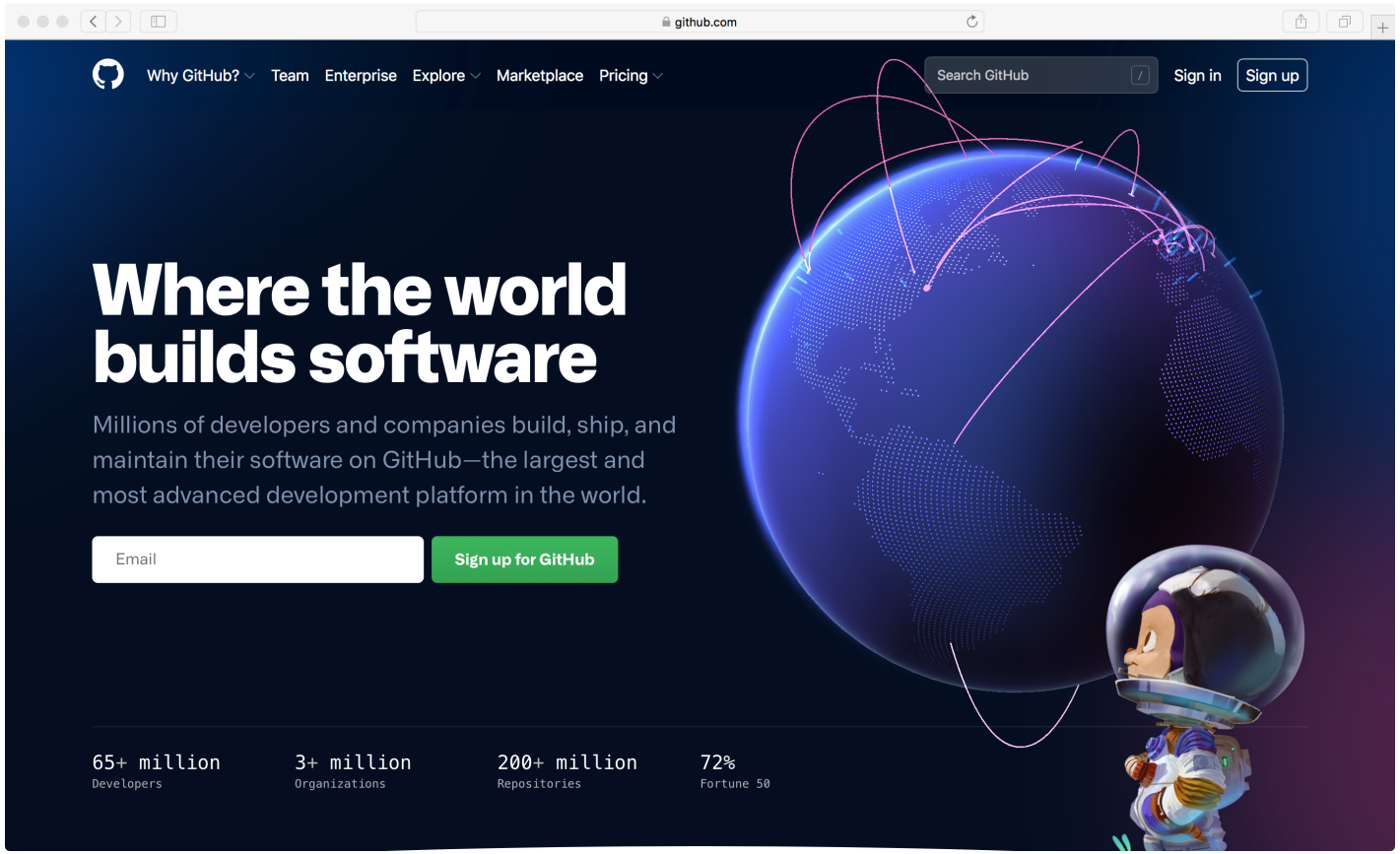
Gurobi software downloads and license center:
`https://www.gurobi.com/downloads/`

Video for

- macOS:
  `https://www.youtube.com/watch?v=ZcL-NmckTxQ`
- windows:
  `https://www.youtube.com/watch?v=fQVxuWOiPpI&t=0s`
- linux (ubuntu):
  `https://www.youtube.com/watch?v=yNmeG6Wom1o&t=0s`

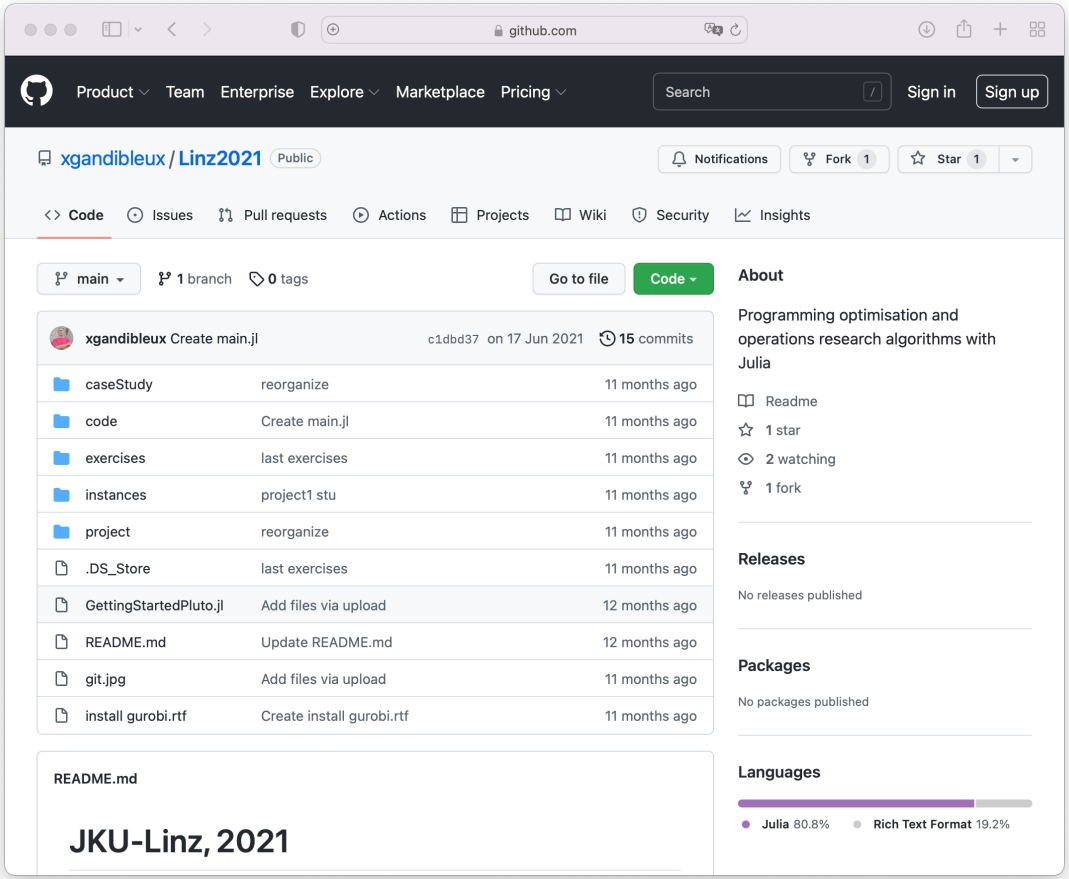
Repository: GitHub and the app GitHub Desktop

`https://github.com` and `https://desktop.github.com`



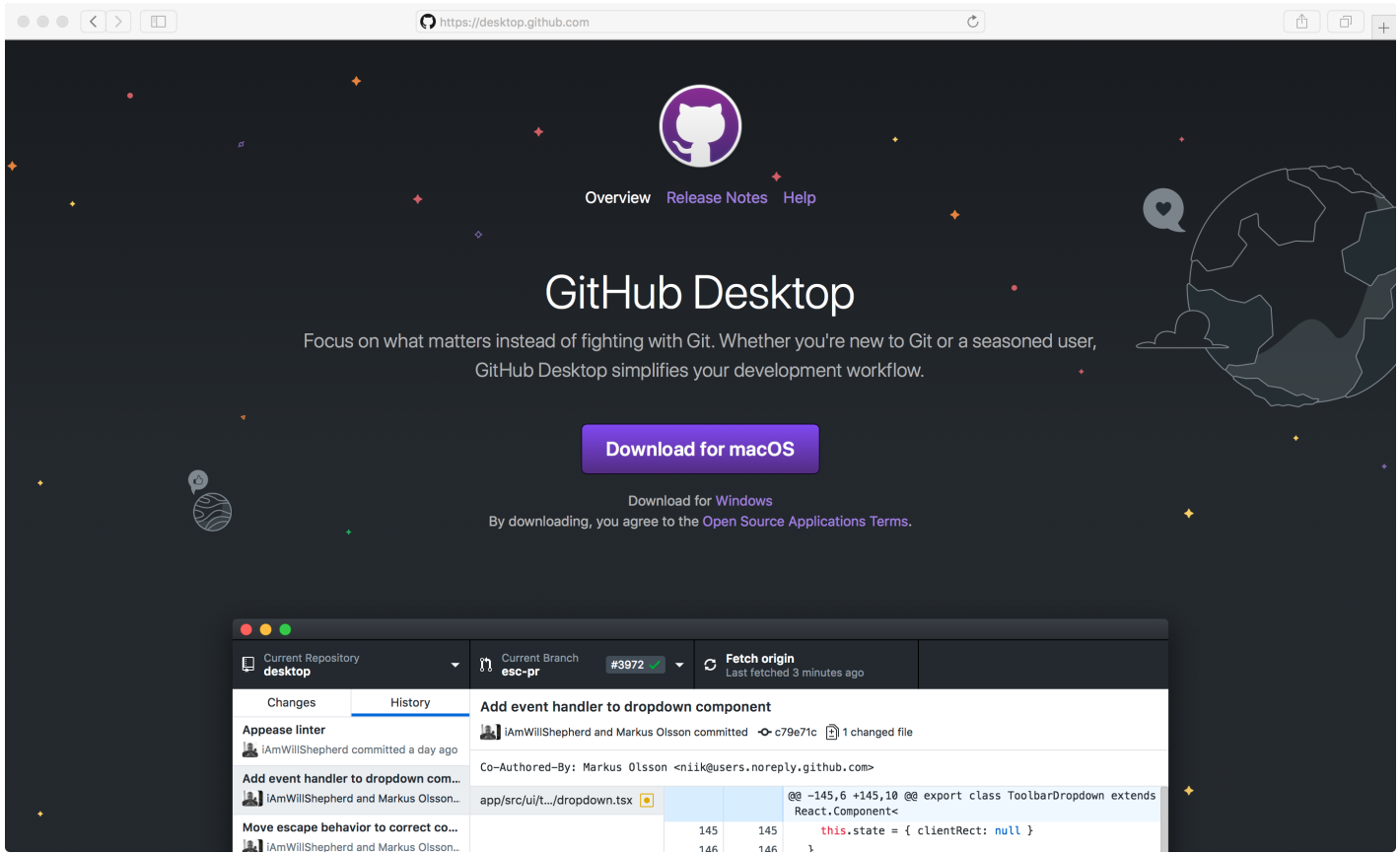
Repository: GitHub and the app GitHub Desktop

`https://github.com` and `https://desktop.github.com`



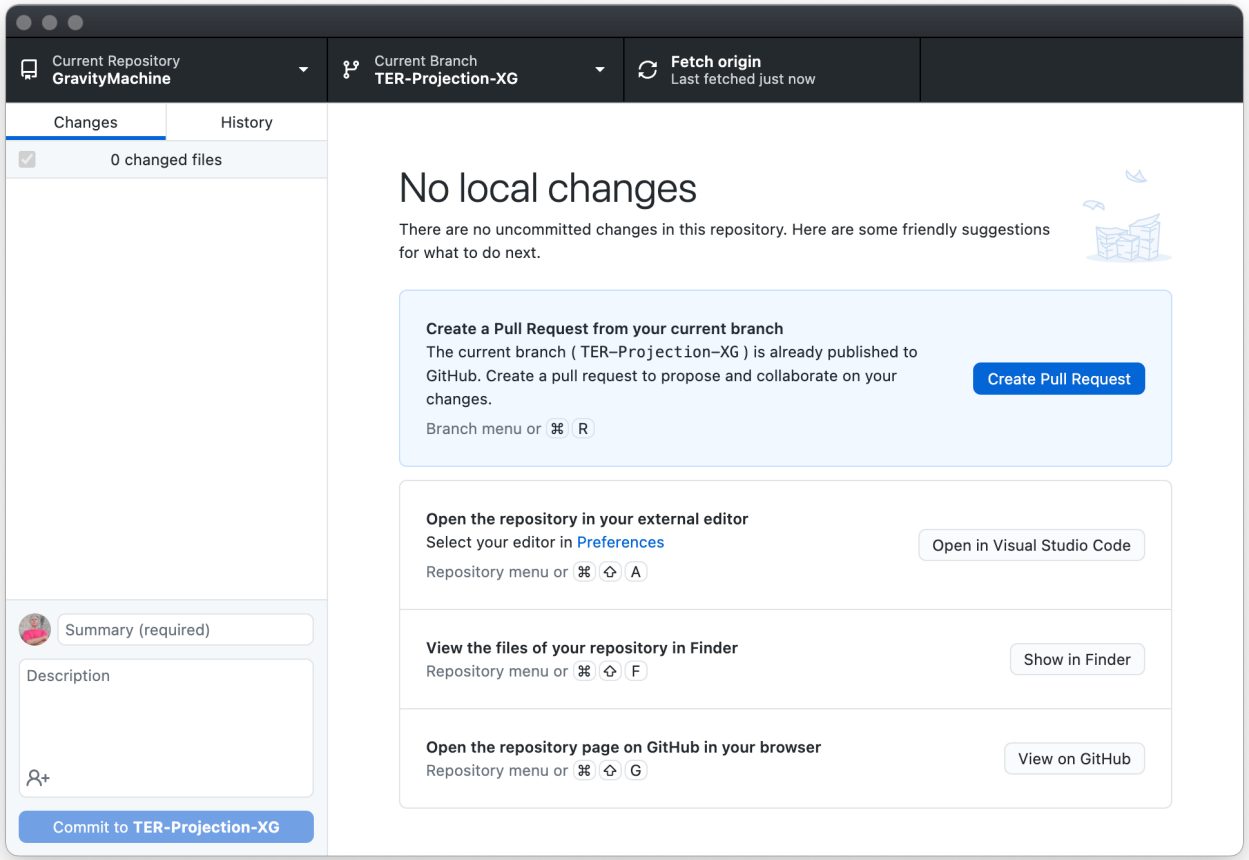
Repository: GitHub and the app GitHub Desktop

`https://github.com` and `https://desktop.github.com`



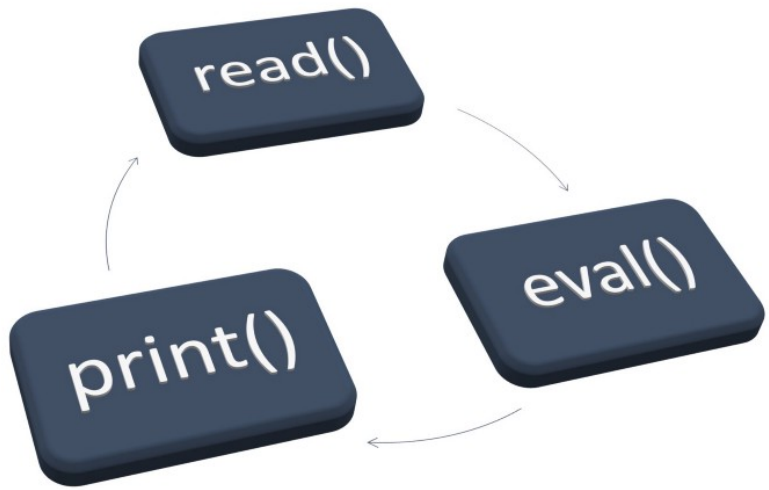
Repository: GitHub and the app GitHub Desktop

`https://github.com` and `https://desktop.github.com`



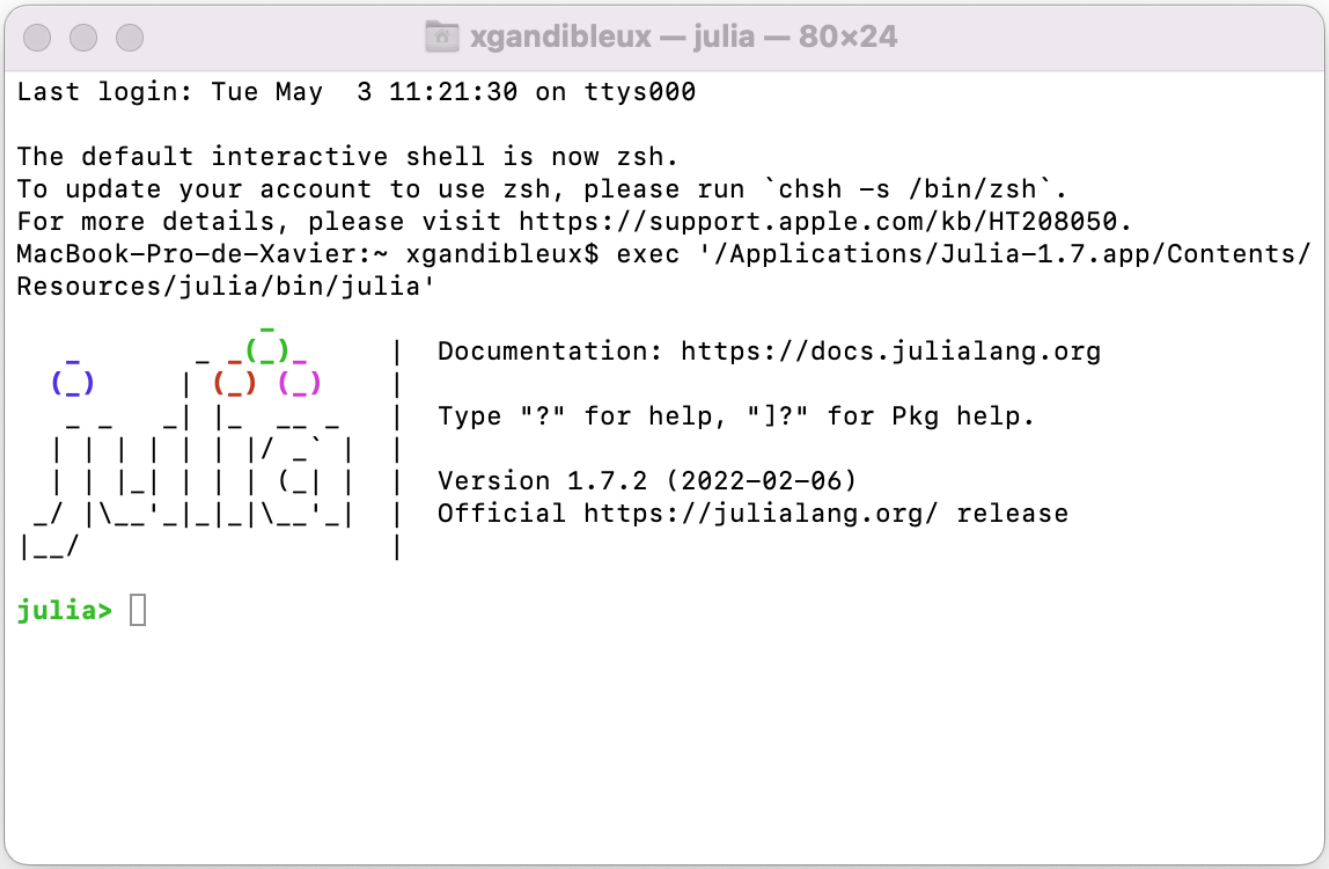
Getting started

Read Evaluate Print Loop



REPL: normal mode

Just click on the app icon



REPL: normal mode (commands)

Some commands to know:

Load (and run) a program	`include("filename.jl")`
Interrupt execution	`[Ctrl] + [C]`
Recall last result	`ans`
Previous-run command	`Up` key
Search backwards into history	`[Ctrl] + [r]`
Tab-completion	characters `+ [Tab]`
Clear screen	`[Ctrl] + [L]`
Exit REPL	`exit()` or `[Ctrl] + [D]`

REPL: normal mode (commands)

Some commands to know:

Load (and run) a program	`include("filename.jl")`
Interrupt execution	`[Ctrl] + [C]`
Recall last result	`ans`
Previous-run command	`Up` key
Search backwards into history	`[Ctrl] + [r]`
Tab-completion	characters `+ [Tab]`
Clear screen	`[Ctrl] + [L]`
Exit REPL	`exit()` or `[Ctrl] + [D]`

REPL: normal mode (commands)

Some commands to know:

Load (and run) a program	`include("filename.jl")`
Interrupt execution	`[Ctrl] + [C]`
Recall last result	`ans`
Previous-run command	`Up` key
Search backwards into history	`[Ctrl] + [r]`
Tab-completion	characters `+ [Tab]`
Clear screen	`[Ctrl] + [L]`
Exit REPL	`exit()` or `[Ctrl] + [D]`

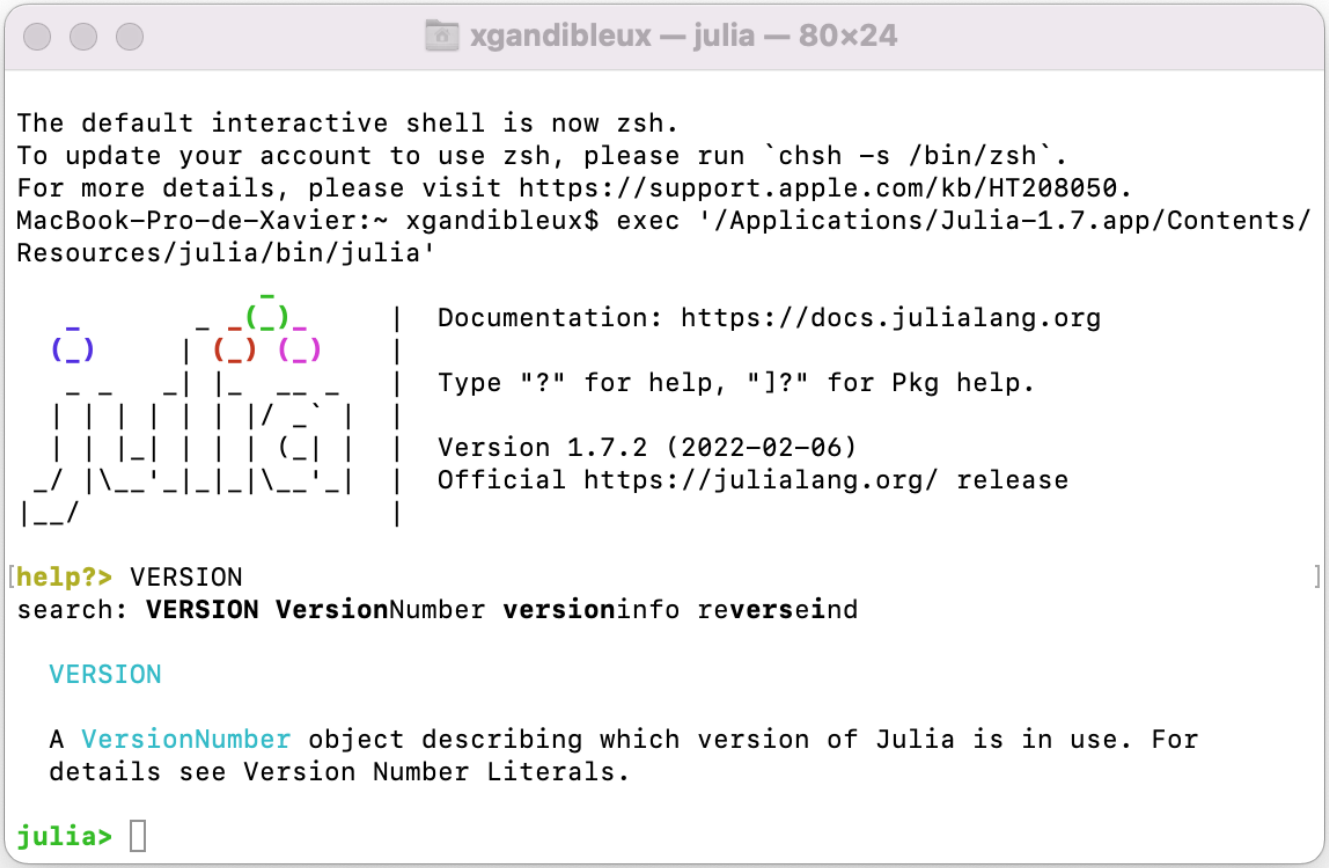
REPL: normal mode (commands)

Some commands to know:

Load (and run) a program	`include("filename.jl")`
Interrupt execution	`[Ctrl] + [C]`
Recall last result	`ans`
Previous-run command	`Up` key
Search backwards into history	`[Ctrl] + [r]`
Tab-completion	characters `+ [Tab]`
Clear screen	`[Ctrl] + [L]`
Exit REPL	`exit()` or `[Ctrl] + [D]`

REPL: help mode

```
julia> ?
```



REPL: shell mode

```
julia> ;
```

Return to normal mode: `[Backspace]` on empty line

REPL: package manager mode

```
julia> ]
```

Return to normal mode: `[Backspace]` on empty line

REPL: package manager mode (commands)

Some commands to know:

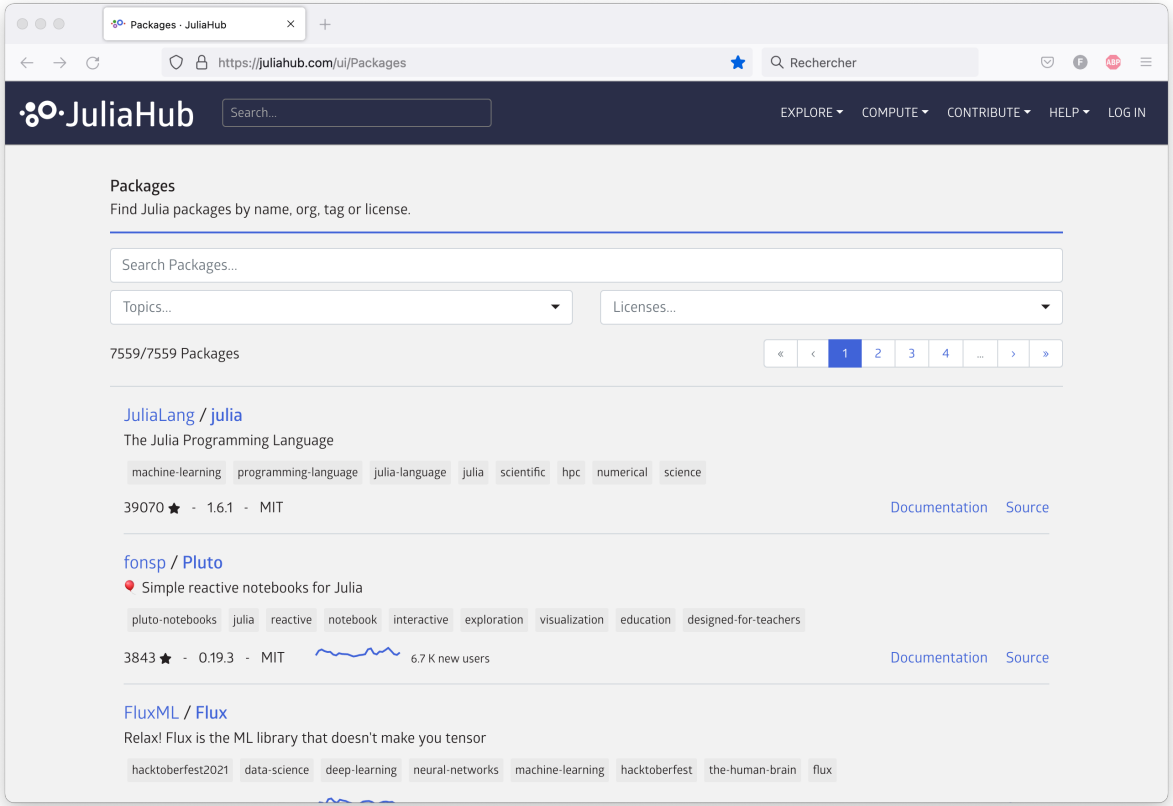
add package(s) to project	`add`
remove package(s) from project	`remove` or `rm`
summarize contents	`status` or `st`
update packages	`update` or `up`
help online	`help`

Packages

Packages

Visual interface for exploring the packages:

Julia hub `https://juliahub.com/ui/Packages`



Alternative:

Julia Packages https://juliapackages.com/

Packages

A selection of useful packages:

Standard library:

`Pkg`	primitives for managing packages
`Random`	primitives for random number
`LinearAlgebra`	primitives for linear algebra
`Printf`	primitives for C-like format to display on screen

External library:

`PyPlot`	Plotting for Julia based on matplotlib
`Plots`	a high-level plotting package
`JuMP`	Modeling language for Mathematical Optimization
`GLPK`	interface for GLPK solver
`Gurobi`	interface for Gurobi solver
`CPLEX`	interface for the CPLEX solver
`vOptGeneric`	Multiobjective linear optimization solver; generic problems
`vOptSpecific`	Multiobjective linear optimization solver; specific problems

Packages

A selection of useful packages:

Standard library:

`Pkg`	primitives for managing packages
`Random`	primitives for random number
`LinearAlgebra`	primitives for linear algebra
`Printf`	primitives for C-like format to display on screen

External library:

`PyPlot`	Plotting for Julia based on matplotlib
`Plots`	a high-level plotting package
`JuMP`	Modeling language for Mathematical Optimization
`GLPK`	interface for GLPK solver
`Gurobi`	interface for Gurobi solver
`CPLEX`	interface for the CPLEX solver
`vOptGeneric`	Multiobjective linear optimization solver; generic problems
`vOptSpecific`	Multiobjective linear optimization solver; specific problems

Packages

A selection of useful packages:

Standard library:

`Pkg`	primitives for managing packages
`Random`	primitives for random number
`LinearAlgebra`	primitives for linear algebra
`Printf`	primitives for C-like format to display on screen

External library:

`PyPlot`	Plotting for Julia based on matplotlib
`Plots`	a high-level plotting package
`JuMP`	Modeling language for Mathematical Optimization
`GLPK`	interface for GLPK solver
`Gurobi`	interface for Gurobi solver
`CPLEX`	interface for the CPLEX solver
`vOptGeneric`	Multiobjective linear optimization solver; generic problems
`vOptSpecific`	Multiobjective linear optimization solver; specific problems

Adding a package (from the external library)

Commands to invoque **once** before the first use of the given package:

Method 1:

```
(@v1.7) pkg> add PyPlot
(@v1.7) pkg> add JuMP,GLPK
```

Method 2:

```
julia> using Pkg
julia> Pkg.add("PyPlot")
julia> Pkg.add("JuMP")
julia> Pkg.add("GLPK")
```

Adding a package (from the external library)

Commands to invoque **once** before the first use of the given package:

Method 1:

```
(@v1.7) pkg> add PyPlot
(@v1.7) pkg> add JuMP,GLPK
```

Method 2:

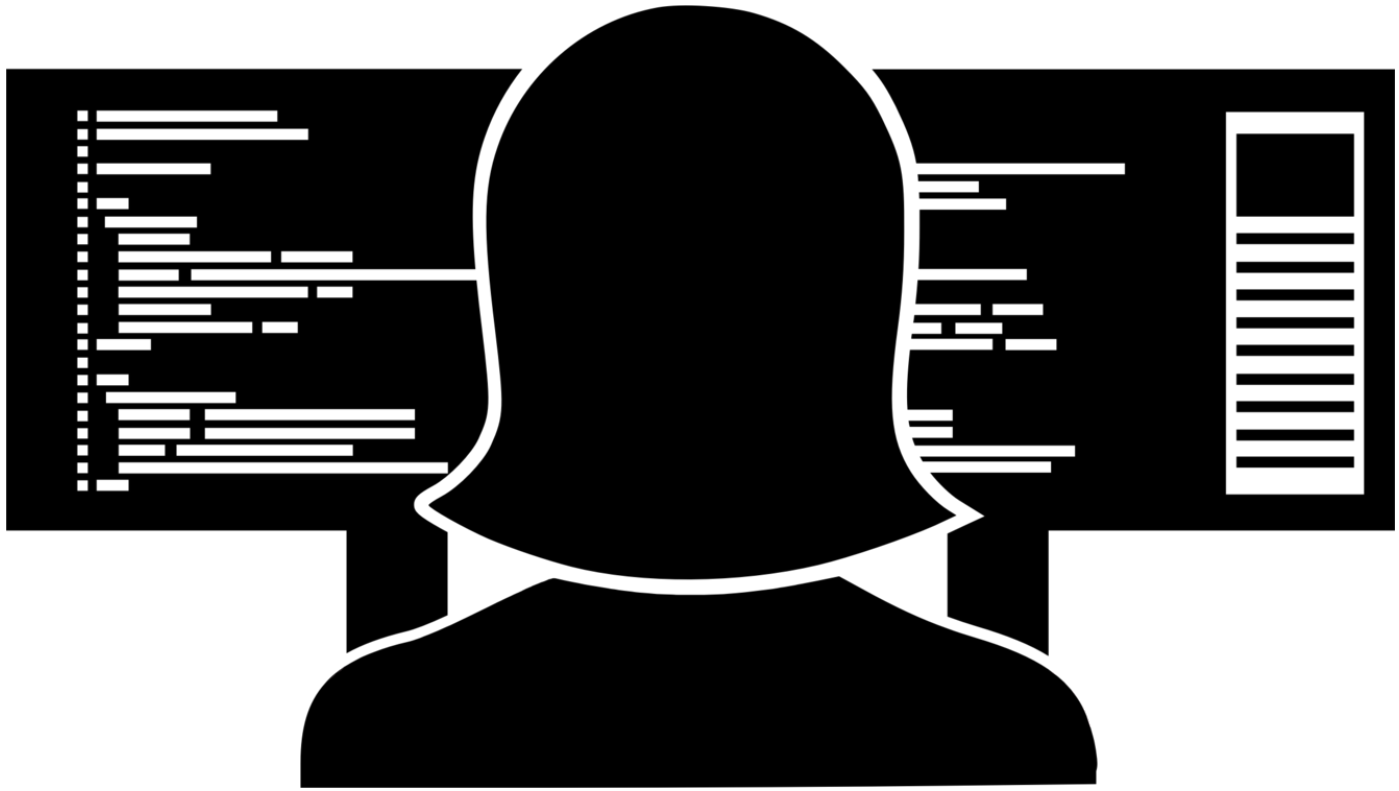
```
julia> using Pkg
julia> Pkg.add("PyPlot")
julia> Pkg.add("JuMP")
julia> Pkg.add("GLPK")
```

Using a package

Commands to invoque **each time** before the first use of the given package:

```
julia> using Pkg
julia> using JuMP,GLPK
julia> using Printf
julia> using LinearAlgebra
```

Let’s try



Running instructions into the REPL

1) Click on the app icon...

... the REPL is ready!

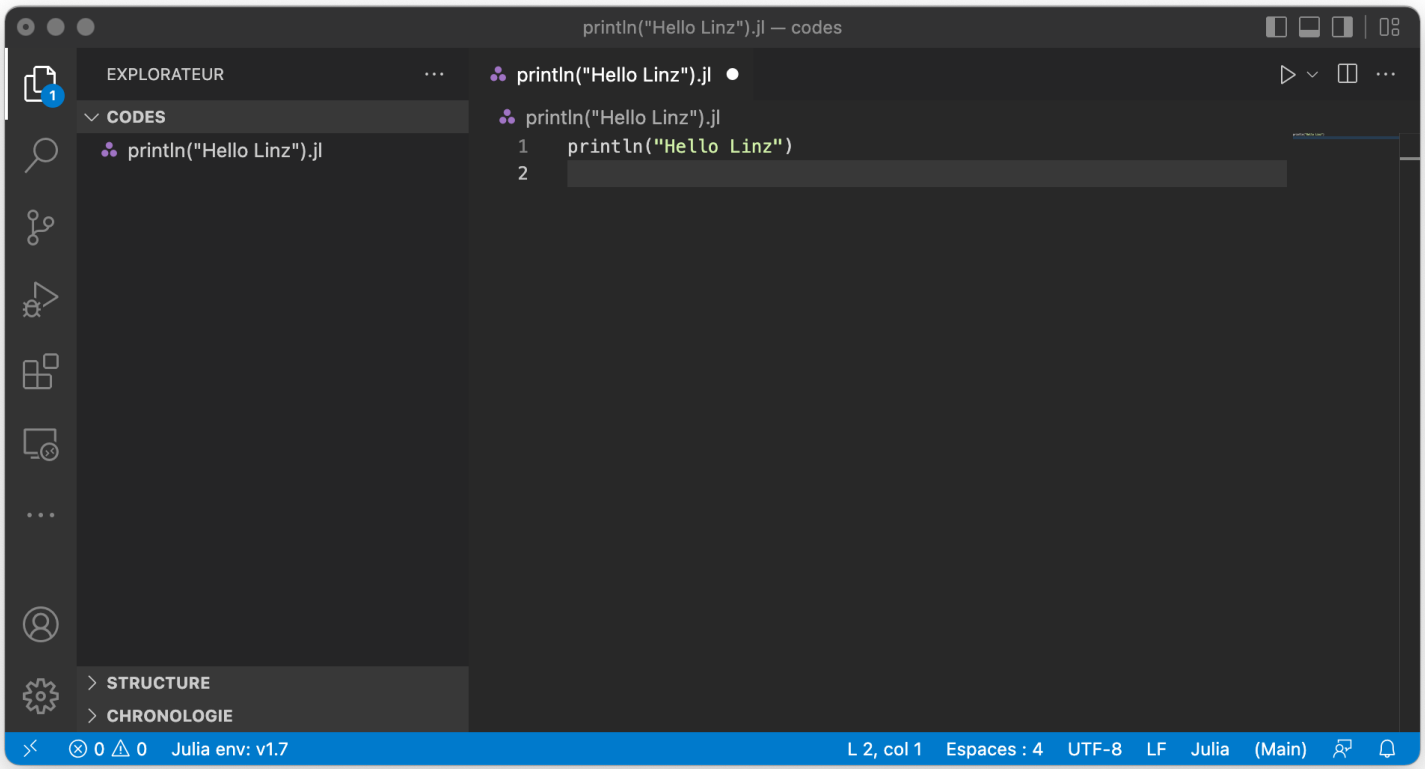
2) Type your instructions...

```
julia> println("Hello Linz")
```

... done!

Running a program into the REPL

Program prepared with a text editor (here VScode)...



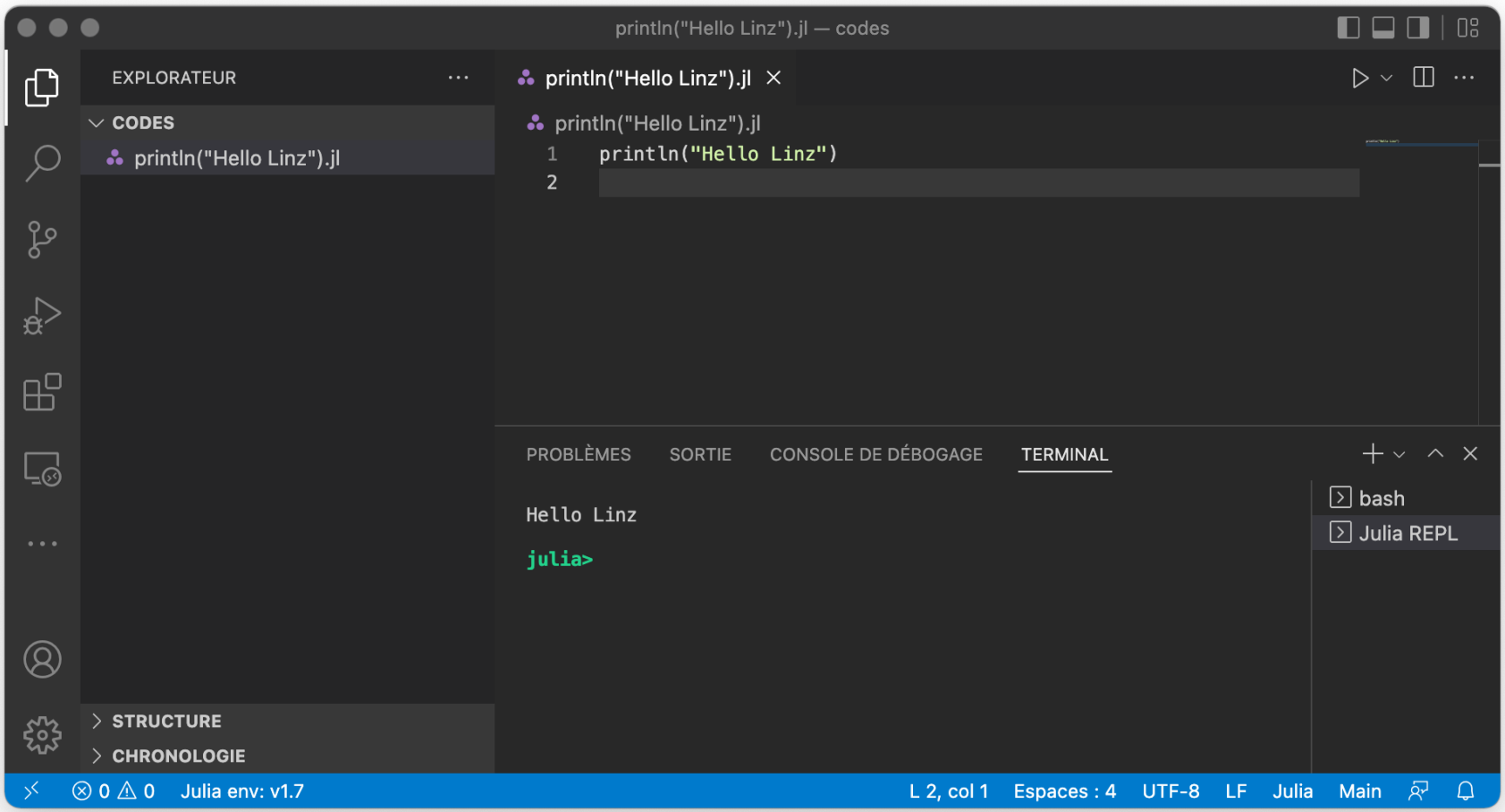
...to load and execute into the REPL :

```
julia> include("hello.jl")
```

Running a program into an IDE (VScode)

Program prepared with a text editor (here VScode)...

...to load and execute into the VScode terminal:



Running Julia code into a notebook (Jupyter)

Jupyter `https://jupyter.org/`

`IJulia` has to be installed
(`https://github.com/JuliaLang/IJulia.jl`)

Steps to follow:

```
(@v1.7) pkg> add IJulia
```

```
julia> using IJulia
julia> notebook()
```

answer `yes` to the questions

Done!

Running Julia code into a notebook (Jupyter)

Jupyter `https://jupyter.org/`

`IJulia` has to be installed
(`https://github.com/JuliaLang/IJulia.jl`)

Steps to follow:

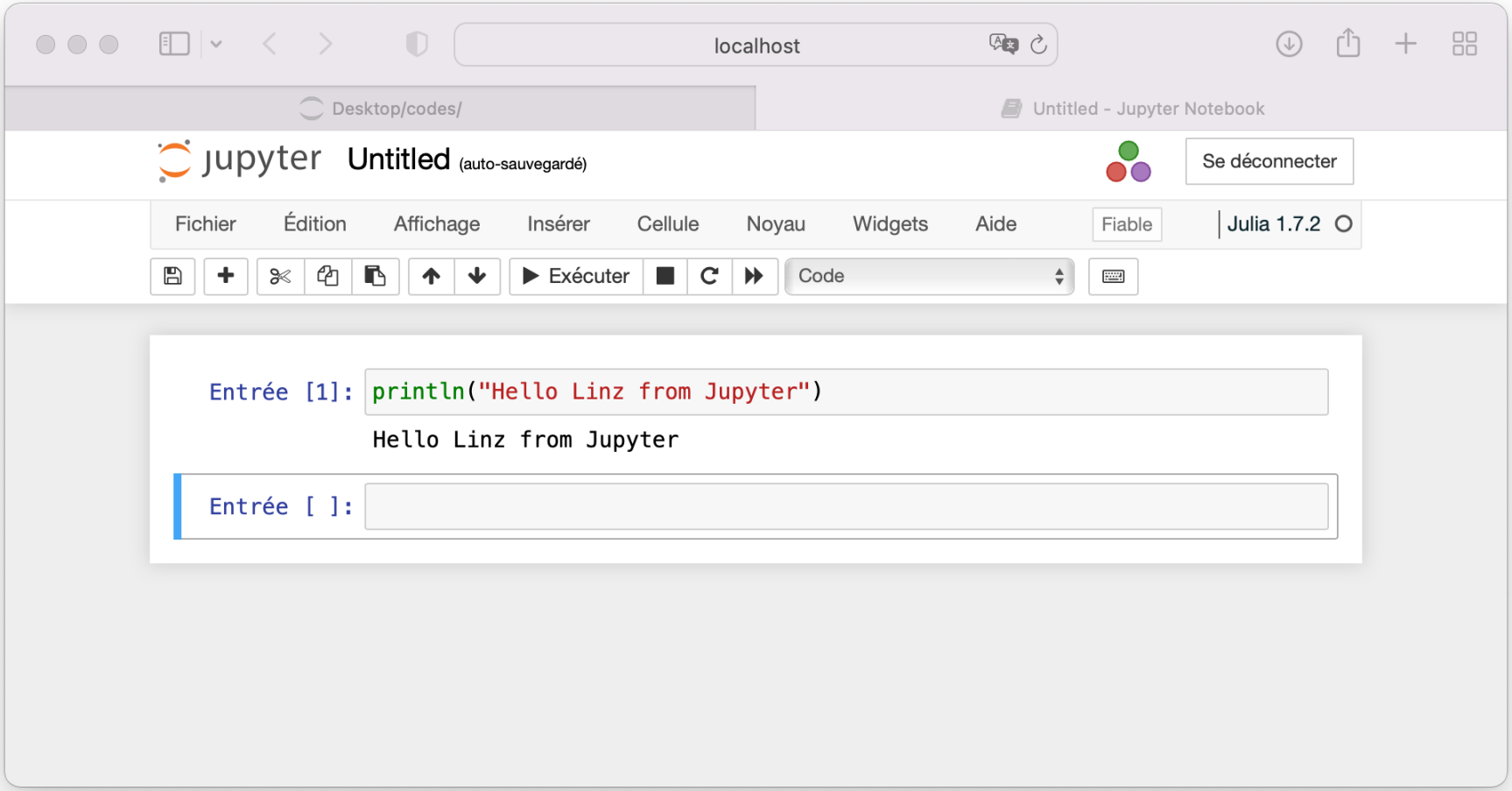
```
(@v1.7) pkg> add IJulia
```

```
julia> using IJulia
julia> notebook()
```

answer `yes` to the questions

Done!

Running Julia code into a notebook (Jupyter)



Running Julia code into a notebook (Pluto)

Pluto `https://github.com/fonsp/Pluto.jl`

Install: `https://www.youtube.com/watch?v=C4QhZcX34mI`

Steps to follow:

```
(@v1.7) pkg> add Pluto
```

```
julia> using Pluto
julia> Pluto.run()
```

Done!

Running Julia code into a notebook (Pluto)

Pluto `https://github.com/fonsp/Pluto.jl`

Install: `https://www.youtube.com/watch?v=C4QhZcX34mI`

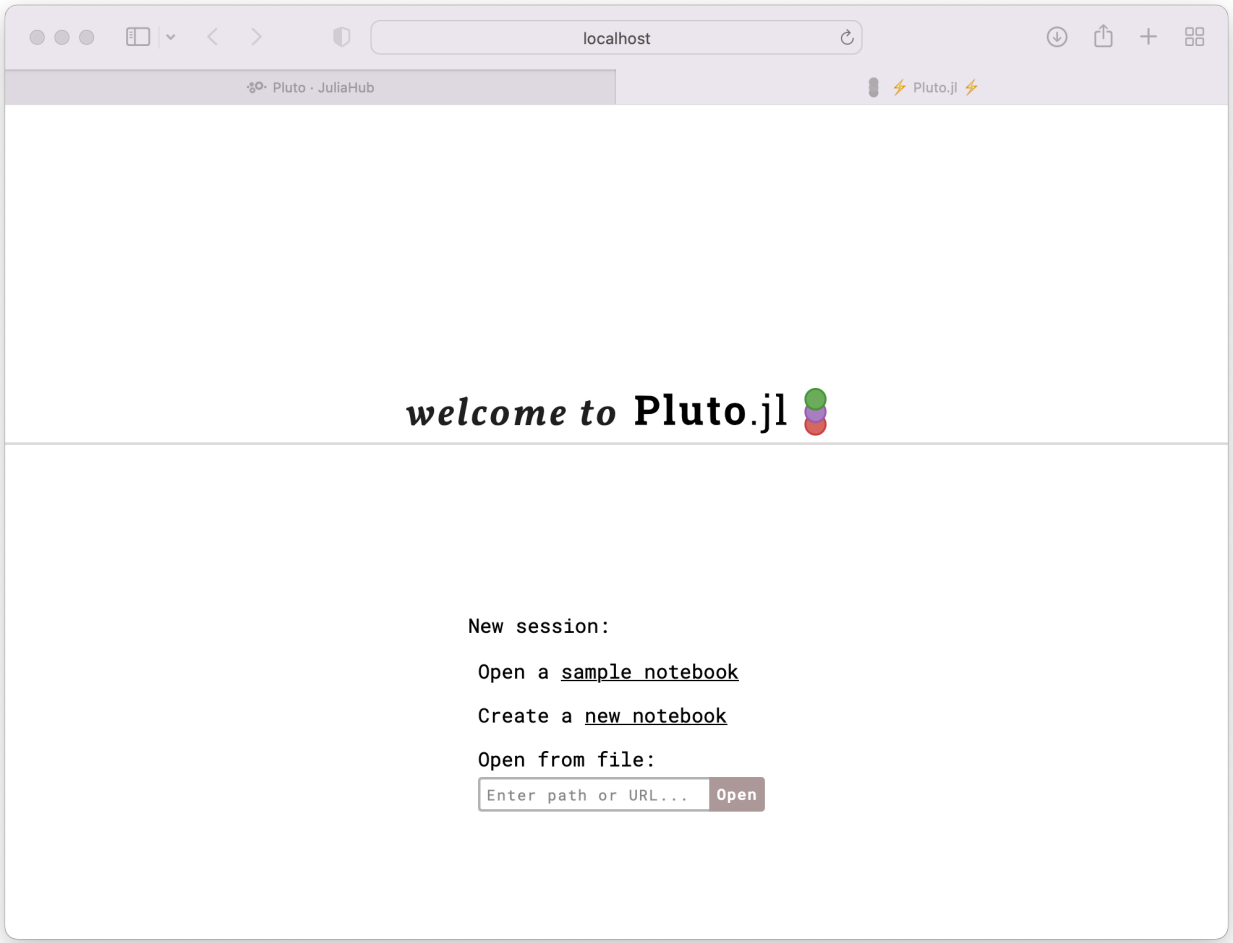
Steps to follow:

```
(@v1.7) pkg> add Pluto
```

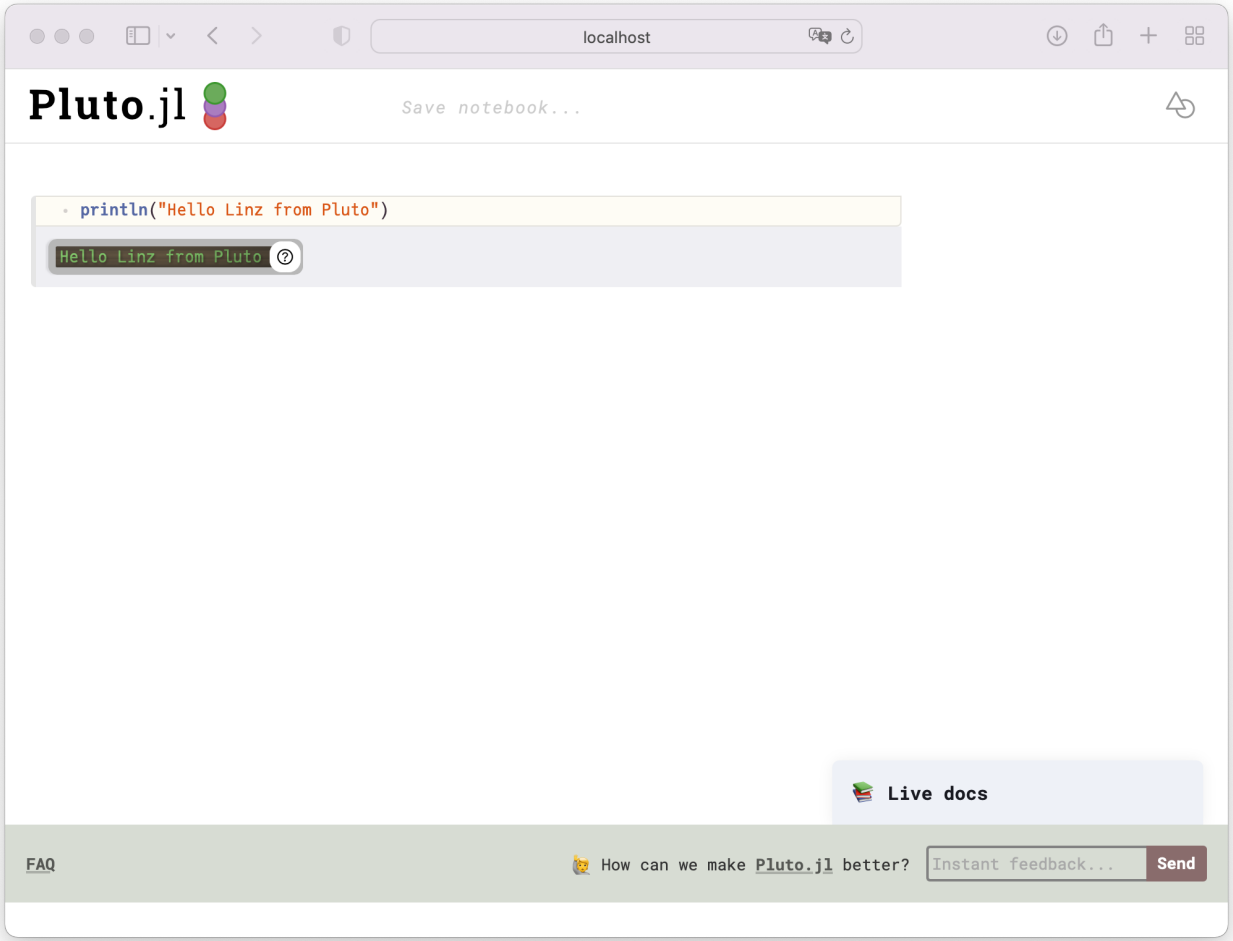
```
julia> using Pluto
julia> Pluto.run()
```

Done!

Running Julia code into a notebook (Pluto)



Running Julia code into a notebook (2)



Let’s go for installing the apps and do your first runs