An Introduction to julia for Scientific Computing

Francesco Martinuzzi

Center for Scalable Data Analytics and Artificial Intelligence
Remote Sensing Centre for Earth System Research
*Email: martinuzzi@informatik.uni-leipzig.de
Web: https://martinuzzifrancesco.github.io
Twitter: @MartinuzziFra

ellis

ScaDS.AI
DRESDEN LEIPZIG

UNIVERSITÄT
LEIPZIG

Who am I?

- PhD student at Leipzig University
- Avid contributor to the Julia open source community:
  - ReservoirComputing.jl, CellularAutomata.jl, WeightInitializers.jl
  - EarthDataLab.jl, YAXArrays.jl, Lux.jl, PredefinedDynamicalSystems.jl
- Ex Machine Learning Engineer (2020-2022) at Julia Computing (now JuliaHub)
- Google Summer of code contributor with the Julia language (summer 2020)

Resources 1: How

Tools:

1. **Visual Studio Code** with the Julia add-on: allows in-line compilation, spell checking, highlighting, methods check etcetc…

2. **Jupyter** Notebook: high level scripting and data visualization

3. **Pluto.jl**: A reactive Jupyter notebook written just for Julia. Changes in a cell propagate in the rest of the notebook

4. The **Julia REPL**

5. Any IDE of your choice: Emacs, Vim, SublimeText

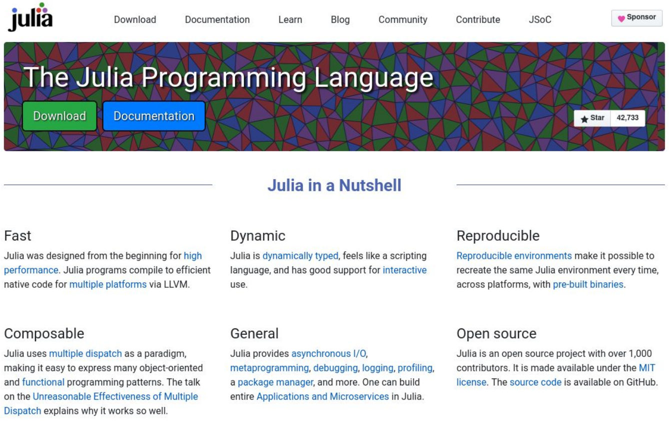
Install:

1. Either go to https://julialang.org/downloads/ and install from binaries

2. Or download the ready-to-use binaries for your OS (sudo apt install julia, Windows apps etc…)

Resources 2: Where

- Official website: https://julialang.org/
- Official documentation: https://docs.julialang.org/en/v1/
- Forum:
  - Discourse: https://discourse.julialang.org/
  - Reddit: https://www.reddit.com/r/Julia/
- Chat:
  - Slack: https://julialang.org/slack/
  - Zulip: https://julialang.zulipchat.com/register/
  - Discord: https://discord.gg/mm2kYjB
- Source Code: https://github.com/JuliaLang/julia



The Julia Programming Language

Julia in a Nutshell

Fast
Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

Dynamic
Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.

Reproducible
Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable
Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General
Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

Open source
Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.

Resources 3: What

https://martinuzzifrancesco.github.io/code/20230710introjl.zip

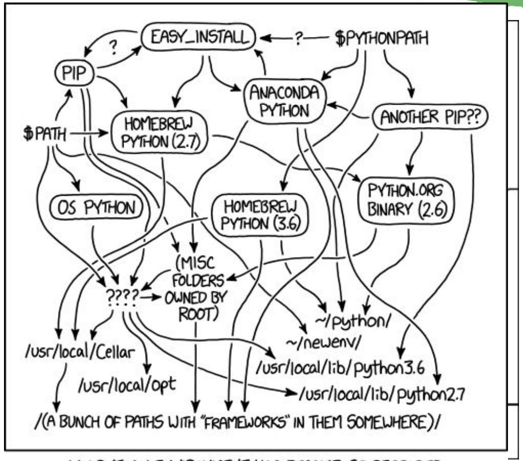
Coding in Science

Interactivity

Performance

Readability

Shareability



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Scientific Programming Languages

High Level

Python

- Slow
- Relies too much on external packages for scientific computing

Matlab

- Closed source
- Also very slow

Low Level

C++

- Dense grammar
- Steep learning curve

Fortran

- Outdated I/O interfacing
- Unfortunate long history of new features

> *We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.*
>
> Donald E. Knuth

- Some languages excel in the 97%
- Others focus on the 3%
- We want to be greedy, we want the 100%

Why Julia

1. Solves **the two languages problem**: it is both expressive (high level) and performant/extensible (low level) thanks to the Just in Time (JIT) compilation

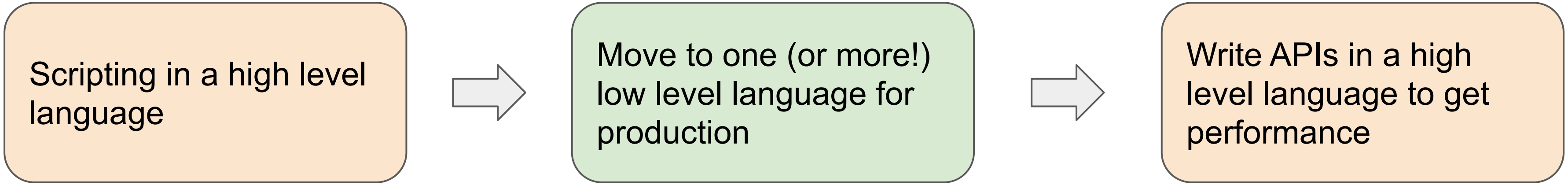
2. Leverages **multiple dispatch**: it just gives you the banana, not the gorilla holding the banana and the entire jungle

3. Built-in **package manager**, **virtual environments** and clean installation: no more pip/conda/miniconda shenanigans

4. Native **GPU** and **parallel** computation support

5. Rich and growing **scientific package ecosystem**

The two languages problem

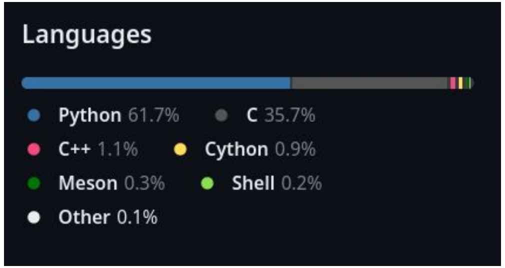


Scripting in a high level language

Move to one (or more!) low level language for production

Write APIs in a high level language to get performance

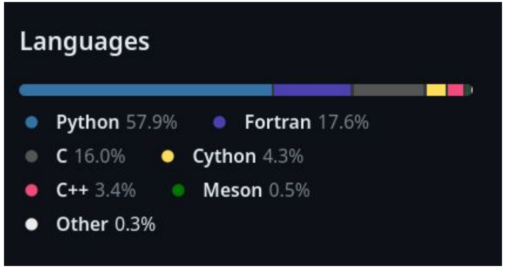
Numpy



Languages

- ● Python 61.7%
- ● C 35.7%
- ● C++ 1.1%
- ● Cython 0.9%
- ● Meson 0.3%
- ● Shell 0.2%
- ● Other 0.1%

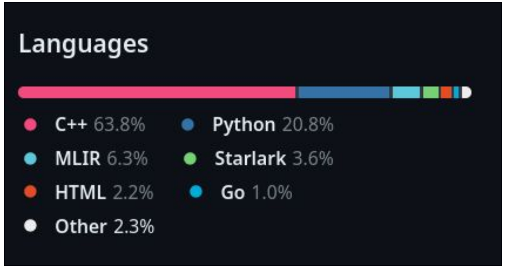
Scipy



Languages

- ● Python 57.9%
- ● Fortran 17.6%
- ● C 16.0%
- ● Cython 4.3%
- ● C++ 3.4%
- ● Meson 0.5%
- ● Other 0.3%

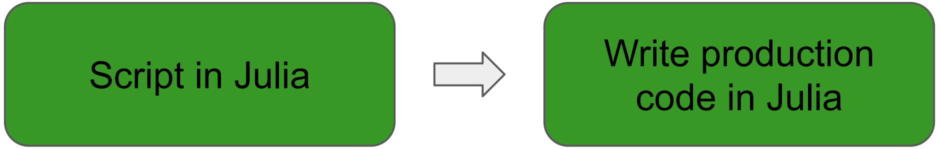
Tensorflow



Languages

- ● C++ 63.8%
- ● Python 20.8%
- ● MLIR 6.3%
- ● Starlark 3.6%
- ● HTML 2.2%
- ● Go 1.0%
- ● Other 2.3%

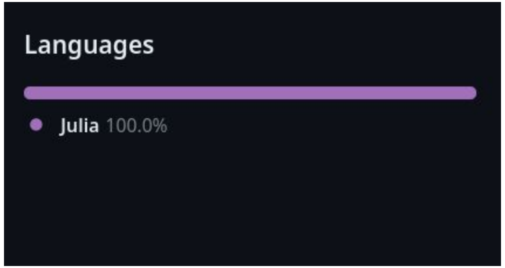
The two languages problem: solution



Script in Julia

Write production code in Julia

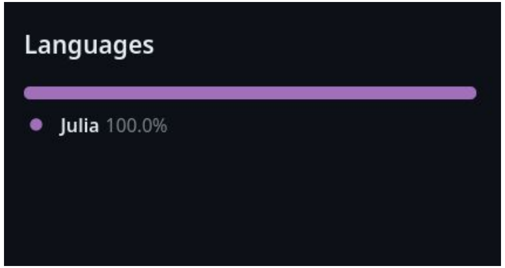
DataFrames.jl



Languages

● Julia 100.0%

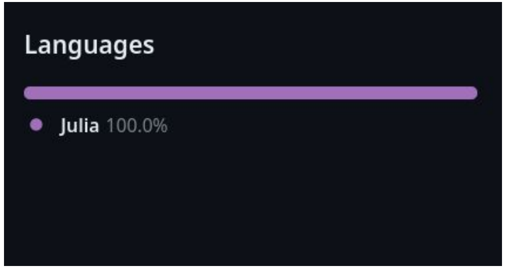
DifferentialEquations.jl



Languages

● Julia 100.0%

Flux.jl

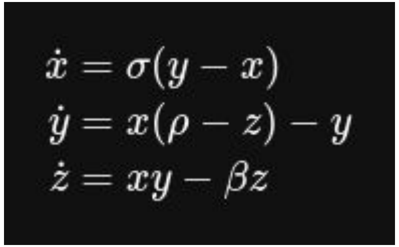


Languages

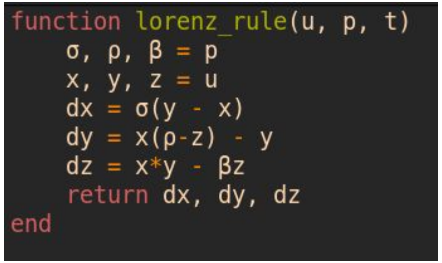
● Julia 100.0%

The two languages problem: Expressive

1. **Unicode** support

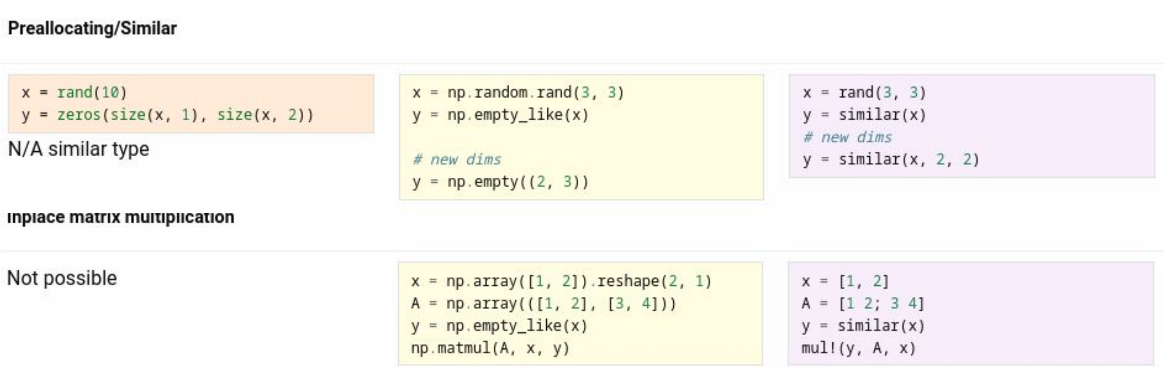


$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = x(\rho - z) - y$$
$$\dot{z} = xy - \beta z$$



```
function lorenz_rule(u, p, t)
    σ, ρ, β = p
    x, y, z = u
    dx = σ(y - x)
    dy = x(ρ-z) - y
    dz = x*y - βz
    return dx, dy, dz
end
```

2. Threats scientific computing as a **first class citizen**



Preallocating/Similar		

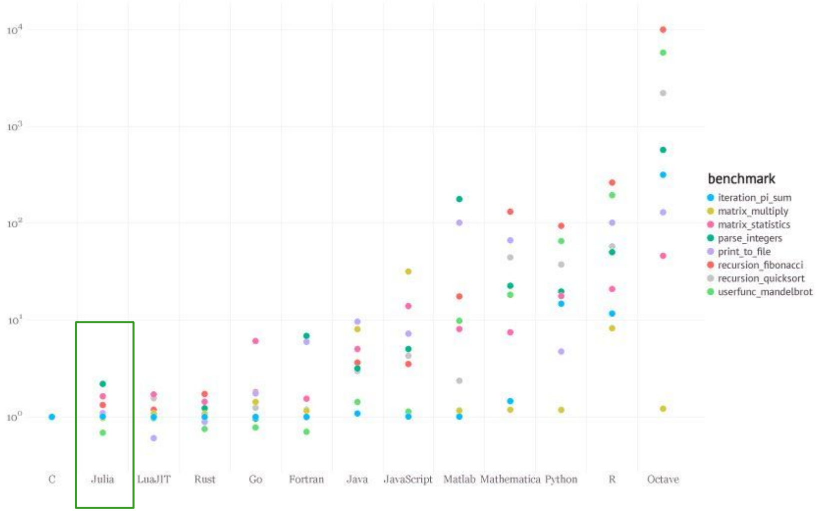
`x = rand(10)`
`y = zeros(size(x, 1), size(x, 2))`
N/A similar type
	`x = np.random.rand(3, 3)`
	`y = np.empty_like(x)`
	`# new dims`
	`y = np.empty((2, 3))`
		`x = rand(3, 3)`
		`y = similar(x)`
		`# new dims`
		`y = similar(x, 2, 2)`

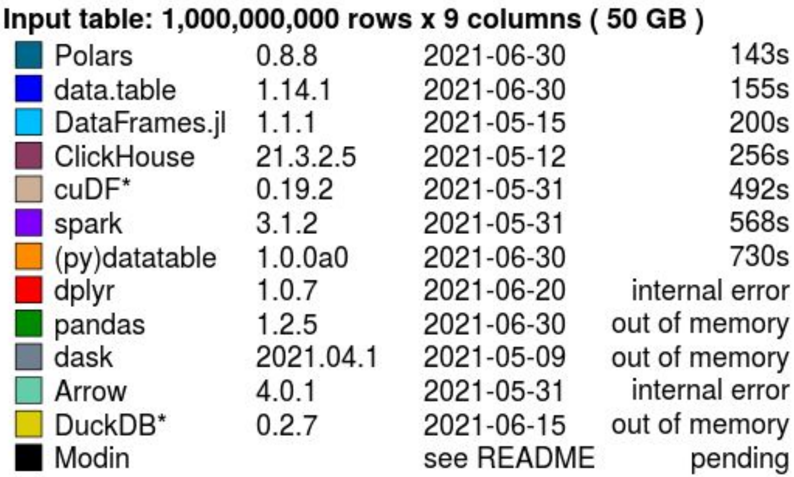
Inplace matrix multiplication		

Not possible	`x = np.array([1, 2]).reshape(2, 1)`
	`A = np.array(([1, 2], [3, 4]))`
	`y = np.empty_like(x)`
	`np.matmul(A, x, y)`
		`x = [1, 2]`
		`A = [1 2; 3 4]`
		`y = similar(x)`
		`mul!(y, A, x)`

https://cheatsheets.quantecon.org/

The two languages problem: Performant





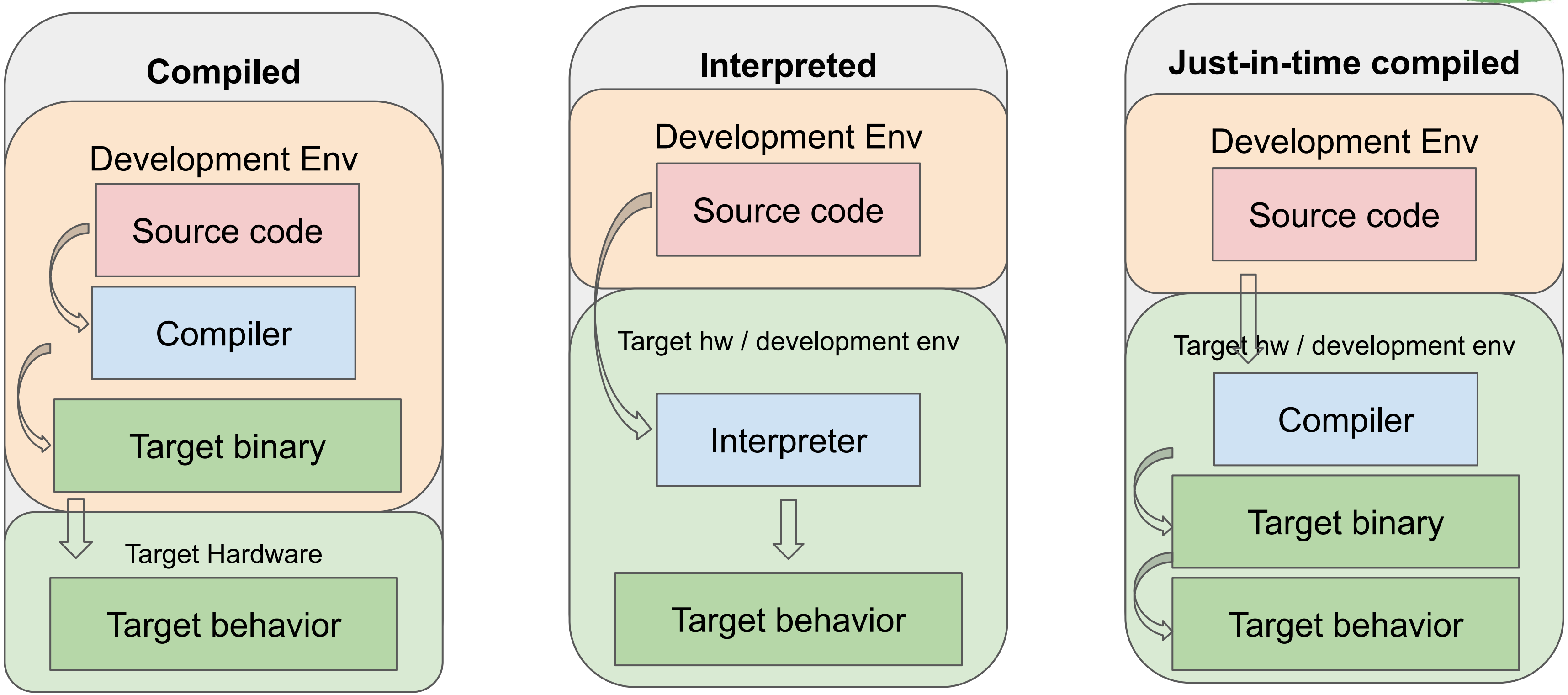
Input table: 1,000,000,000 rows x 9 columns (50 GB)

🟦	Polars	0.8.8	2021-06-30	143s
🟦	data.table	1.14.1	2021-06-30	155s
🟦	DataFrames.jl	1.1.1	2021-05-15	200s
🟪	ClickHouse	21.3.2.5	2021-05-12	256s
🟫	cuDF*	0.19.2	2021-05-31	492s
🟪	spark	3.1.2	2021-05-31	568s
🟧	(py)datatable	1.0.0a0	2021-06-30	730s
🟥	dplyr	1.0.7	2021-06-20	internal error
🟩	pandas	1.2.5	2021-06-30	out of memory
⬛	dask	2021.04.1	2021-05-09	out of memory
🟩	Arrow	4.0.1	2021-05-31	internal error
🟨	DuckDB*	0.2.7	2021-06-15	out of memory
⬛	Modin		see README	pending

VS Code Example from

https://nbviewer.org/github/rdeits/DetroitTechWatch2020.jl/blob/master/Intro%20to%20Julia.ipynb

The two languages problem: JIT



https://everyday.codes/python/why-python-written-in-python-is-faster-than-regular-python/

The Expression Problem

> *The expression problem is a new name for an old problem. The goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code, and while retaining static type safety (e.g., no casts).*
>
> Philip Wadler

The Expression Problem: Multiple Dispatch

Multiple dispatch is a feature that allows a function to **behave differently** based on the types of its arguments.

- No inheritance hierarchies and method overriding
- Easier to extend the behavior of existing functions without modifying their original code
- More concise and readable code
- It allows different modules or libraries to define their own methods for functions, enabling seamless integration without conflicts

REPL Example

- addition (Dates)
- Measurements
- Unitful

The Support System: Pkg.jl

- Allows you to create a virtual environment by default
  - Helps reproducibility
  - Helps code sharing
- Native creation of packages
- Clean installation of external packages

REPL Examples

Great Scientific Ecosystem

- Scientific Machine Learning: SciML
- Machine Learning: FluxML, LuxDL, MLJ
- Astrophysics: JuliaAstro, JuliaSpace
- Bio/Chemistry: JuliaBio, Molecular simulations
- Complex systems, nonlinear dynamics: JuliaDynamics
- Solid state: QuantumOptics, JuliaPhysics
- Economics: QuantEcon, JuliaQuant
- Geosciences/Climate: JuliaGeo, JuliaEarth, JuliaClimate, JuliaDataCubes

Outstanding Use Cases/Applications

- **Celeste**: ariational Bayesian inference for astronomical images (doi:10.1214/19-AOAS1258), 1.54 petaflops using 1.3 million threads on 9,300 Knights Landing (KNL) nodes on Cori at NERSC

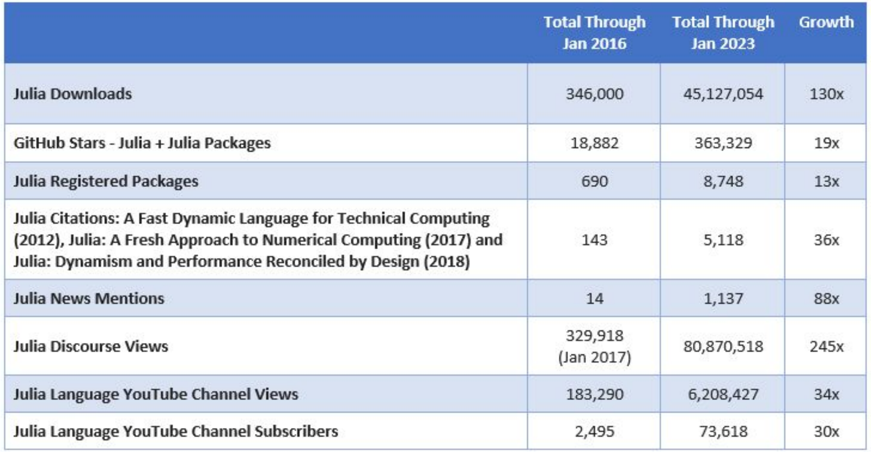
- **Clima**: Full earth climate simulation

- https://tshort.github.io/Lorenz-WebAssembly-Model.jl/

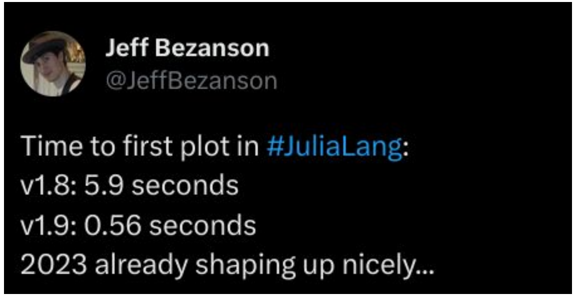
- https://alexander-barth.github.io/FluidSimDemo-WebAssembly/

Julia: Cons

- Very young
  - First public release in 2012
  - Stable v1.0 release in 2018
- Not good for very short automatization script (think shell scripting)
- Not (yet) suitable for non-computing web apps
- Compiler latency: startup time is sometimes still a pain



	Total Through Jan 2016	Total Through Jan 2023	Growth
Julia Downloads	346,000	45,127,054	130x
GitHub Stars - Julia + Julia Packages	18,882	363,329	19x
Julia Registered Packages	690	8,748	13x
Julia Citations: A Fast Dynamic Language for Technical Computing (2012), Julia: A Fresh Approach to Numerical Computing (2017) and Julia: Dynamism and Performance Reconciled by Design (2018)	143	5,118	36x
Julia News Mentions	14	1,137	88x
Julia Discourse Views	329,918 (Jan 2017)	80,870,518	245x
Julia Language YouTube Channel Views	183,290	6,208,427	34x
Julia Language YouTube Channel Subscribers	2,495	73,618	30x



Jeff Bezanson
@JeffBezanson

Time to first plot in #JuliaLang:
v1.8: 5.9 seconds
v1.9: 0.56 seconds
2023 already shaping up nicely...

Resources

- https://nbviewer.org/github/rdeits/DetroitTechWatch2020.jl/blob/master/Intro%20to%20Julia.ipynb
- https://www.youtube.com/watch?v=7y-ahkUsIrY
- https://github.com/Datseris/Zero2Hero-JuliaWorkshop#why-should-i-learn-julia
- https://ucidatascienceinitiative.github.io/IntroToJulia/
- https://github.com/carstenbauer/JuliaWorkshop19
- https://gdalle.github.io/JuliaComputationSolutions/hw1a_solutions.html
- https://gdalle.github.io/IntroJulia/sales_pitch.html
- https://scientificcoder.com/the-art-of-multiple-dispatch
- https://www.youtube.com/watch?v=kc9HwsxE1OY
- https://www.youtube.com/watch?v=2MBD10IqWp8
- https://indico.cern.ch/event/1074269/contributions/4539601/attachments/2317518/3945412/why-julia slides.pdf
- https://h2oai.github.io/db-benchmark/