

Advanced Julia Workshop

Carsten Bauer @ University of Cologne, October 2019

How good are you at programming? A CEFR-like approach to measure programming proficiency

Raphael 'kena' Poss

A1 level



B1 level

Programming skills self-assessment matrix

		A1 Basic User	A2 Basic User	B1 Intermediate User	B2 Intermediate User	C1 Proficient User	C2 Proficient User
Writing	Writing code	I can produce a correct implementation for a simple function, given a well-defined specification of desired behavior and interface, without help from others.	I can determine a suitable interface and produce a correct implementation, given a loose specification for a simple function, without help from others. I can break down a complex function specification in smaller functions.	I can estimate the space and time costs of my code during execution. I can empirically compare different implementations of the same function specification using well-defined metrics, including execution time and memory footprint. I express invariants in my code using preconditions, assertions and post-conditions. I use stubs to gain flexibility on implementation order.	I use typing and interfaces deliberately and productively to structure and plan ahead my coding activity. I can design and implement entire programs myself given well-defined specifications on external input and output. I systematically attempt to generalize functions to increase their reusability.	I can systematically recognize inconsistent or conflicting requirements in specifications. I can break down a complex program architecture in smaller components that can be implemented separately, including by other people. I can use existing (E)DSLs or metaprogramming patterns to increase my productivity.	I can reliably recognize when under-specification is intentional or not. I can exploit under-specification to increase my productivity in non-trivial ways. I can devise new (E)DSLs or create new metaprogramming patterns to increase my productivity and that of other programmers.
	Refactoring	I can adapt my code when I receive small changes in its specification without rewriting it entirely, provided I know the change is incremental. I can change my own code given detailed instructions from a more experienced programmer.	I can determine myself whether a small change in specification is incremental or requires a large refactoring. I can change my own code given loose instructions from a more experienced programmer.	I can derive a refactoring strategy on my own code, given relatively small changes in specifications. I can change other people's code given precise instructions from a person already familiar with the code.	I can predict accurately the effort needed to adapt my own code base to a new specification. I can follow an existing refactoring strategy on someone else's code. I can take full responsibility for the integration of someone else's patch onto my own code.	I can reverse-engineer someone else's code base with help from the original specification, and predict accurately the effort needed to adapt it to a new specification.	I can reverse-engineer someone else's code base without original specification, and predict accurately the effort needed to adapt it to a new specification.

Workshop Schedule

	Monday	Tuesday	Wednesday	Thursday	Friday
9:30 - 11:30 (2h)	Types + Dispatch	Environments + Julia Gotchas	Parallel Computing		
	Short break				
11:45 - 12:45 (1h)	Custom types	Linear Algebra	Distributed	Holiday	Hackathon + MiniWorkshops
	Lunch break				
14:00 - 16:00 (2h)	Specialization + Generic Prog.	Machine Learning (Physics)	Tips&Tricks + Ecosystem		

juliacan

2018



Workshop materials

Terminal

```
git clone https://github.com/crstnbr/julia-workshop-2019
```

```
cd julia-workshop-2019
```

```
julia install_dependencies.jl
```

Workshop materials

Julia REPL

```
using Pkg  
Pkg.GitTools.clone("http://tiny.cc/julia", "workshop")  
cd("workshop")  
include("install_dependencies.jl") # might take a while
```

Workshop materials

Browser

Download <https://github.com/crstnbr/julia-workshop-2019/archive/master.zip>

Unzip the file.

Open the Jupyter notebook `install_dependencies.ipynb` and execute the cells.



What does science need from a programming language?

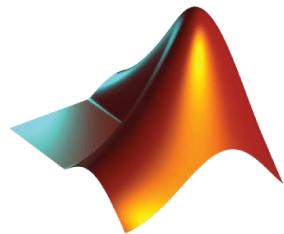
Easy to write and read !

Fast and scalable !

Interactive !



There's a plethora of programming languages



MATLAB



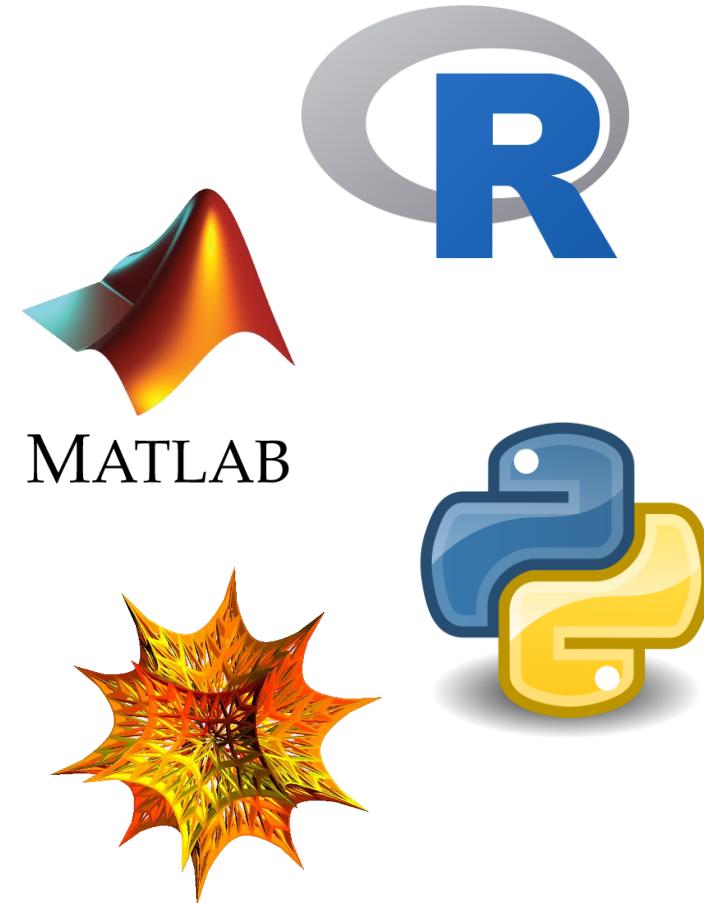
Fortran



There's a plethora of programming languages



Fortran



There's a plethora of programming languages

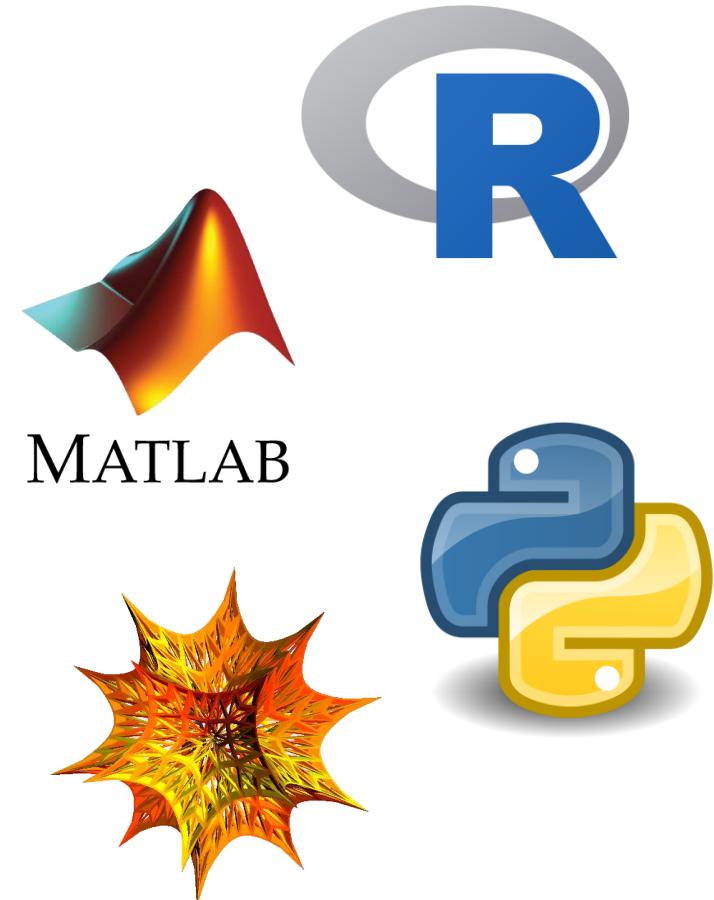


Fortran



Fast

Slow(ish)



There's a plethora of programming languages

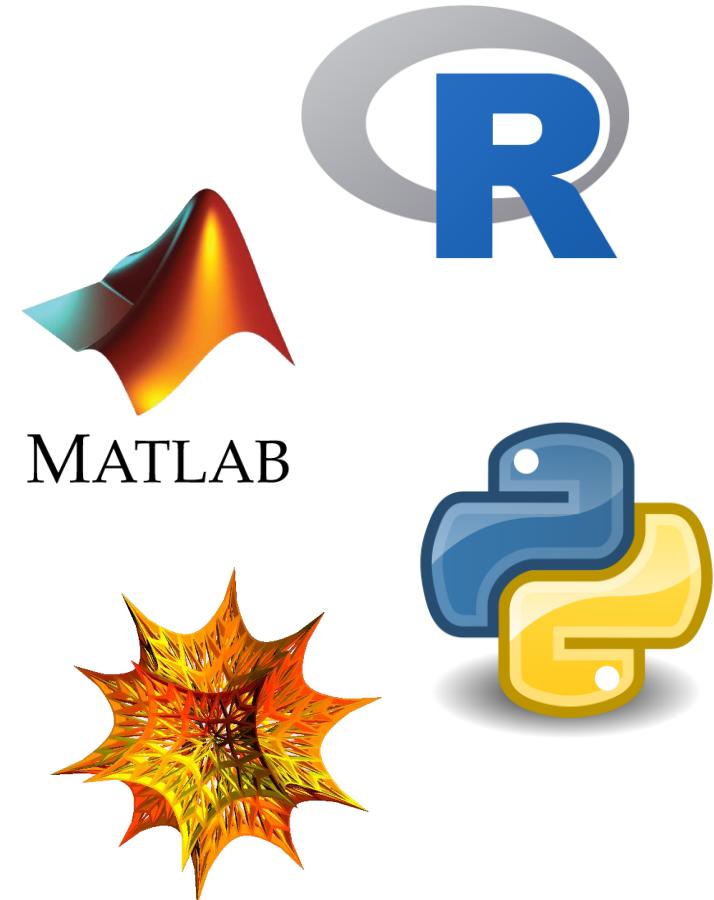


Fortran



Compiled

Interpreted



There's a plethora of programming languages

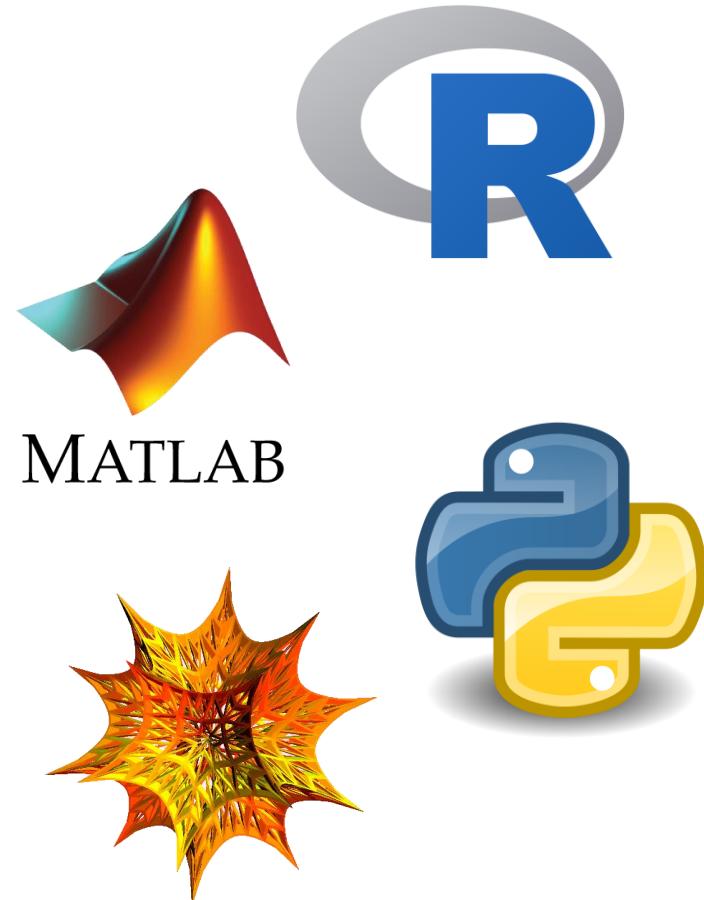


Fortran



Static

Dynamic



There's a plethora of programming languages

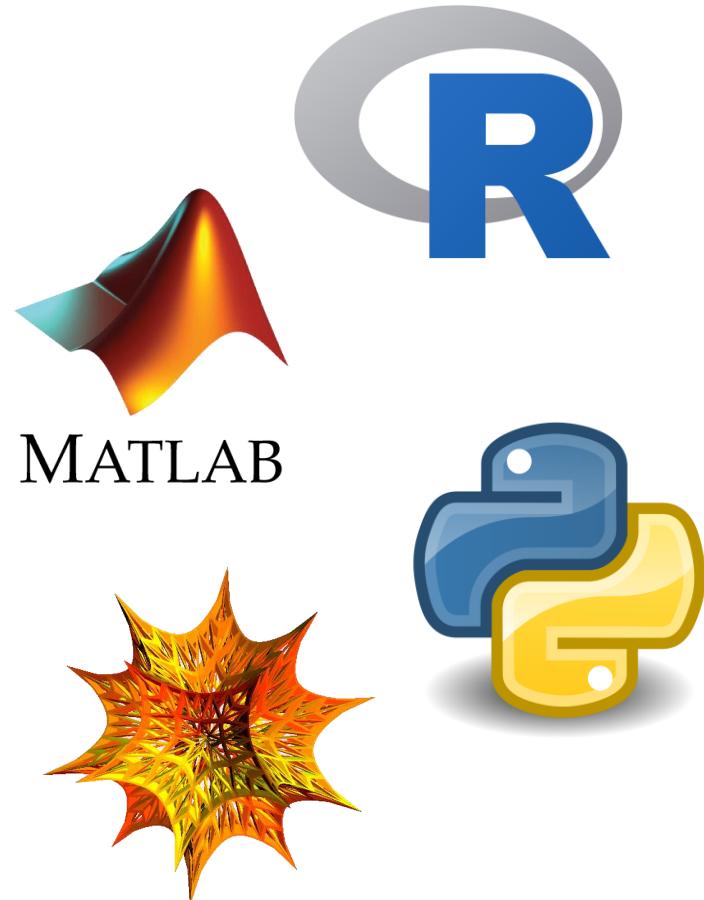


Fortran



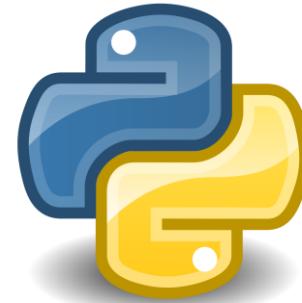
Speed

Convenience



The “two language problem”

a.k.a Ousterhout's dichotomy

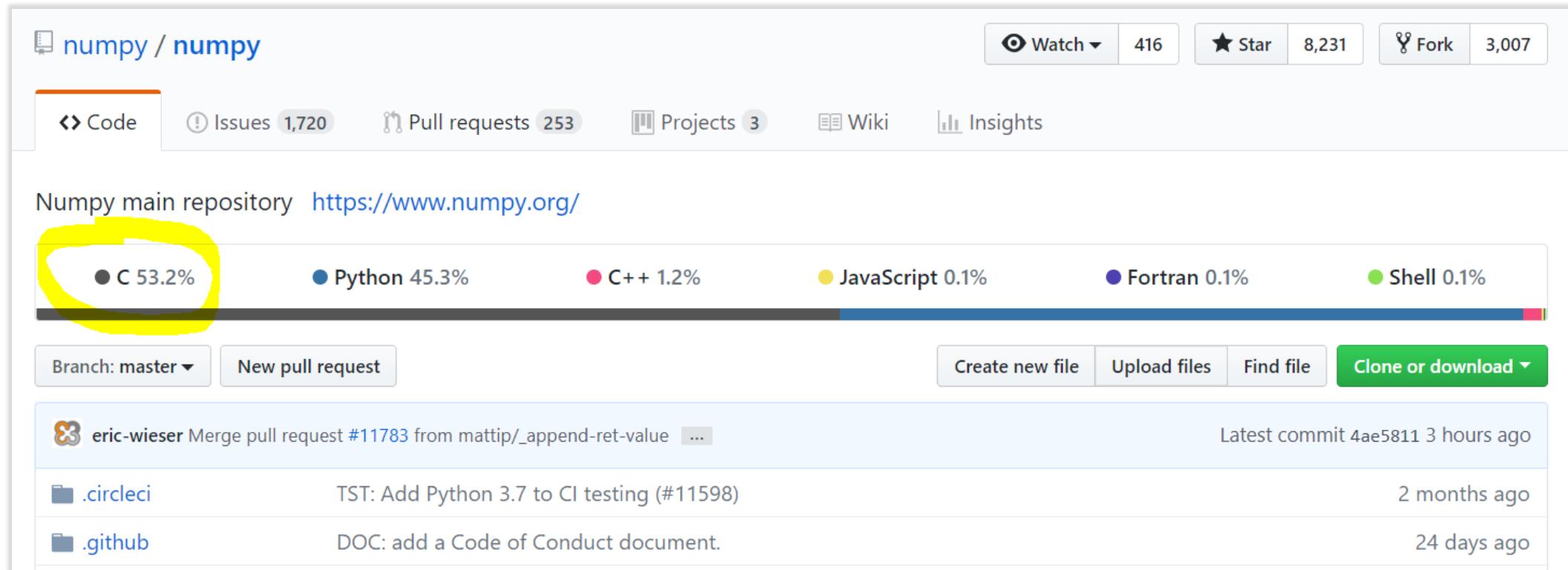


Prototype

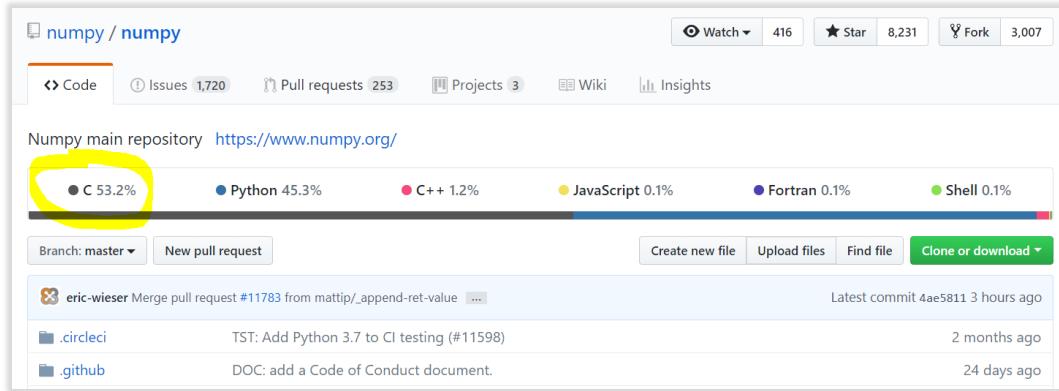
Production



The “two language problem”



The “two language problem”



Developer

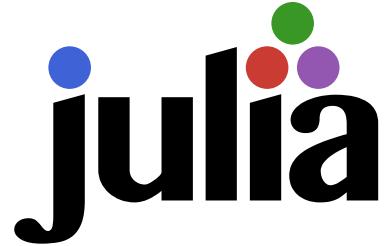
User



The “two language problem”

static
compiled
user types
standalone

dynamic
interpreted
standard types
glue



dynamic

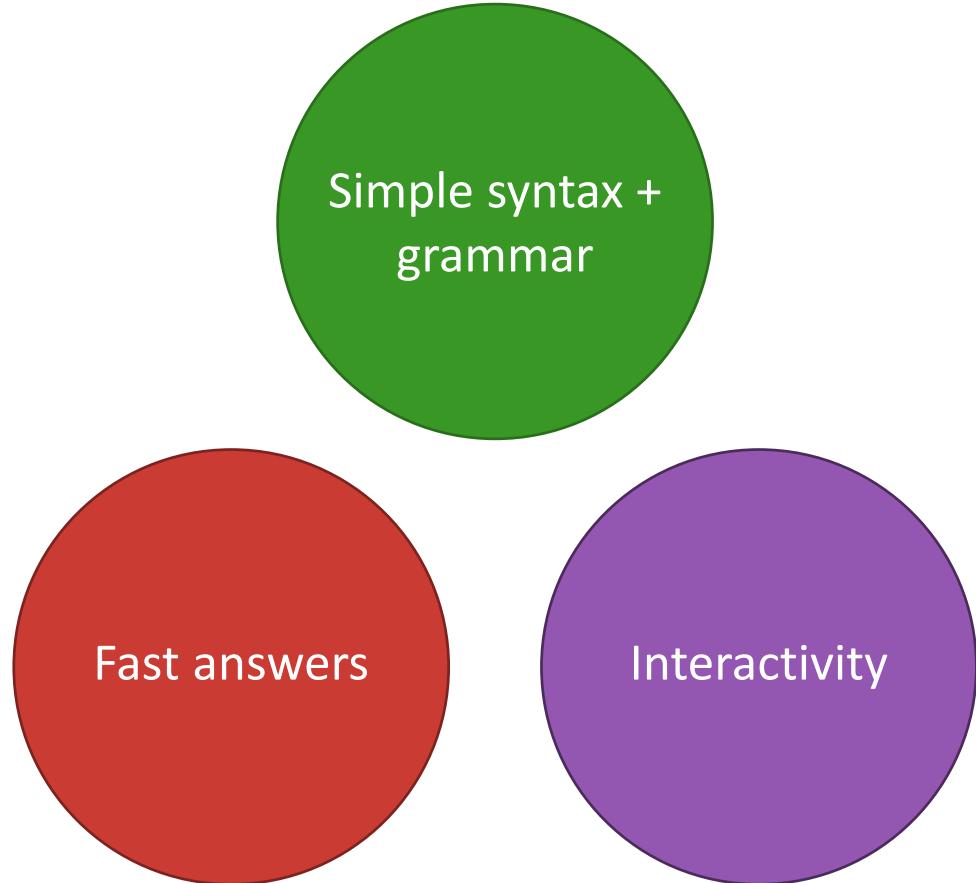
compiled

user types **and** standard types

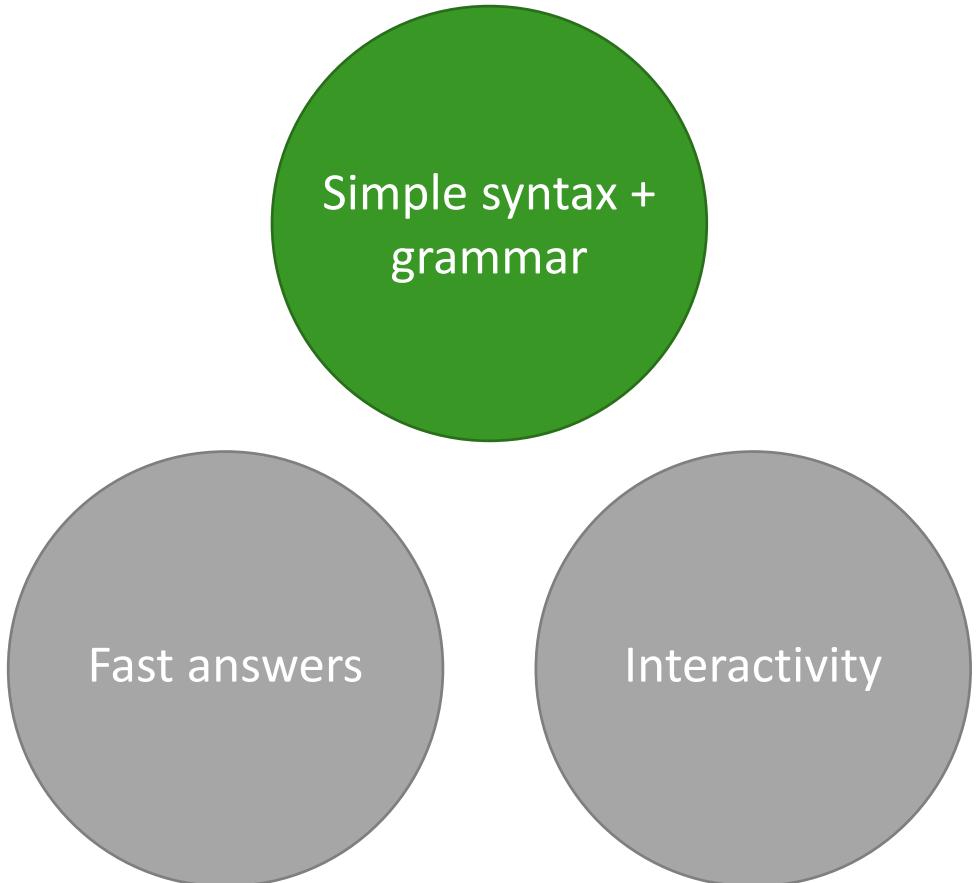
standalone **or** glue

“Feels like Python, runs like C”

The unification



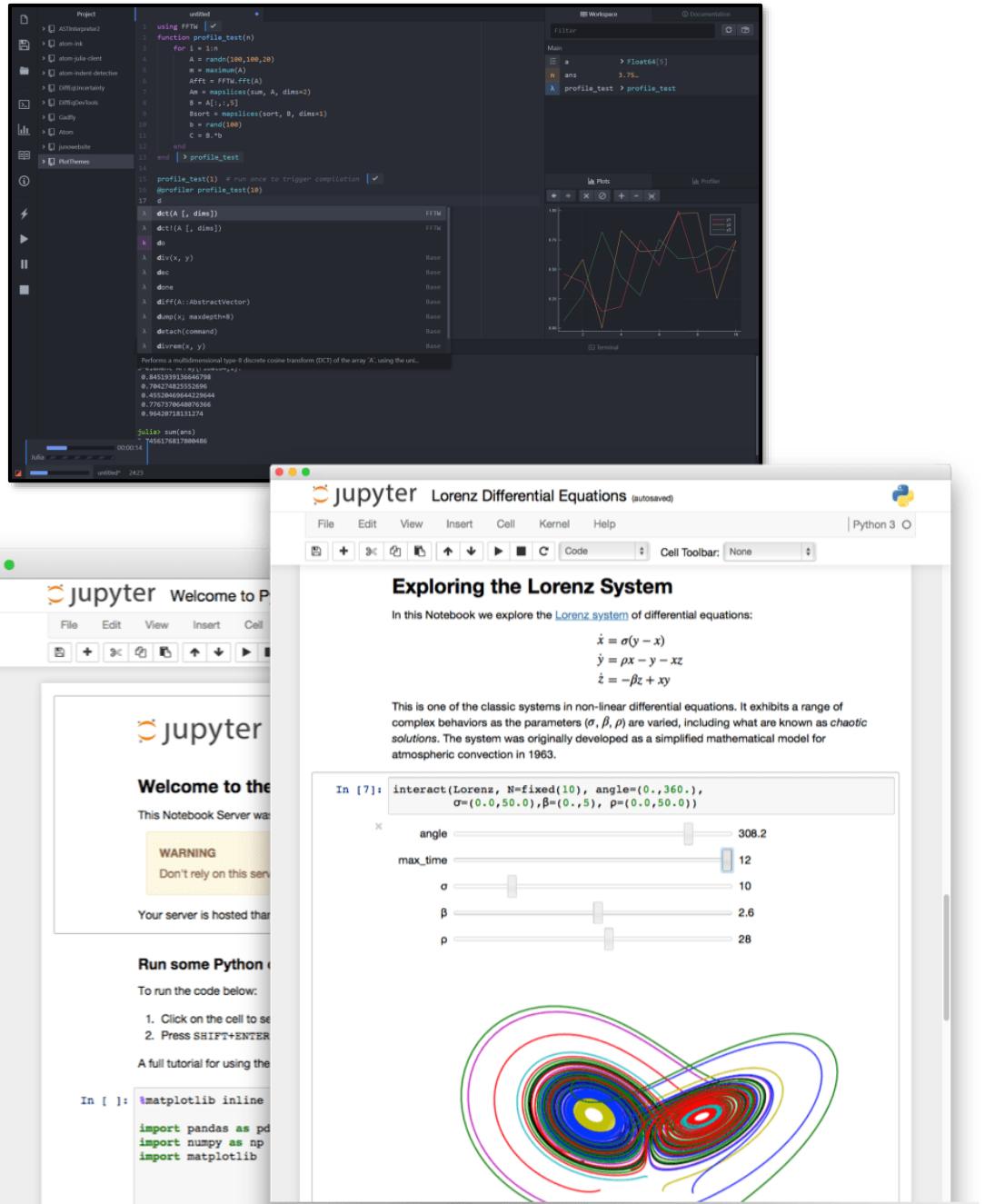
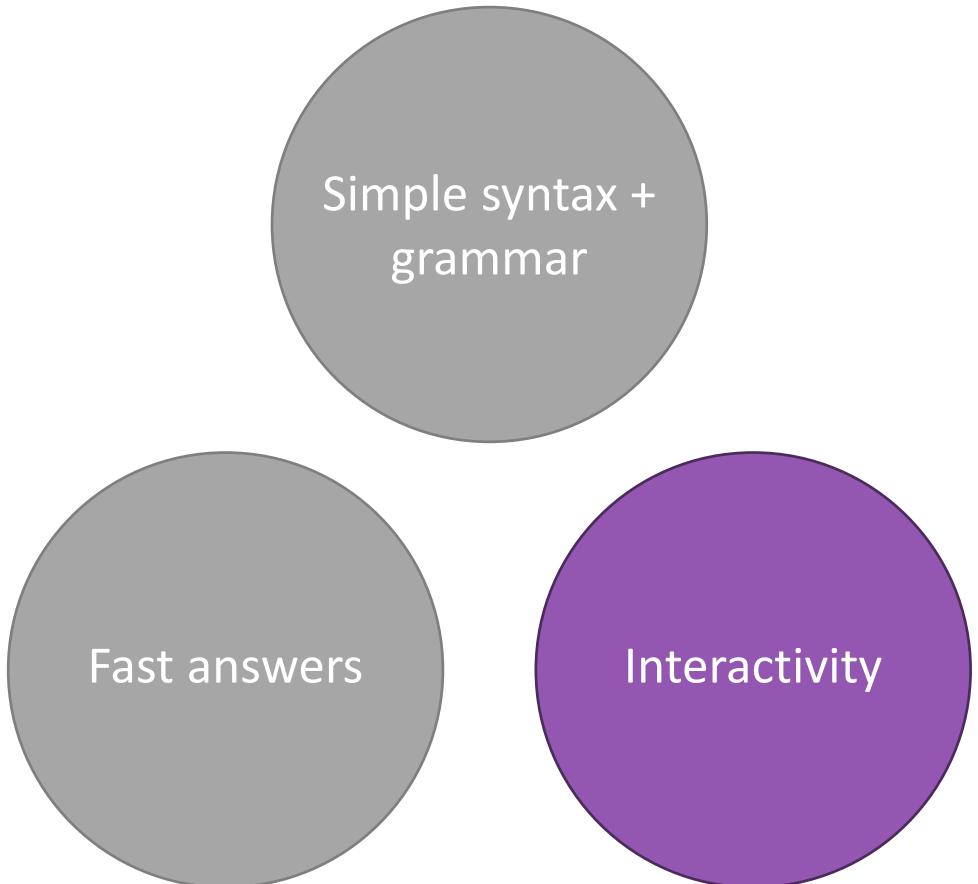
The unification



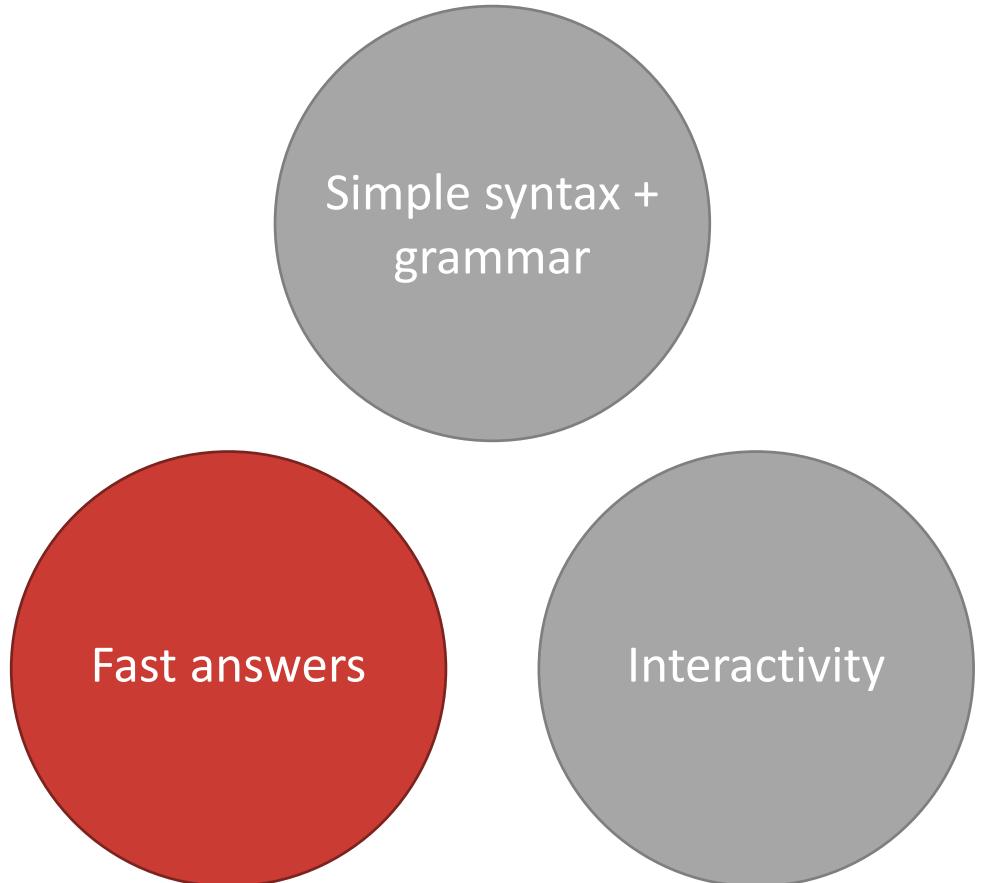
```
function babylonian(α; N = 10)
    @assert α > 0 "α must be > 0"
    t = (1+α)/2
    for i = 2:N
        t = (t + α/t)/2
    end
    t
end

babylonian(π) ≈ √π
```

The **julia** unification



The unification



```
julia> function sumup()
           x = 0
           for i in 1:100
               x += i
           end
           x
       end
sumup (generic function with 2 methods)
```

```
julia> @code_llvm debuginfo=:none sumup()

; Function Attrs: uwtable
define i64 @julia_sumup_12626() #0 {
top:
    ret i64 5050
}
```

Just returns the answer!
(The for loop was compiled away)

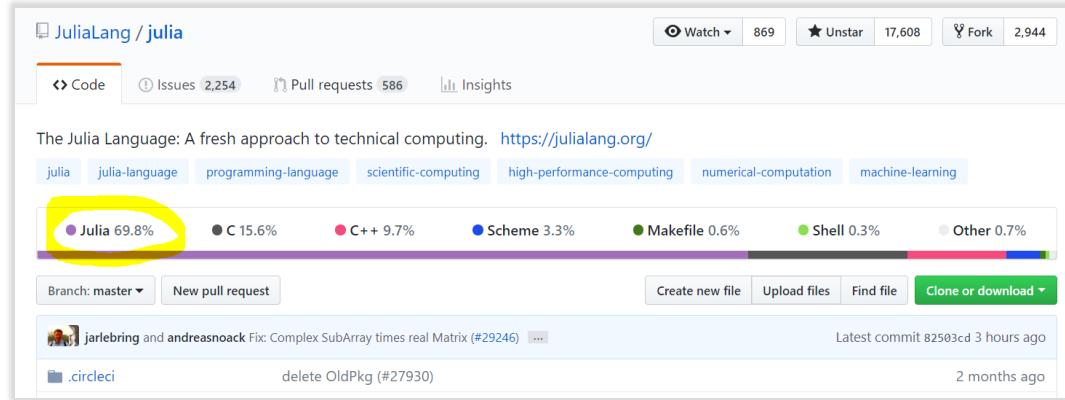
It is free and open source

Screenshot of the JuliaLang/julia GitHub repository page.

The page shows the following details:

- Repository:** JuliaLang/julia
- Watched:** 869
- Unstarred:** 17,608
- Forked:** 2,944
- Code:** 2,254 issues, 586 pull requests, Insights
- Tags:** julia, julia-language, programming-language, scientific-computing, high-performance-computing, numerical-computation, machine-learning
- Languages:** Julia 69.8%, C 15.6%, C++ 9.7%, Scheme 3.3%, Makefile 0.6%, Shell 0.3%, Other 0.7% (The "Julia" entry is highlighted with a yellow oval).
- Branch:** master
- Actions:** New pull request, Create new file, Upload files, Find file, Clone or download
- Recent Activity:**
 - jarlebring and andreasnoack Fix: Complex SubArray times real Matrix (#29246) ... Latest commit 82503cd 3 hours ago
 - .circleci delete OldPkg (#27930) 2 months ago

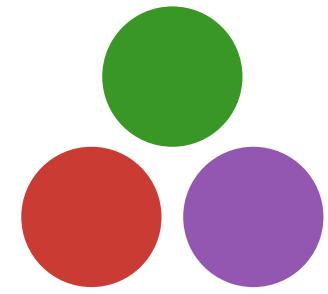
It is inviting



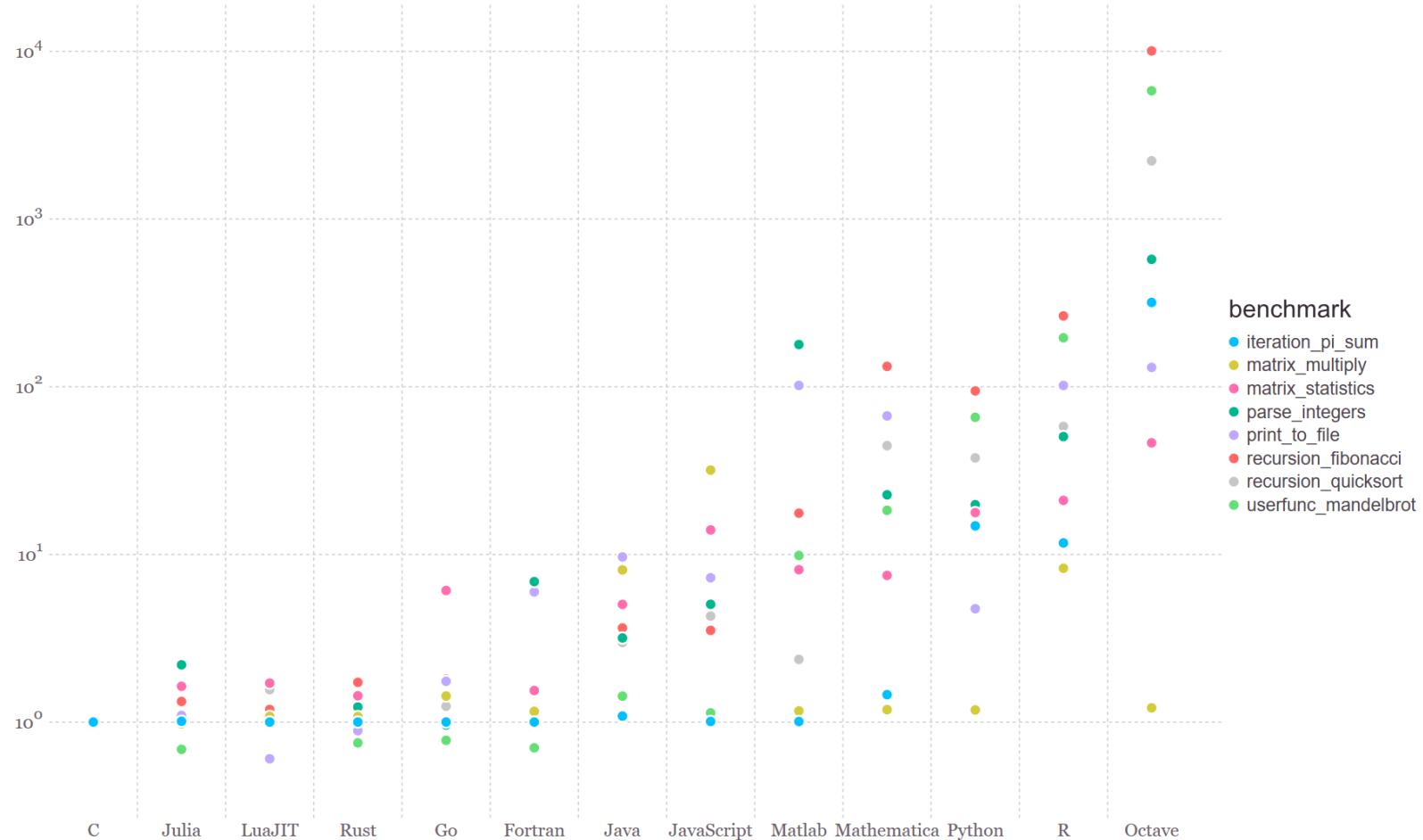
User



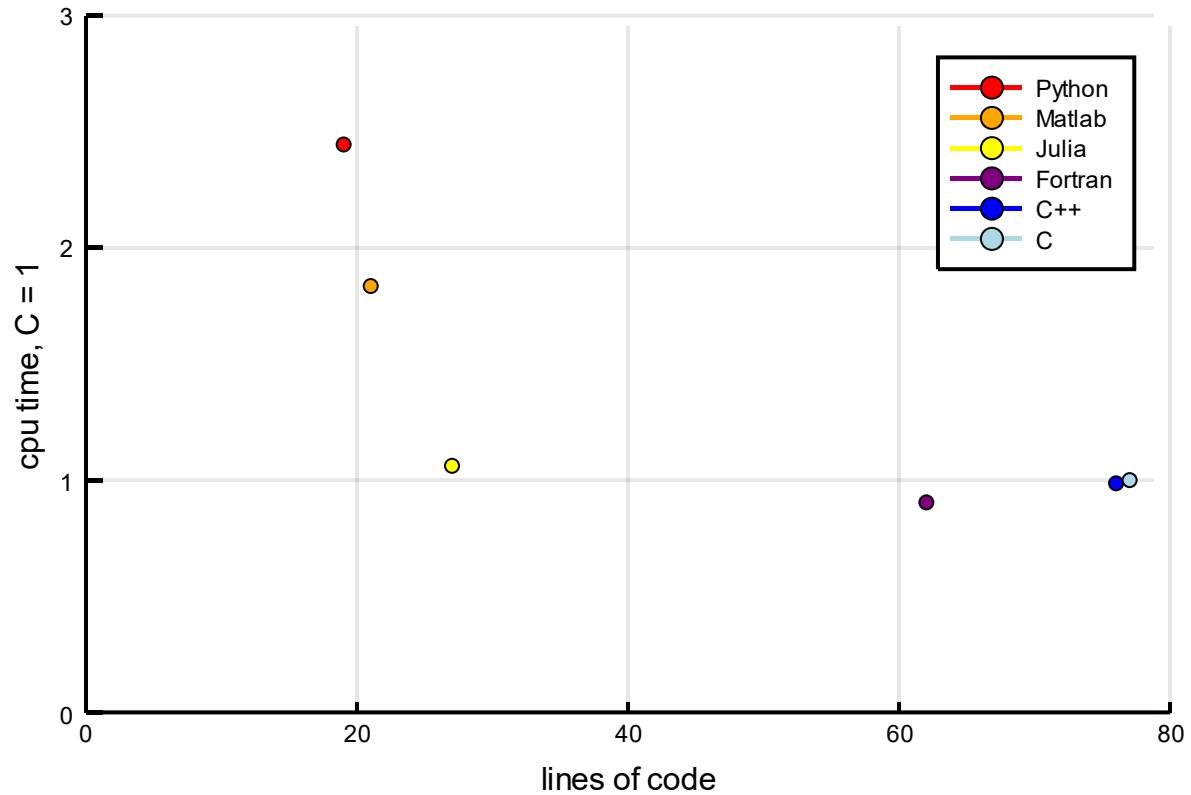
Developer



It is fast



It is expressive



How old is Julia?



Jeff, Stefan, Viral, and Alan start working on Julia



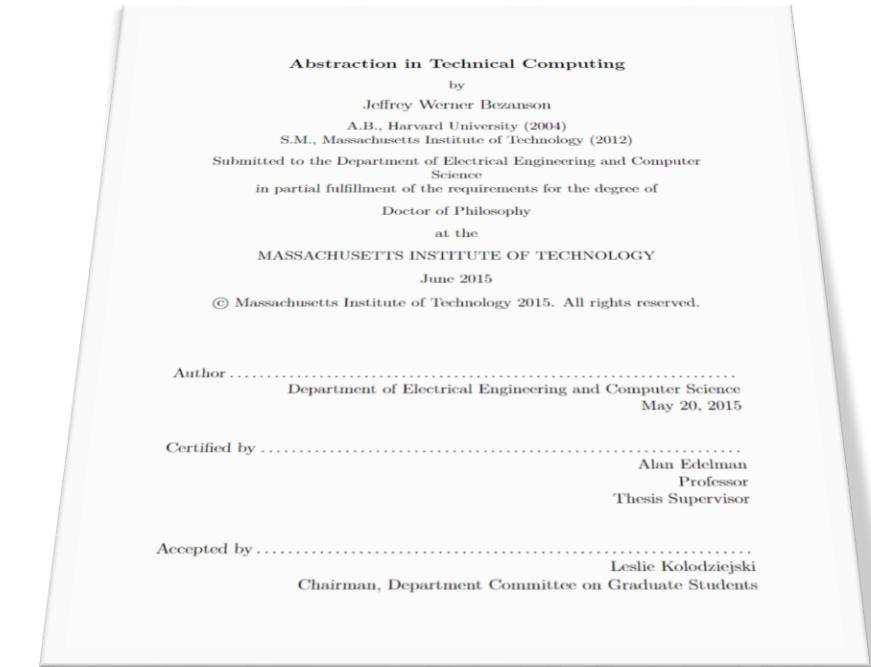
Blog post: [Why we created Julia](#)



Jeff's PhD & JuliaComputing

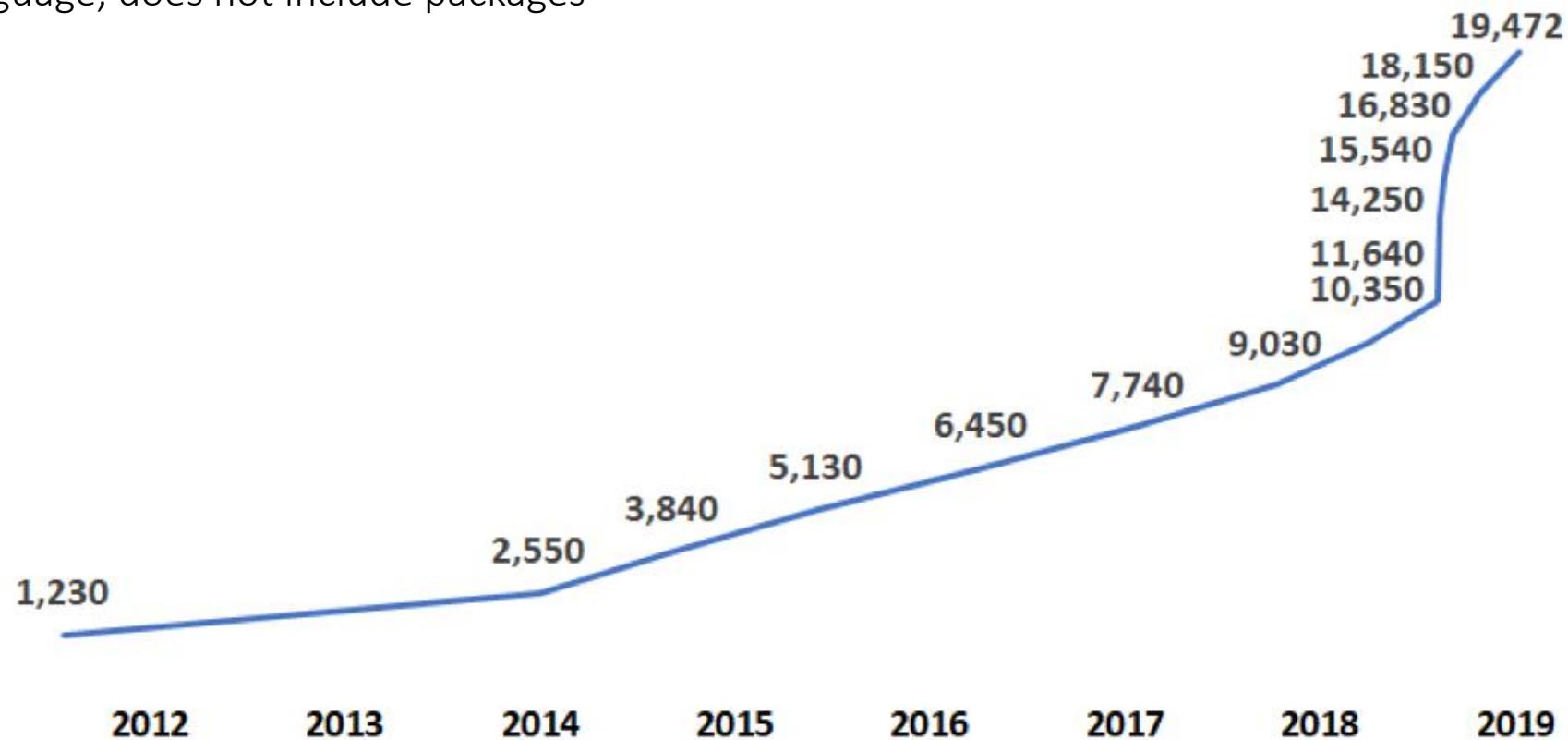


Julia 1.0



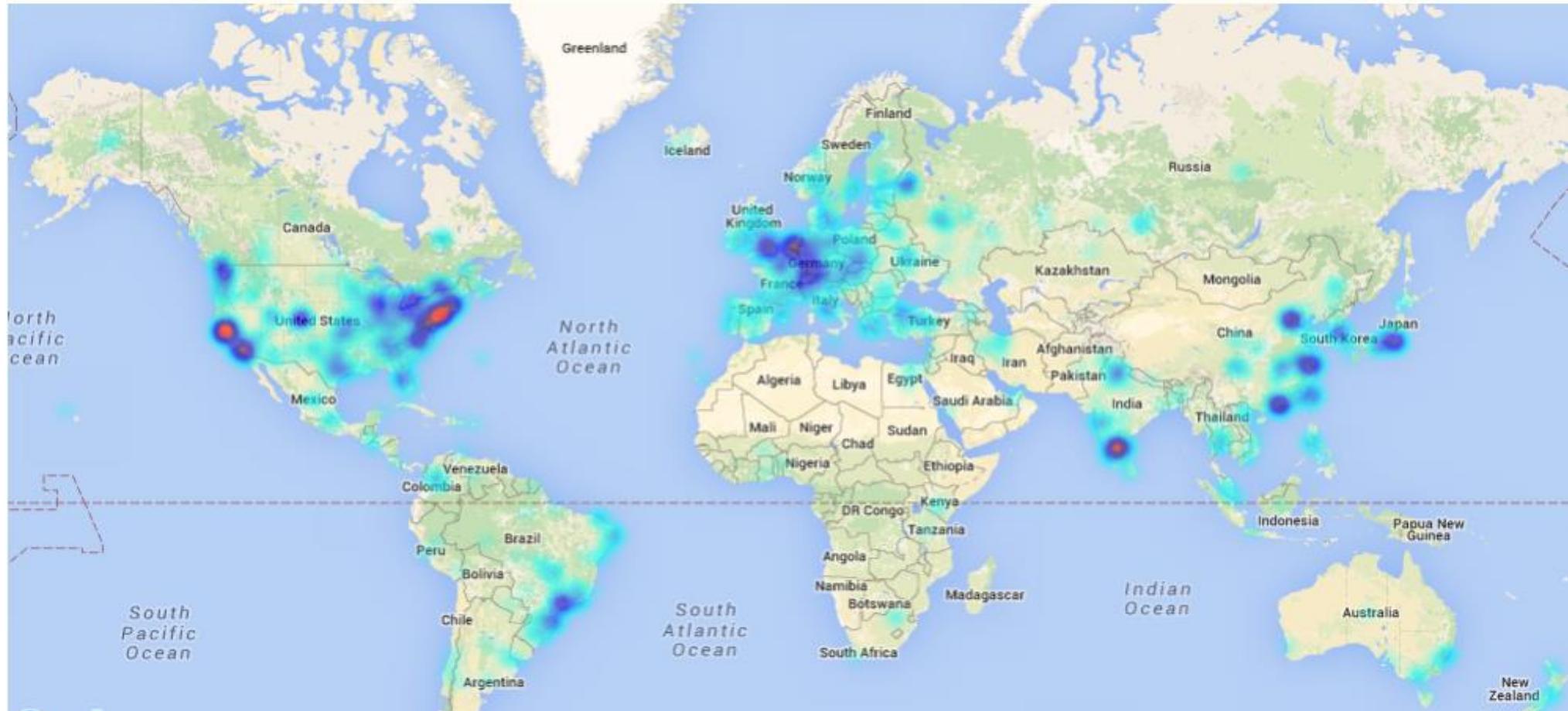
Julia GitHub stars

* Base language, does not include packages



A global community

More than 3 Million downloads, 2500 packages



James H. Wilkinson Prize
For Numerical Software

Stefan Karpinski
Viral B. Shah
Jeff Bezanson

(2019)



Forbes
30 under 30

Keno Fischer

(2019)

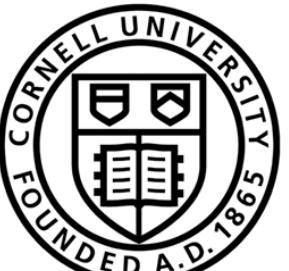


IEEE Babbage Prize
IEEE Fellow

Prof. Alan Edelman

(2018)





EMORY
UNIVERSITY



S
Stanford
University



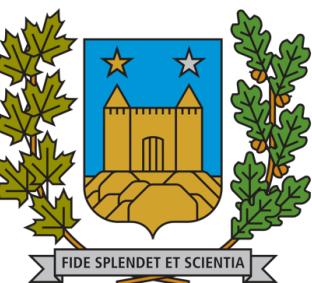
UNIVERSITY
of
GLASGOW



UNIVERSITE
PAUL
SABATIER



TOULOUSE III



CU THE CITY
UNIVERSITY
OF
NEW YORK

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



TOKYO METROPOLITAN UNIVERSITY
首都大学東京



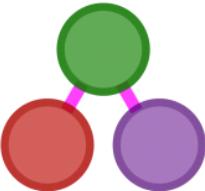
UCLA

AGH

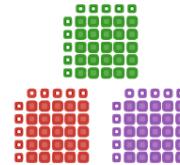
Best in class packages



Differential
Equations



Graph Processing



Data Science



Image Processing



Deep Learning



Mathematical
Optimization



Signal Processing



Computational
Biology

Recommended talks

[Nick Eubank: What Julia Offers Academic Researchers](#)

[George Datseris: Why Julia is the most suitable language for science](#)