# INTRODUCTION TO julia

## Combining fast code writing with efficient code running

By Robbe Ceulemans

University of Antwerp

University of Antwerp
TQC I Theory of Quantum Systems and Complex Systems

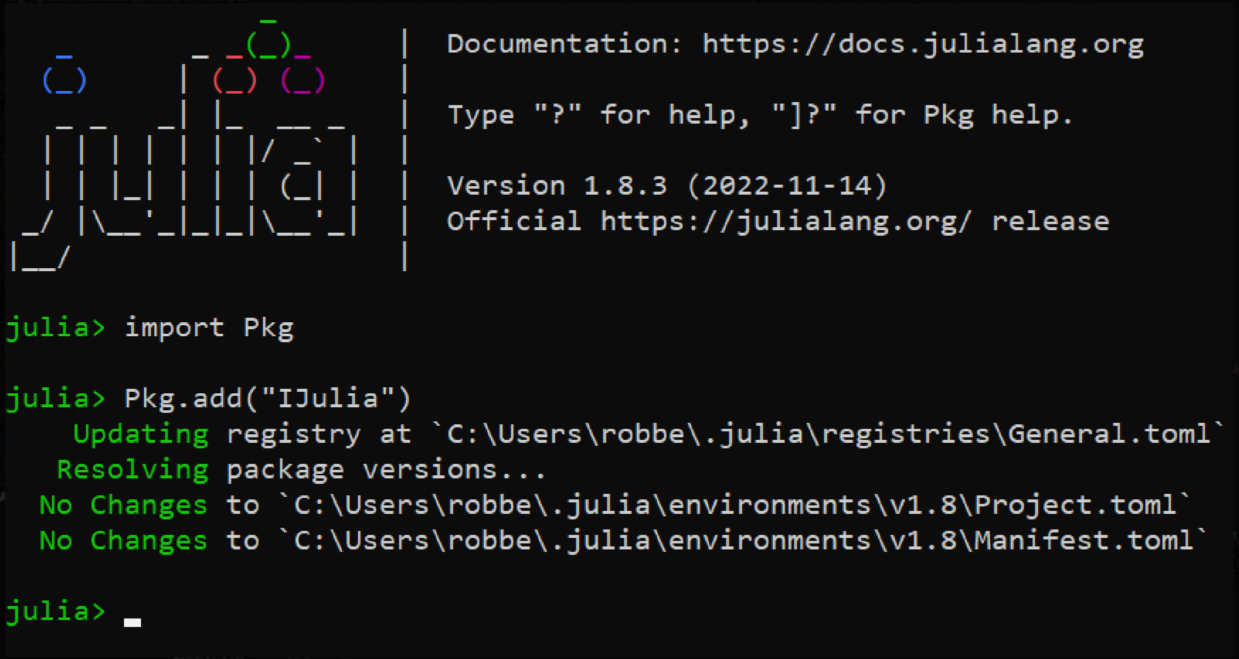
# OVERVIEW

- ❑ What is julia?
- ❑ Introduction to syntax

< Coffee break!

- ❑ Your turn to try
- ❑ Community today: Who uses Julia?

# BUT FIRST…

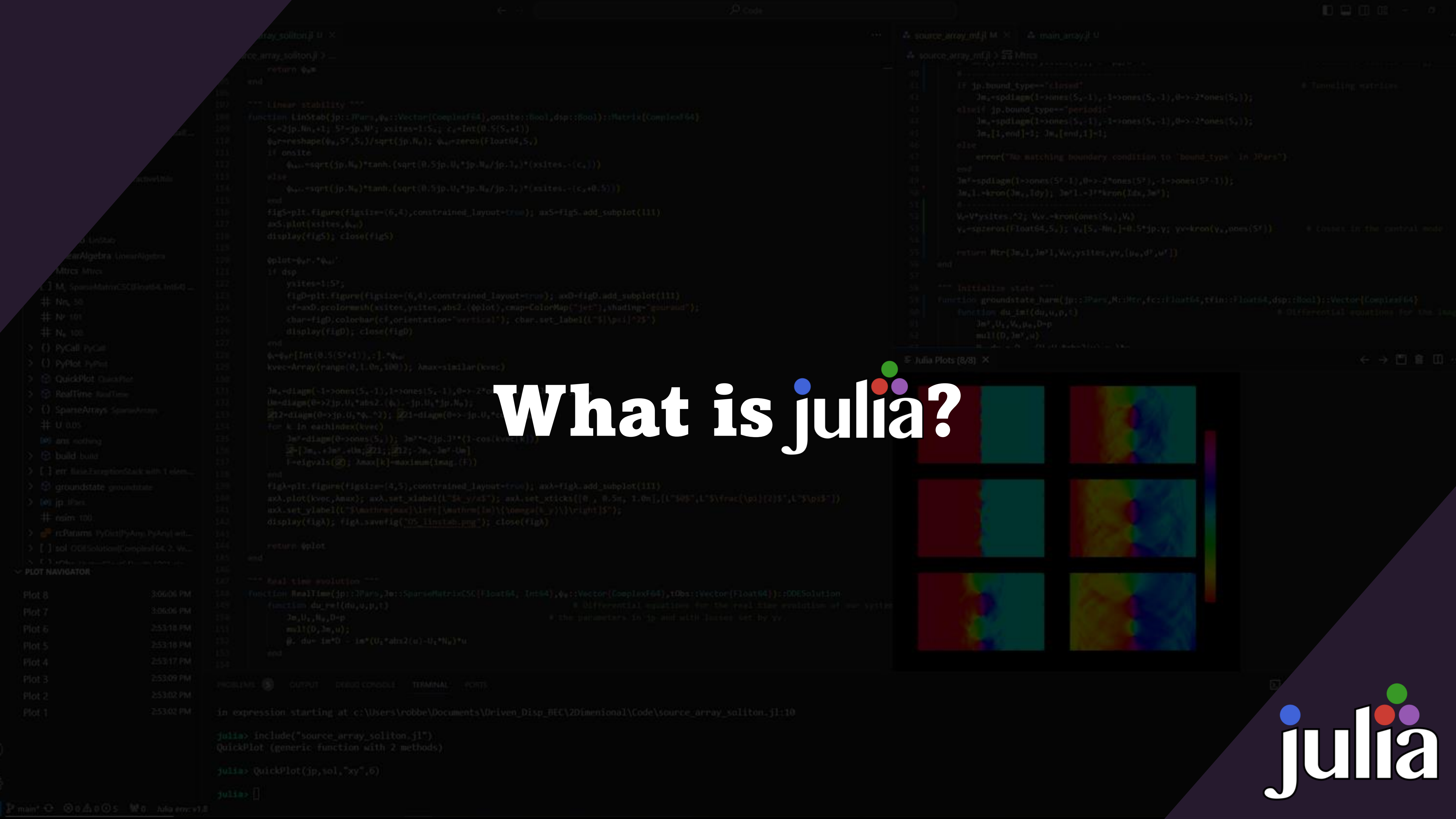


```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.8.3 (2022-11-14)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> import Pkg

julia> Pkg.add("IJulia")
    Updating registry at `C:\Users\robbe\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\robbe\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\robbe\.julia\environments\v1.8\Manifest.toml`

julia>
```

**Go to:**

https://github.com/RobbeCeulemans/Introtojulia

# What is julia?

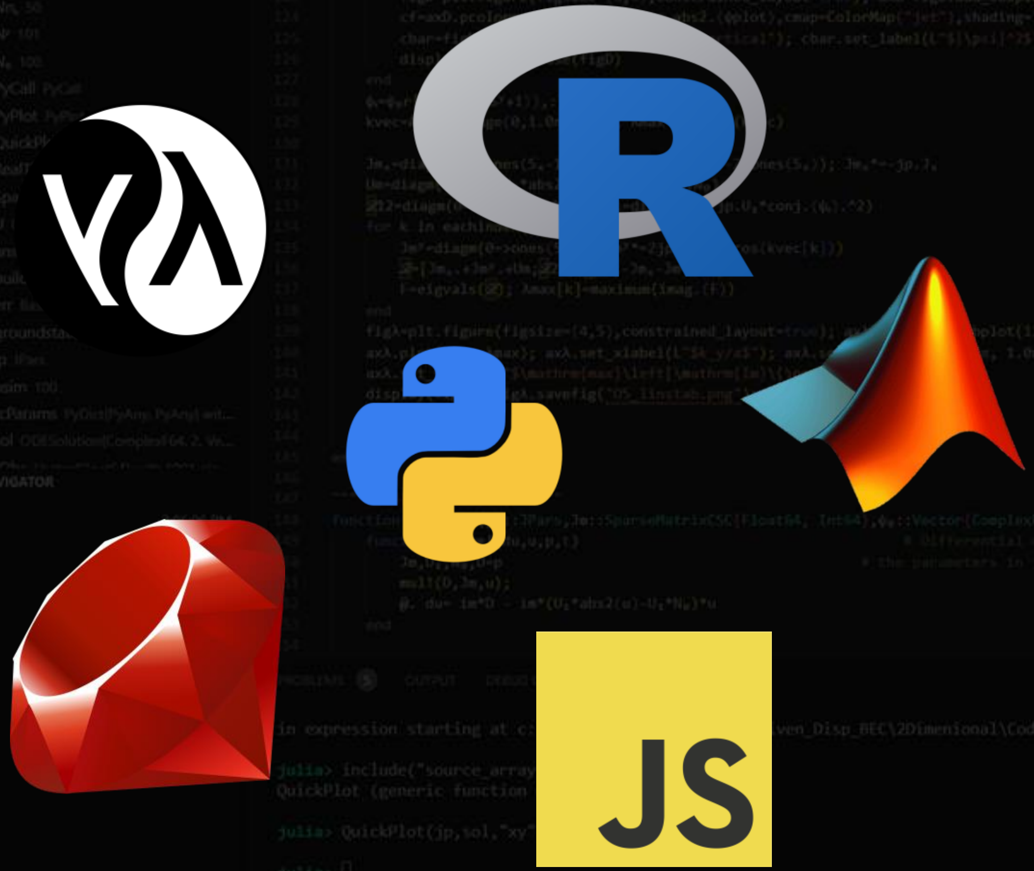


```
in expression starting at c:\Users\robbe\Documents\Driven_Disp_BEC\2Dimenional\Code\source_array_soliton.jl:10

julia> include("source_array_soliton.jl")
QuickPlot (generic function with 2 methods)

julia> QuickPlot(jp,sol,"xy",6)

julia> 
```

# Problem: What language?

# Problem: What language?

## Scripting languages



## Compiled languages

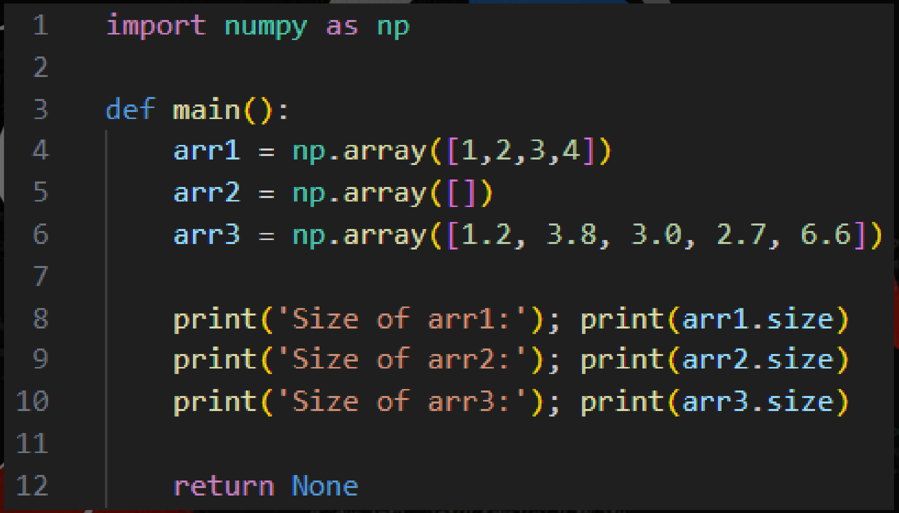
# Problem: What language?

## Dynamic

## Static

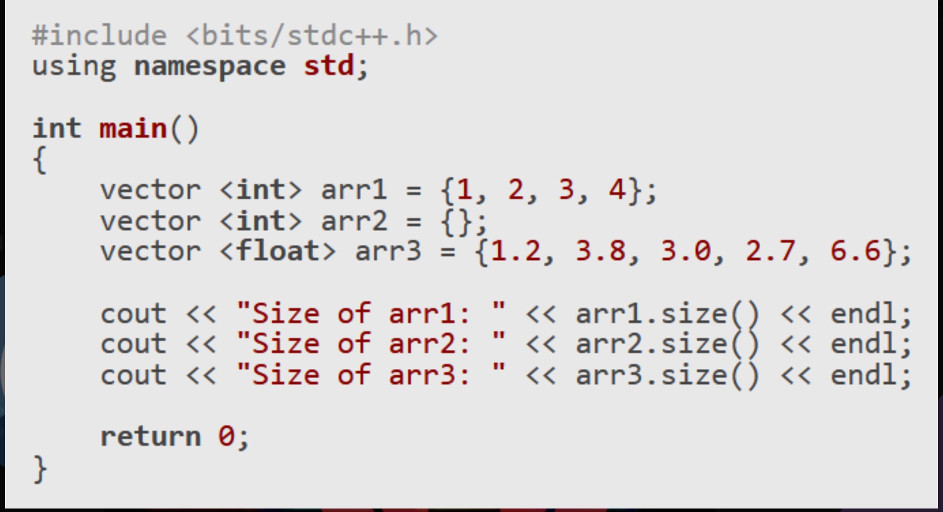
# Problem: What language?

## Dynamic



```python
import numpy as np

def main():
    arr1 = np.array([1,2,3,4])
    arr2 = np.array([])
    arr3 = np.array([1.2, 3.8, 3.0, 2.7, 6.6])

    print('Size of arr1:'); print(arr1.size)
    print('Size of arr2:'); print(arr2.size)
    print('Size of arr3:'); print(arr3.size)

    return None
```

## Static



```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector <int> arr1 = {1, 2, 3, 4};
    vector <int> arr2 = {};
    vector <float> arr3 = {1.2, 3.8, 3.0, 2.7, 6.6};

    cout << "Size of arr1: " << arr1.size() << endl;
    cout << "Size of arr2: " << arr2.size() << endl;
    cout << "Size of arr3: " << arr3.size() << endl;

    return 0;
}
```

# Problem: What language?

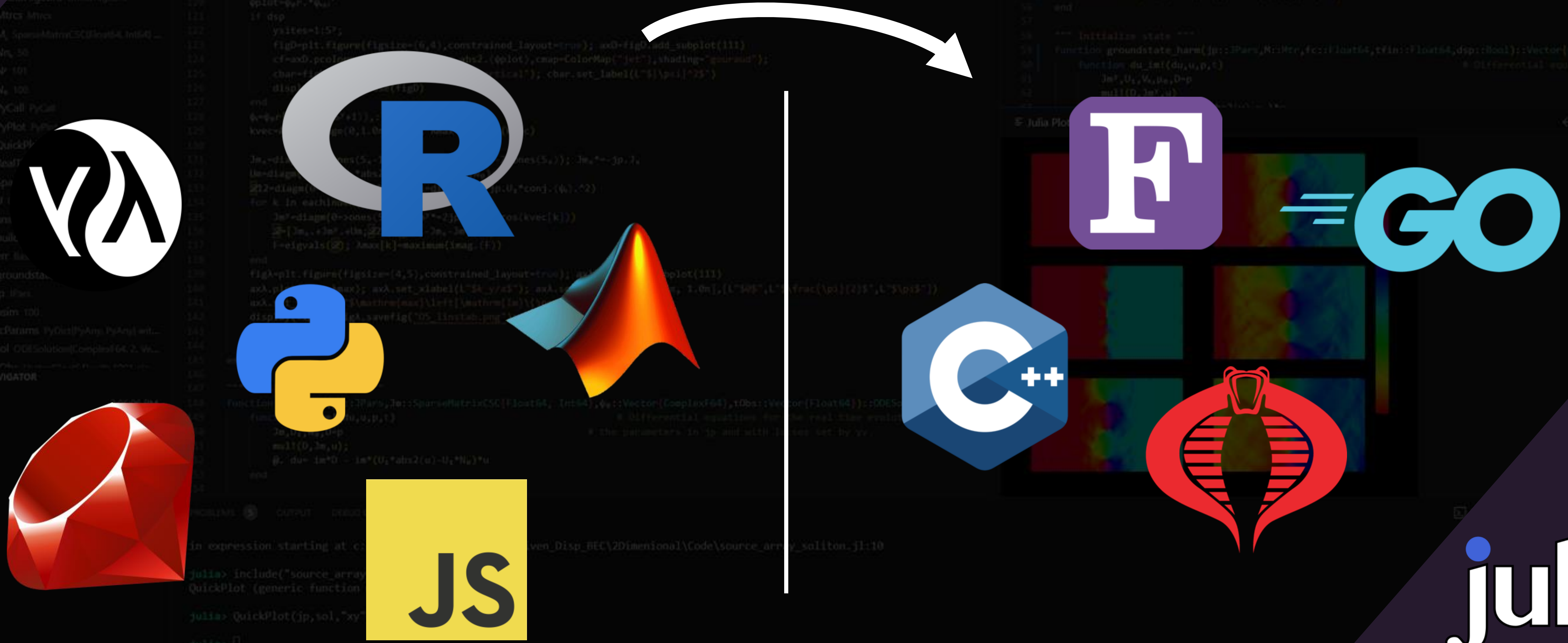
## Efficient writing

## Performance

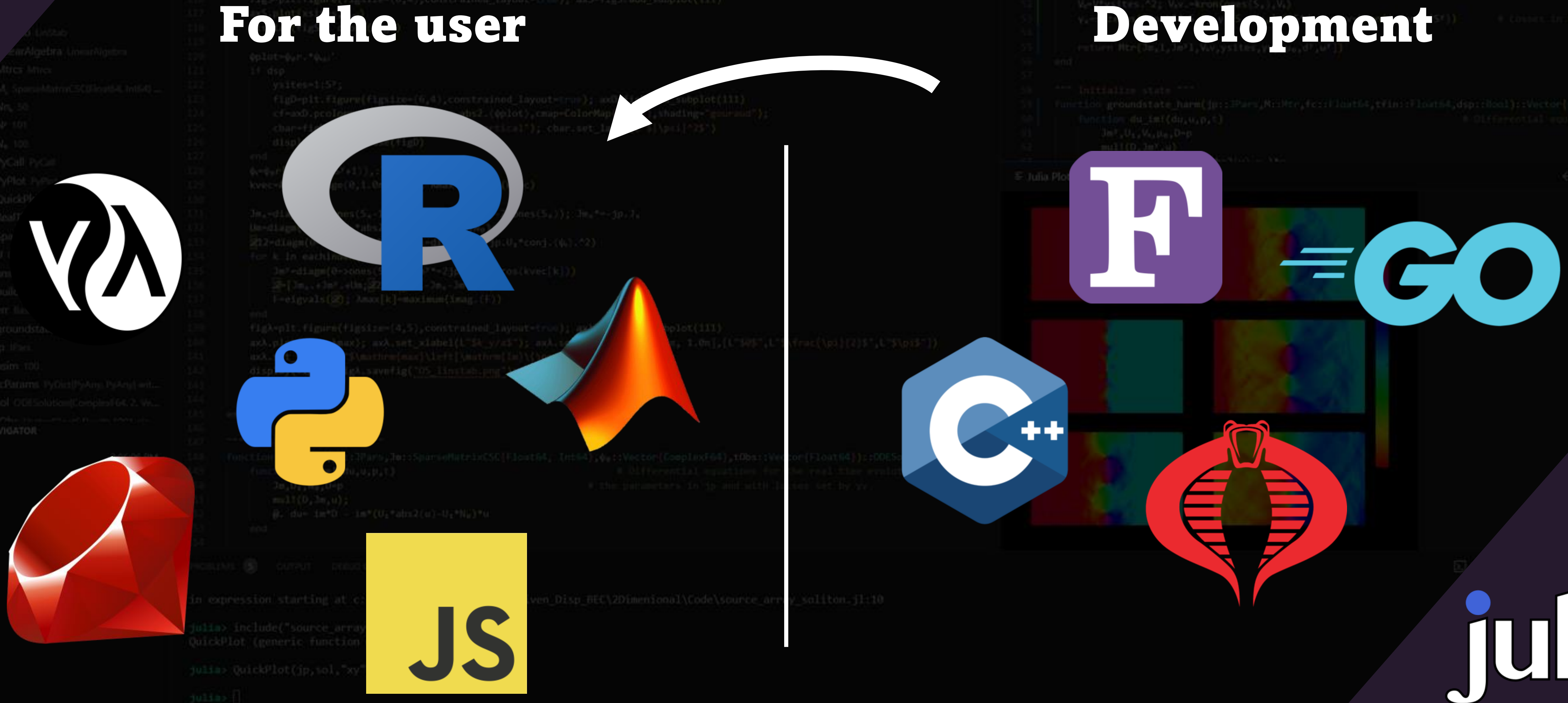
# Two-Language Problem

Proof of concept

Production



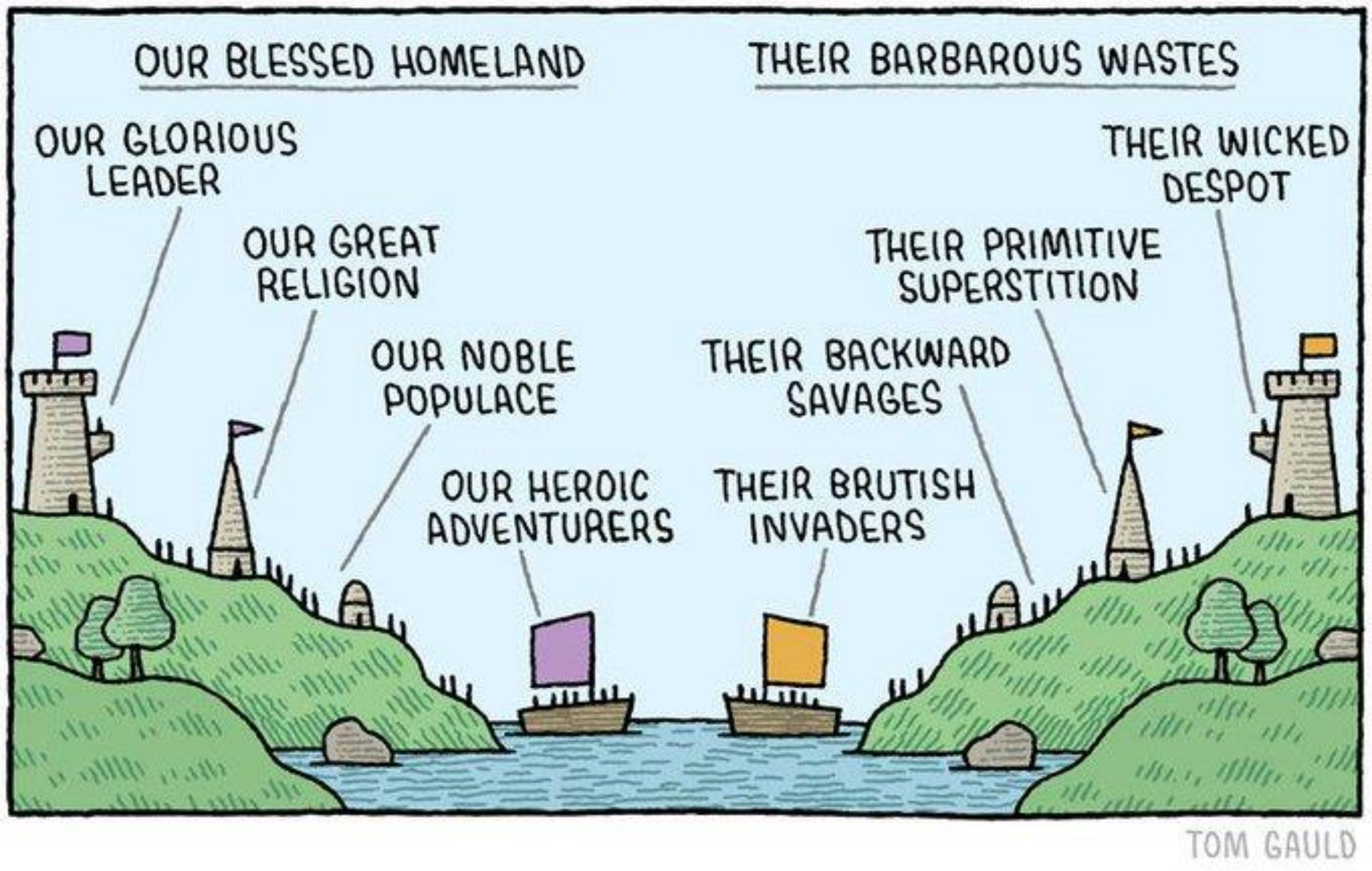
# Two-Culture Problem



# Two-Culture Problem

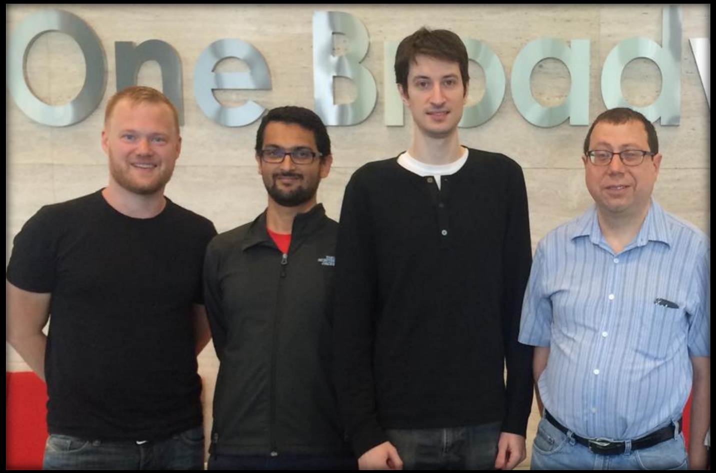
## For the user

## Development



# Where does julia fit in?

**Development started at MIT in 2009**
**First version in 2012 == julia 0.1**



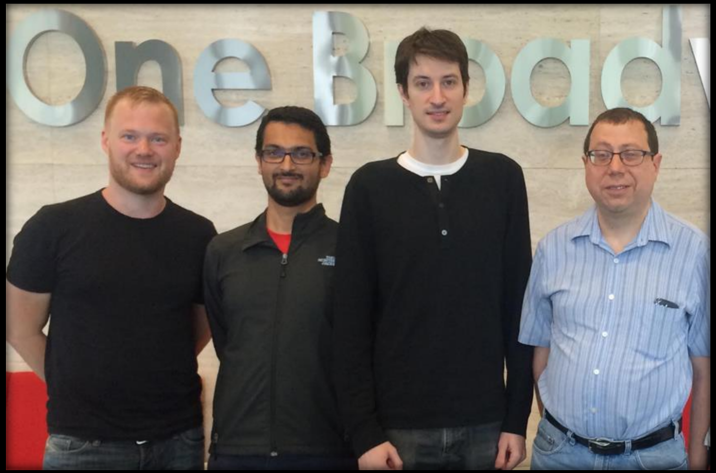
**From left to right: Stefan Karpinski, Viral B. Shah, Jeff Bezanson and prof. Alan Edelman**

*"We want a language that's open source, with a liberal license. We want the speed of* ***C*** *with the dynamism of* ***Ruby****. … We want something as usable for general programming as* ***Python****, as easy for statistics as* ***R****, as natural for string processing as* ***Perl****, as powerful for linear algebra as* ***Matlab****, …"* [1]

[1] J. Bezanson, S Karpinski, V.B. Shah, and A. Edelman, *Why we created Julia*, The Julia Language Blog, https://julialang.org/blog/2012/02/why-we-created-julia/, (2012).

# Where does julia fit in?

Development started at MIT in 2009
First version in 2012 == julia 0.1



From left to right: Stefan Karpinski, Viral B. Shah, Jeff Bezanson and prof. Alan Edelman

**Goal**: Combining best of two worlds

Scripting languages + Compiling languages

**Result**: Flexible and easy-to-use programming language with a performance comparable to traditional languages like C or Fortran

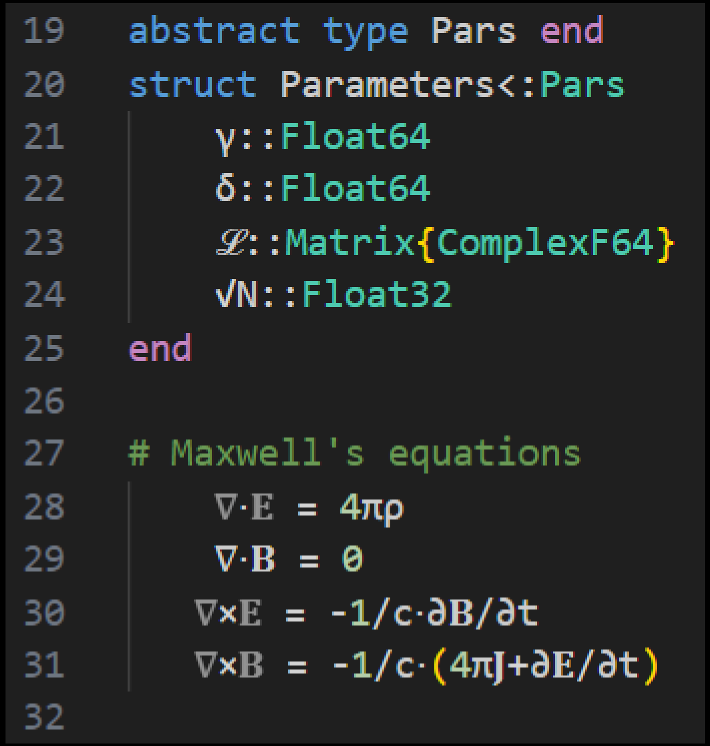
August 2023: #20 in TIOBE index[1]

[1] www.tiobe.com/tiobe-index/

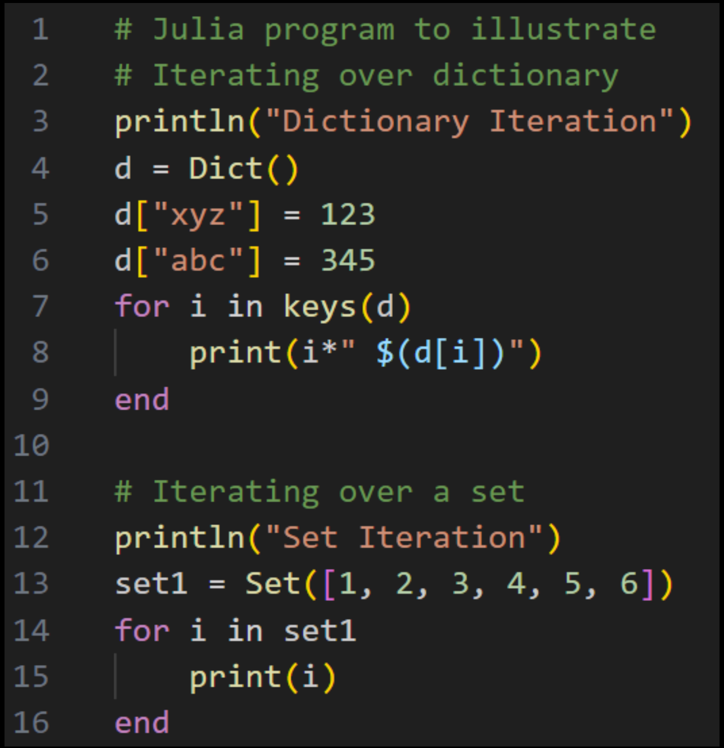
# Key properties

❑ **User friendly syntax**

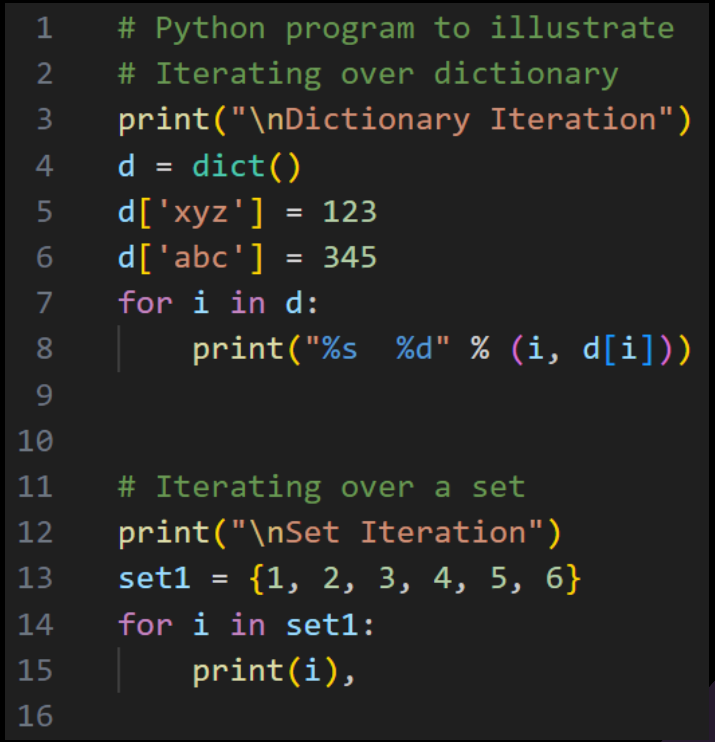
**Support for Unicode and $\LaTeX$**



```julia
abstract type Pars end
struct Parameters<:Pars
    γ::Float64
    δ::Float64
    ℒ::Matrix{ComplexF64}
    √N::Float32
end

# Maxwell's equations
    ∇⋅E = 4πρ
    ∇⋅B = 0
   ∇×E = -1/c⋅∂B/∂t
   ∇×B = -1/c⋅(4πJ+∂E/∂t)

```

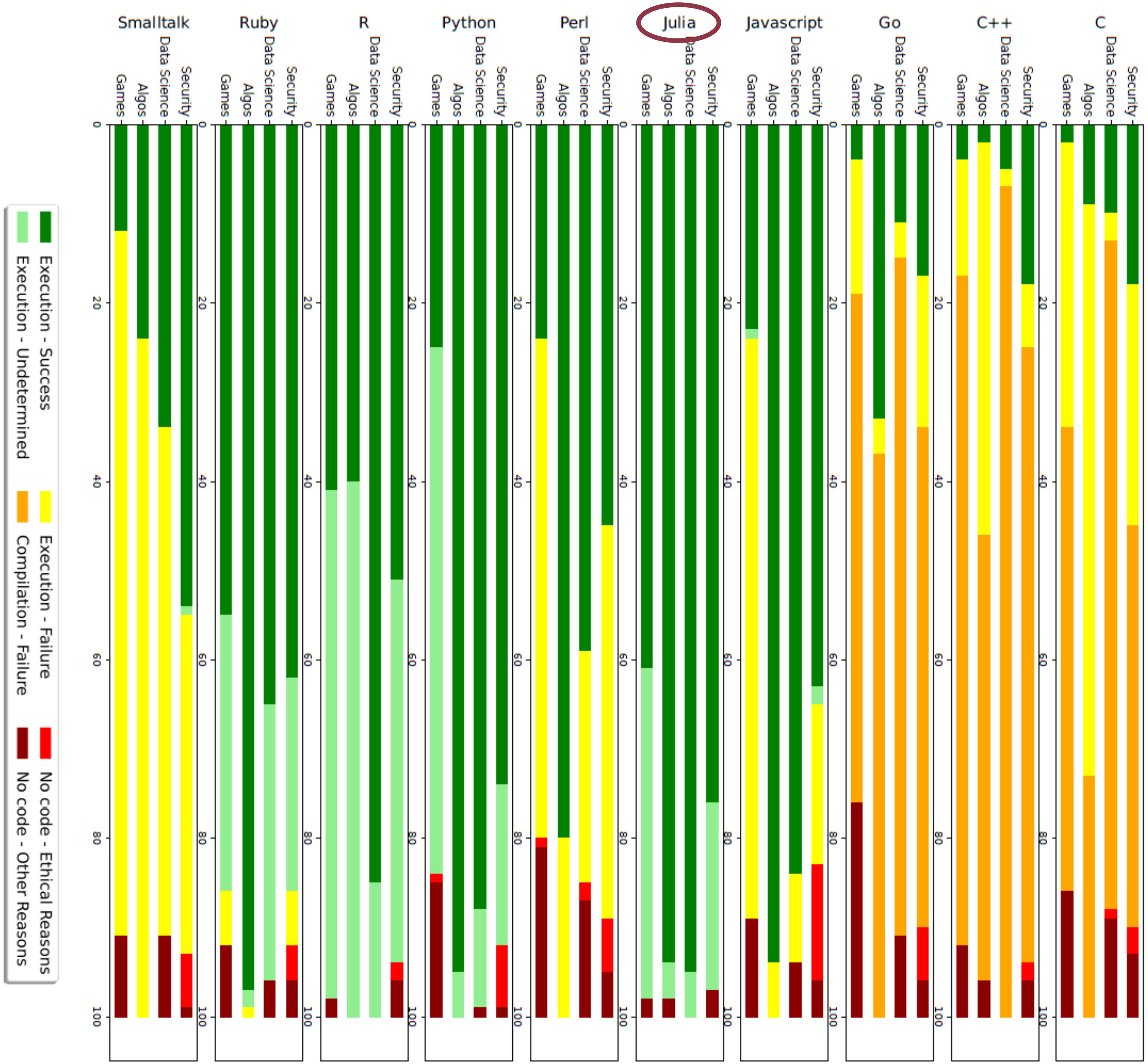


```julia
# Julia program to illustrate
# Iterating over dictionary
println("Dictionary Iteration")
d = Dict()
d["xyz"] = 123
d["abc"] = 345
for i in keys(d)
    print(i*" $(d[i])")
end

# Iterating over a set
println("Set Iteration")
set1 = Set([1, 2, 3, 4, 5, 6])
for i in set1
    print(i)
end
```



```python
# Python program to illustrate
# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d:
    print("%s  %d" % (i, d[i]))


# Iterating over a set
print("\nSet Iteration")
set1 = {1, 2, 3, 4, 5, 6}
for i in set1:
    print(i),

```

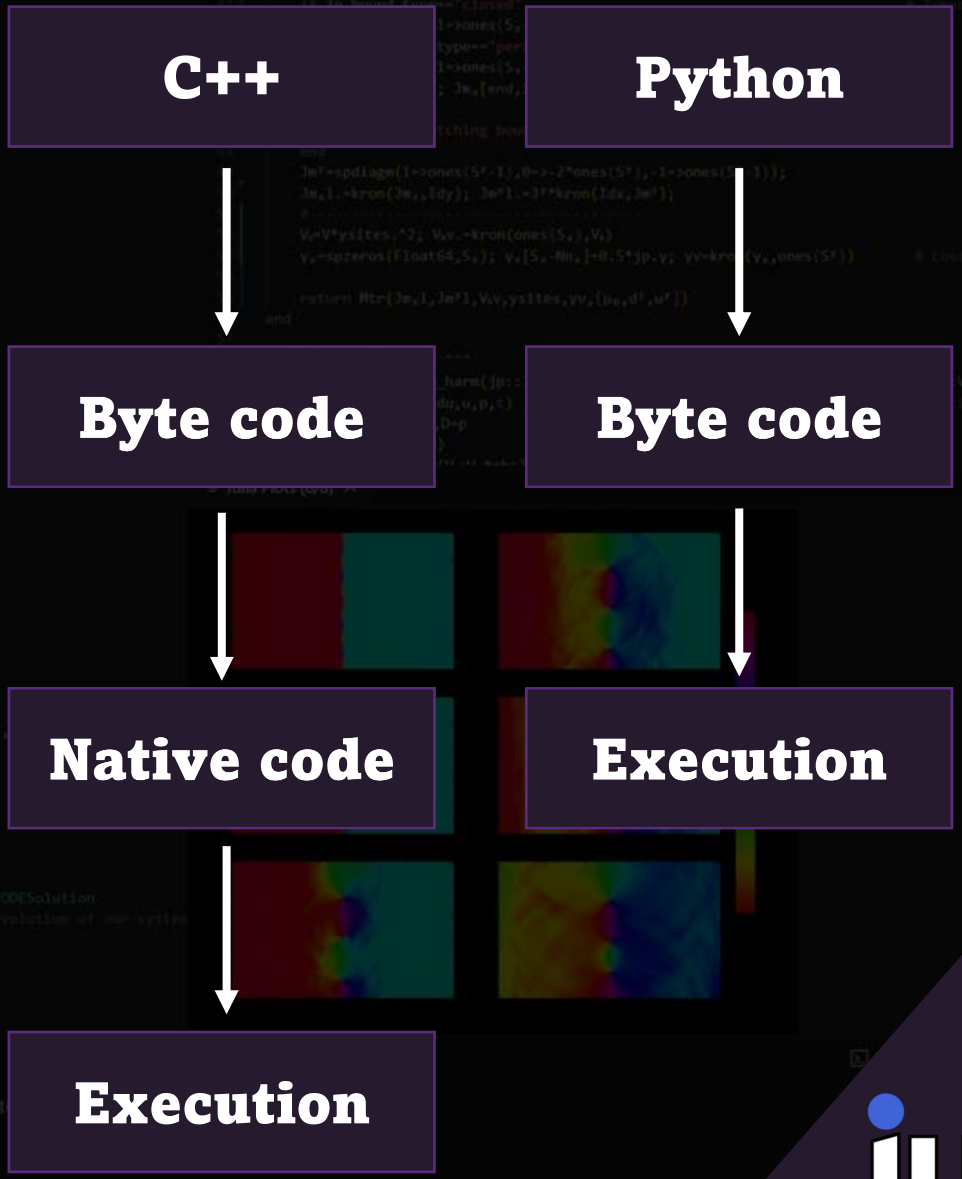
https://cheatsheets.quantecon.org/



Source: A. Buscemi,
arXiv:2308.04477v1 (2023)

# Key properties

- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation

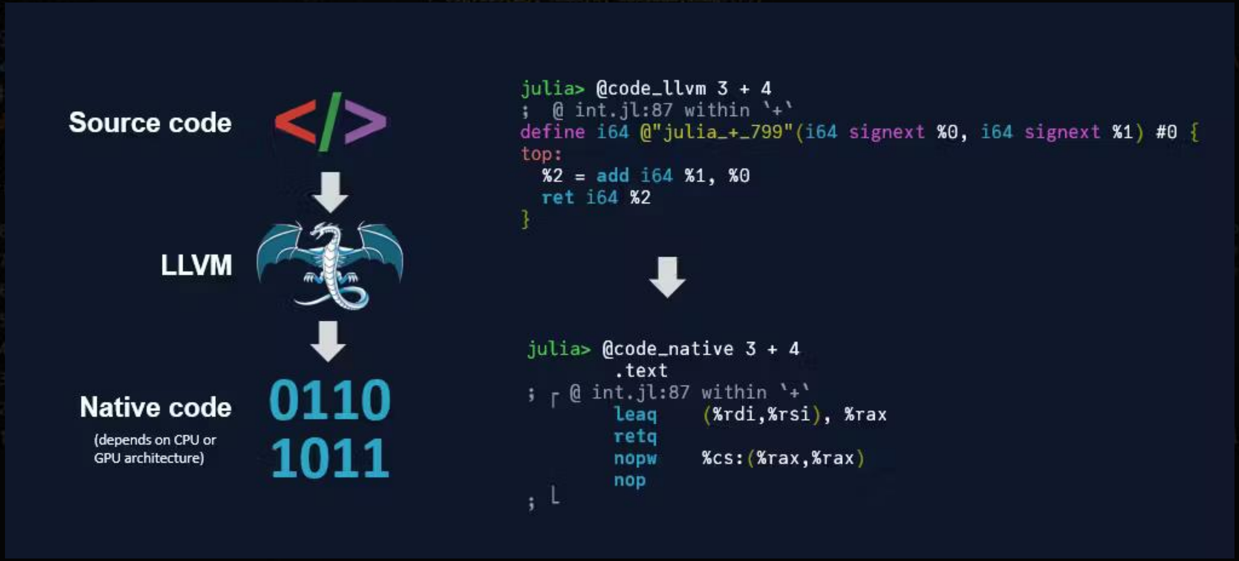


C++ → Byte code → Native code → Execution

Python → Byte code → Execution

# Key properties

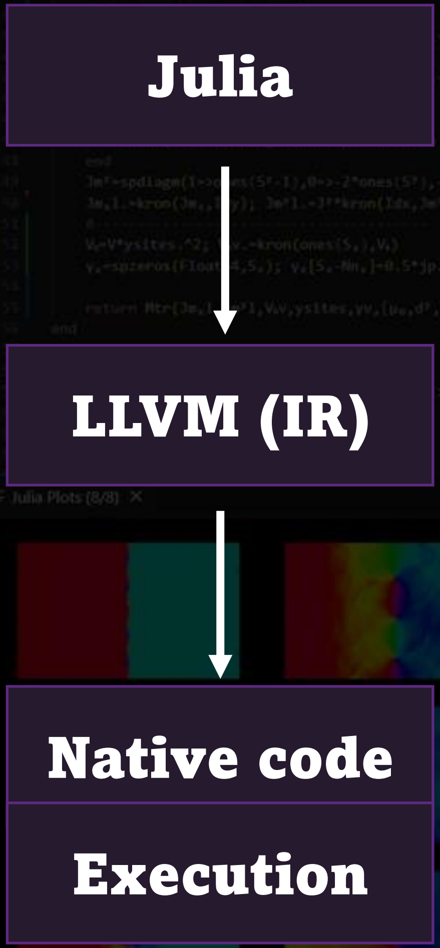
- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation
  - ↘ Dynamic recompilation



```
julia> @code_llvm 3 + 4
;  @ int.jl:87 within `+`
define i64 @"julia_+_799"(i64 signext %0, i64 signext %1) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

```
julia> @code_native 3 + 4
        .text
; ┌ @ int.jl:87 within `+`
        leaq    (%rdi,%rsi), %rax
        retq
        nopw    %cs:(%rax,%rax)
        nop
; └
```

Source: M. Cox, The scientific coder, https://shorturl.at/gDHR8 (2023)



Julia

LLVM (IR)

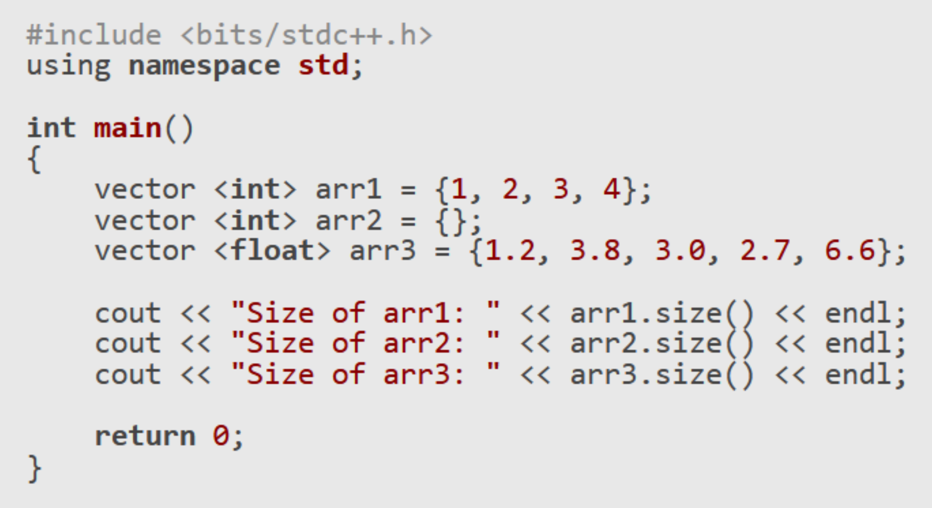
Native code

Execution

# Key properties

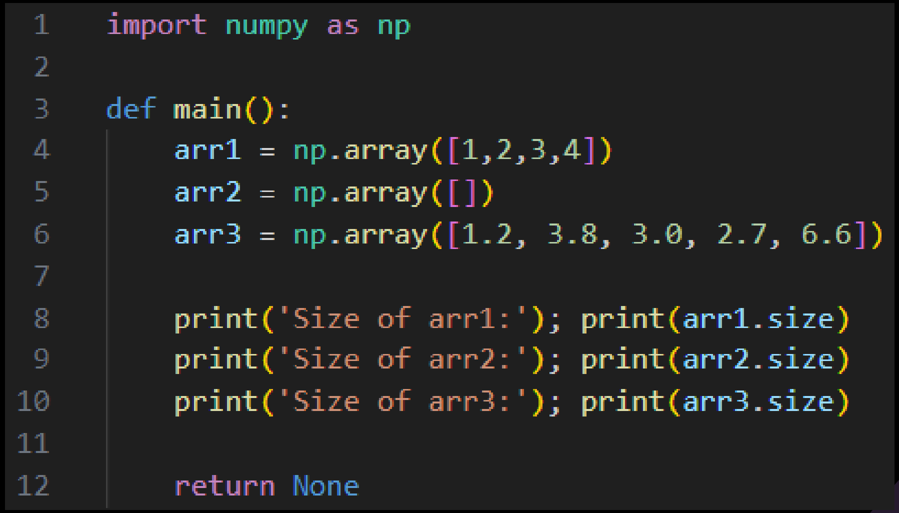
- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation
  - ↘ Dynamic recompilation
- ❑ *Type-oriented* programming

## Statically typed



```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector <int> arr1 = {1, 2, 3, 4};
    vector <int> arr2 = {};
    vector <float> arr3 = {1.2, 3.8, 3.0, 2.7, 6.6};

    cout << "Size of arr1: " << arr1.size() << endl;
    cout << "Size of arr2: " << arr2.size() << endl;
    cout << "Size of arr3: " << arr3.size() << endl;

    return 0;
}
```

## vs. Dynamic typing



```
import numpy as np

def main():
    arr1 = np.array([1,2,3,4])
    arr2 = np.array([])
    arr3 = np.array([1.2, 3.8, 3.0, 2.7, 6.6])

    print('Size of arr1:'); print(arr1.size)
    print('Size of arr2:'); print(arr2.size)
    print('Size of arr3:'); print(arr3.size)

    return None
```

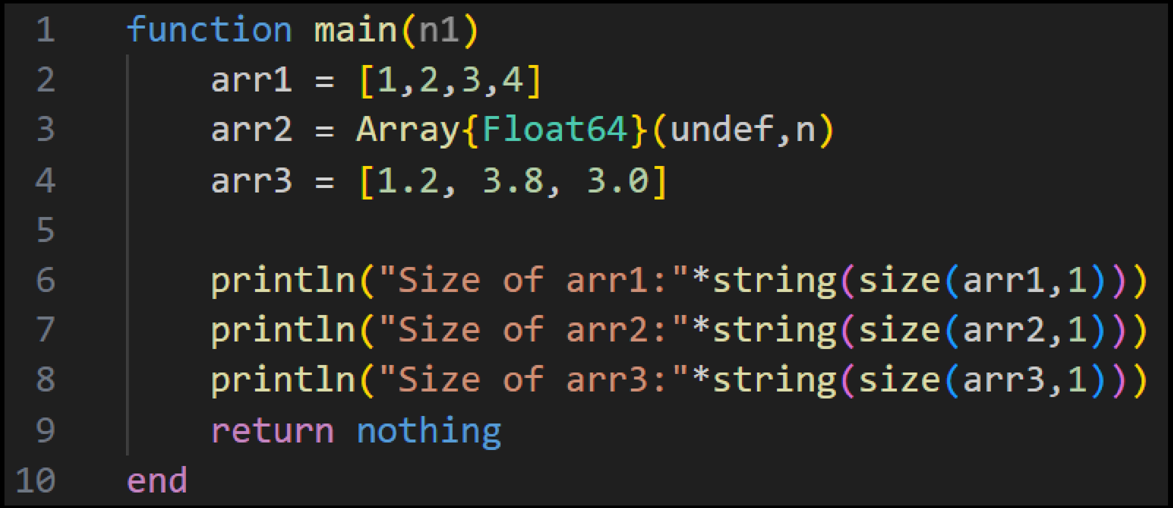
# Key properties

❑ **User friendly syntax**

❑ **Just-in-time (JIT) compilation**

↘ **Dynamic recompilation**

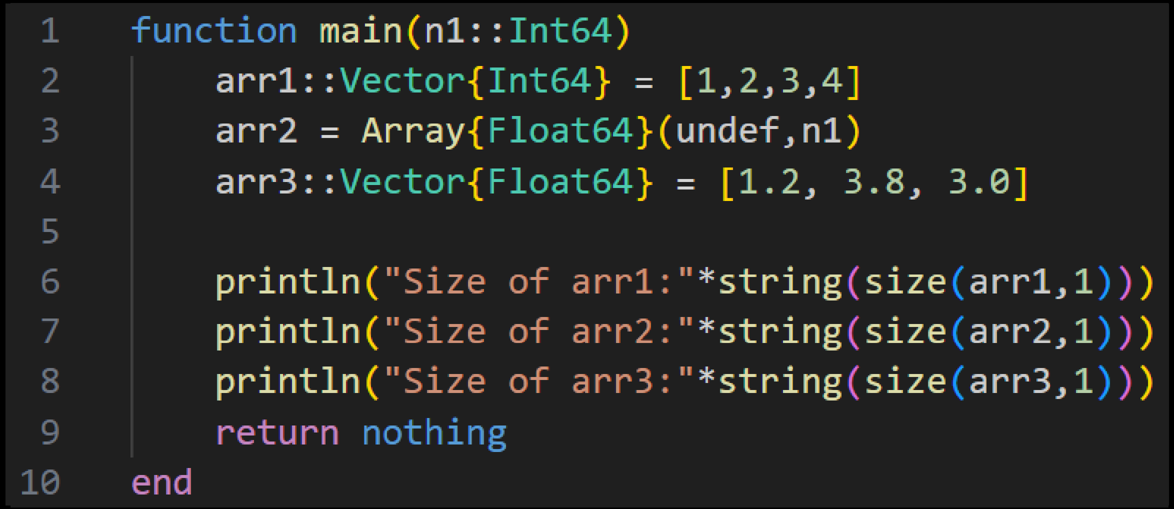
❑ ***Type-oriented* programming**



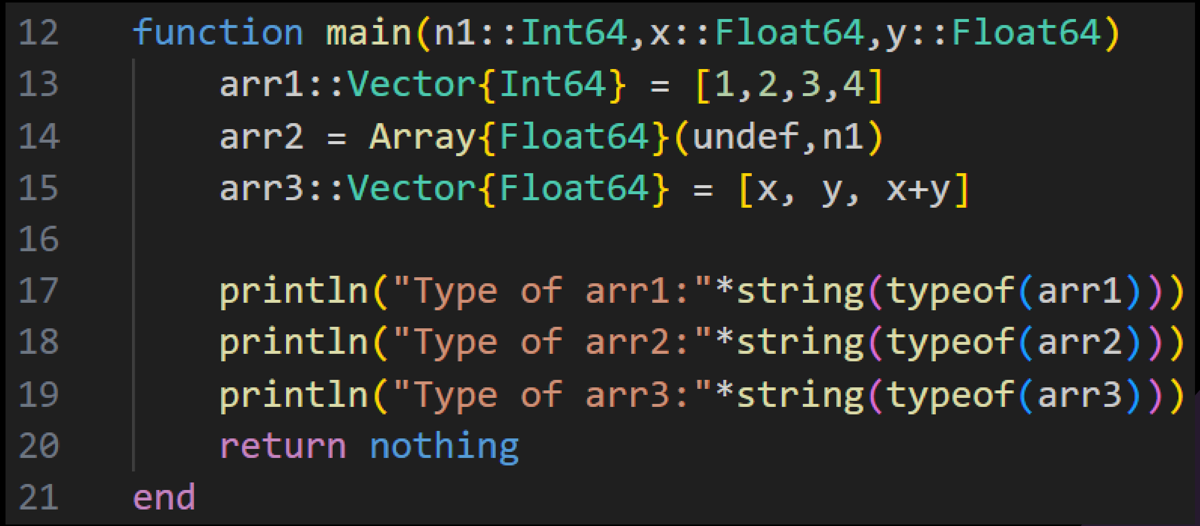
```julia
function main(n1)
    arr1 = [1,2,3,4]
    arr2 = Array{Float64}(undef,n)
    arr3 = [1.2, 3.8, 3.0]

    println("Size of arr1:"*string(size(arr1,1)))
    println("Size of arr2:"*string(size(arr2,1)))
    println("Size of arr3:"*string(size(arr3,1)))
    return nothing
end
```

# Key properties

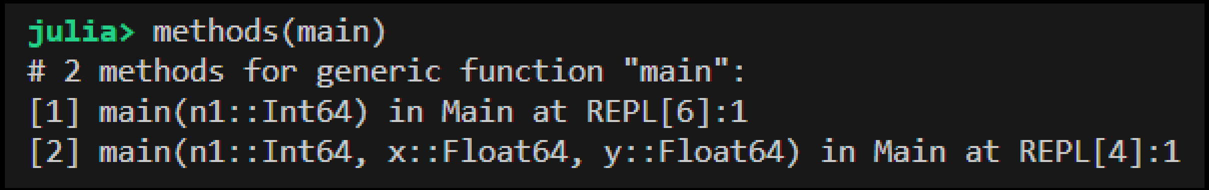
- ❑ **User friendly syntax**
- ❑ **Just-in-time (JIT) compilation**
  - ↘ **Dynamic recompilation**
- ❑ ***Type-oriented* programming**
  - ↘ **Multiple dispatch**



```julia
function main(n1::Int64)
    arr1::Vector{Int64} = [1,2,3,4]
    arr2 = Array{Float64}(undef,n1)
    arr3::Vector{Float64} = [1.2, 3.8, 3.0]

    println("Size of arr1:"*string(size(arr1,1)))
    println("Size of arr2:"*string(size(arr2,1)))
    println("Size of arr3:"*string(size(arr3,1)))
    return nothing
end
```



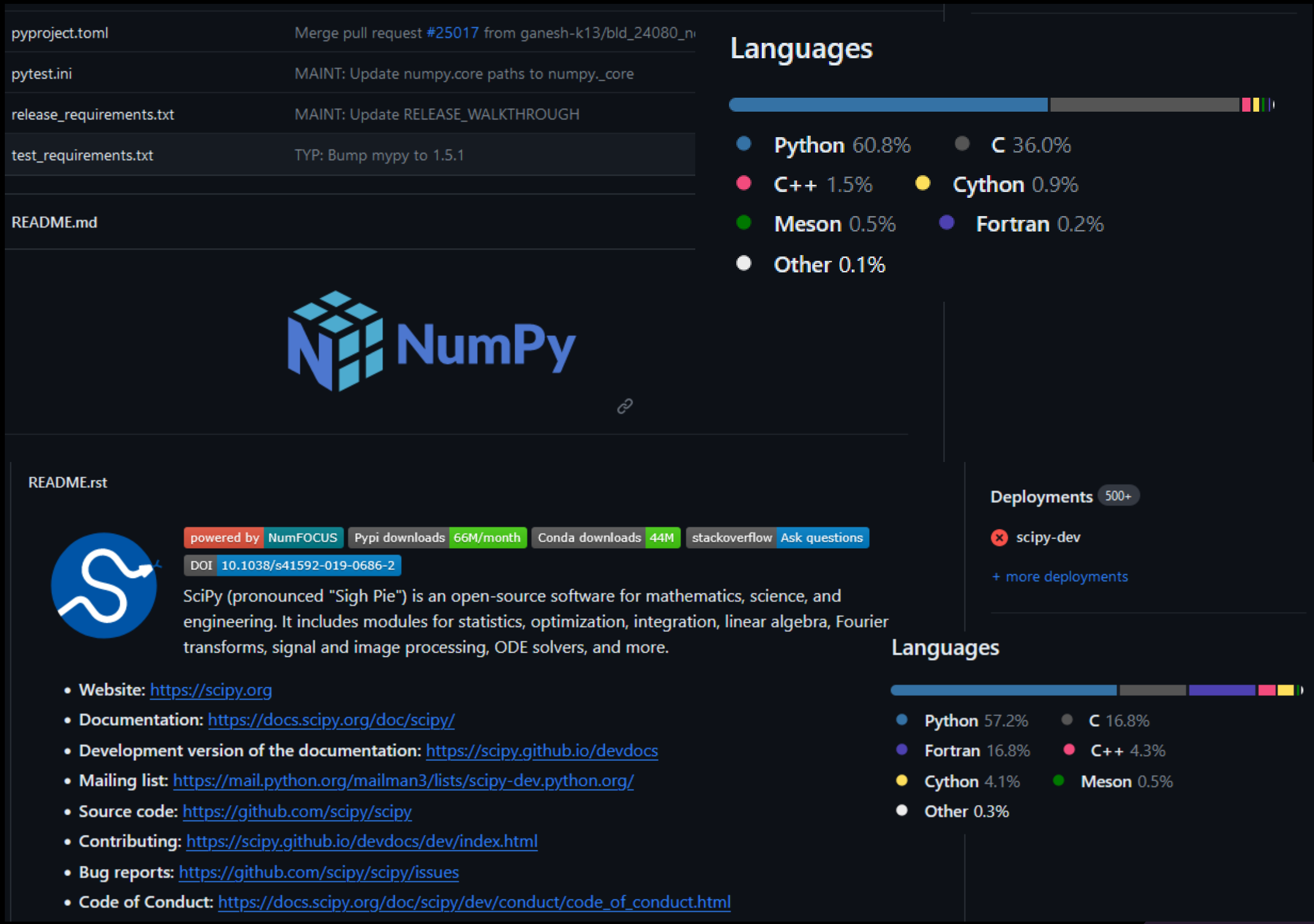
```julia
function main(n1::Int64,x::Float64,y::Float64)
    arr1::Vector{Int64} = [1,2,3,4]
    arr2 = Array{Float64}(undef,n1)
    arr3::Vector{Float64} = [x, y, x+y]

    println("Type of arr1:"*string(typeof(arr1)))
    println("Type of arr2:"*string(typeof(arr2)))
    println("Type of arr3:"*string(typeof(arr3)))
    return nothing
end
```



```
julia> methods(main)
# 2 methods for generic function "main":
[1] main(n1::Int64) in Main at REPL[6]:1
[2] main(n1::Int64, x::Float64, y::Float64) in Main at REPL[4]:1
```

# Key properties

- ❑ **User friendly syntax**
- ❑ **Just-in-time (JIT) compilation**
  - ↘ **Dynamic recompilation**
- ❑ ***Type-oriented* programming**
  - ↘ **Multiple dispatch**
- ❑ **Self-hosted packages**



| | |
|---|---|
| pyproject.toml | Merge pull request #25017 from ganesh-k13/bld_24080_n |
| pytest.ini | MAINT: Update numpy.core paths to numpy._core |
| release_requirements.txt | MAINT: Update RELEASE_WALKTHROUGH |
| test_requirements.txt | TYP: Bump mypy to 1.5.1 |

README.md

NumPy

Languages

● Python 60.8% ● C 36.0%
● C++ 1.5% ● Cython 0.9%
● Meson 0.5% ● Fortran 0.2%
● Other 0.1%

README.rst

powered by NumFOCUS Pypi downloads 66M/month Conda downloads 44M stackoverflow Ask questions
DOI 10.1038/s41592-019-0686-2

SciPy (pronounced "Sigh Pie") is an open-source software for mathematics, science, and engineering. It includes modules for statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing, ODE solvers, and more.

- **Website:** https://scipy.org
- **Documentation:** https://docs.scipy.org/doc/scipy/
- **Development version of the documentation:** https://scipy.github.io/devdocs
- **Mailing list:** https://mail.python.org/mailman3/lists/scipy-dev.python.org/
- **Source code:** https://github.com/scipy/scipy
- **Contributing:** https://scipy.github.io/devdocs/dev/index.html
- **Bug reports:** https://github.com/scipy/scipy/issues
- **Code of Conduct:** https://docs.scipy.org/doc/scipy/dev/conduct/code_of_conduct.html

Deployments 500+

☒ scipy-dev

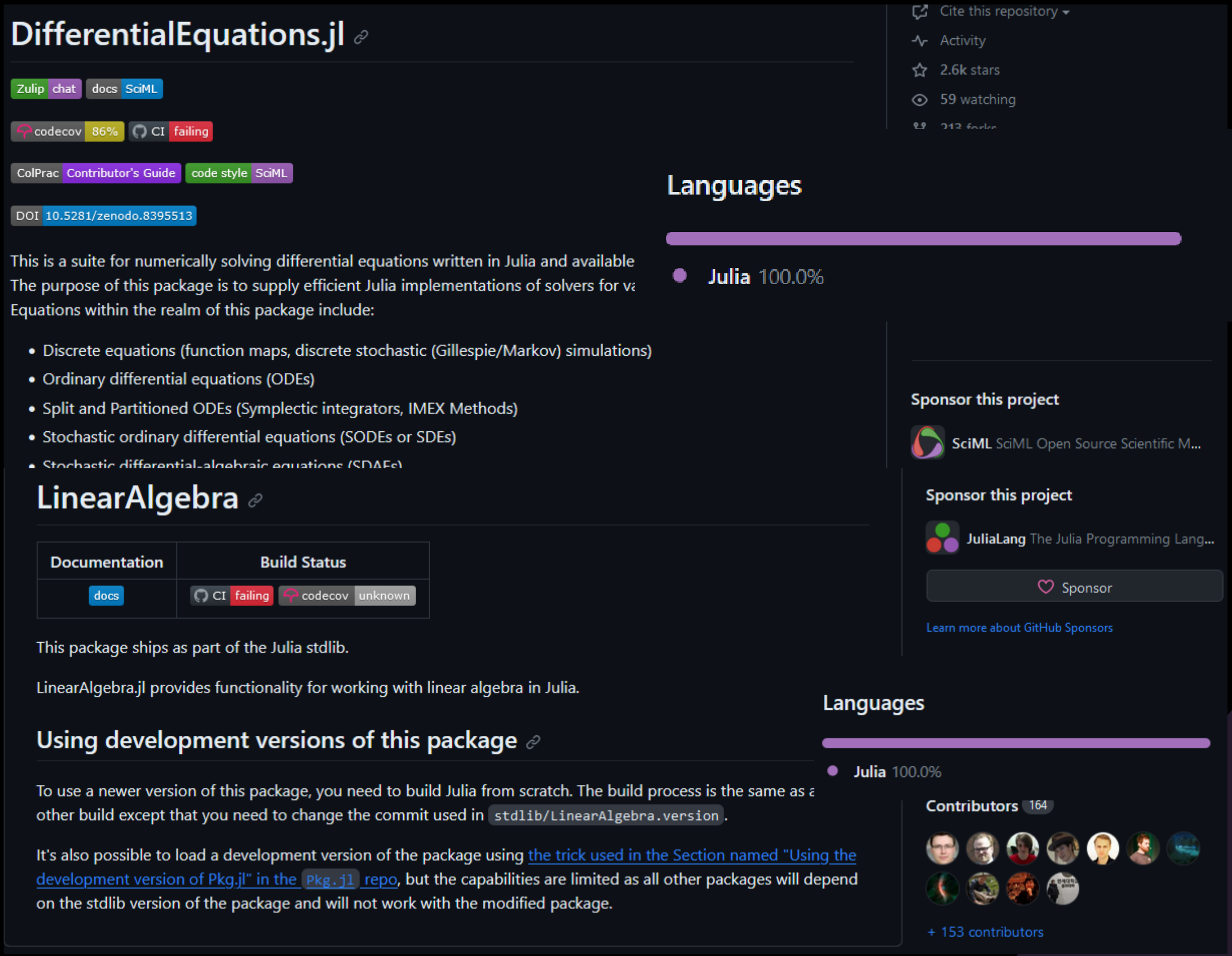
+ more deployments

Languages

● Python 57.2% ● C 16.8%
● Fortran 16.8% ● C++ 4.3%
● Cython 4.1% ● Meson 0.5%
● Other 0.3%

# Key properties

- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation
  - ↘ Dynamic recompilation
- ❑ *Type-oriented* programming
  - ↘ Multiple dispatch
- ❑ Self-hosted packages



## DifferentialEquations.jl

Zulip chat | docs SciML

codecov 86% | CI failing

ColPrac Contributor's Guide | code style SciML

DOI 10.5281/zenodo.8395513

This is a suite for numerically solving differential equations written in Julia and available
The purpose of this package is to supply efficient Julia implementations of solvers for va
Equations within the realm of this package include:

- Discrete equations (function maps, discrete stochastic (Gillespie/Markov) simulations)
- Ordinary differential equations (ODEs)
- Split and Partitioned ODEs (Symplectic integrators, IMEX Methods)
- Stochastic ordinary differential equations (SODEs or SDEs)
- Stochastic differential-algebraic equations (SDAEs)

## LinearAlgebra

| Documentation | Build Status |
|---|---|
| docs | CI failing codecov unknown |

This package ships as part of the Julia stdlib.

LinearAlgebra.jl provides functionality for working with linear algebra in Julia.

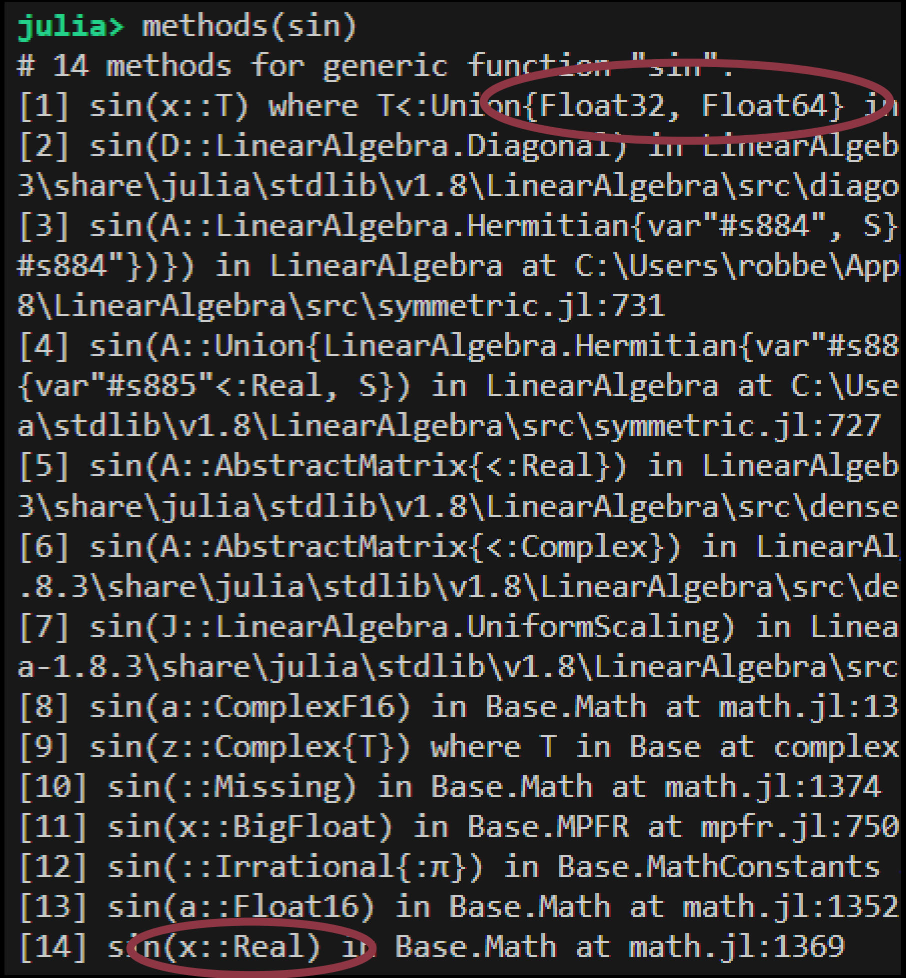
### Using development versions of this package

To use a newer version of this package, you need to build Julia from scratch. The build process is the same as a other build except that you need to change the commit used in `stdlib/LinearAlgebra.version`.

It's also possible to load a development version of the package using the trick used in the Section named "Using the development version of Pkg.jl" in the `Pkg.jl` repo, but the capabilities are limited as all other packages will depend on the stdlib version of the package and will not work with the modified package.

# Key properties

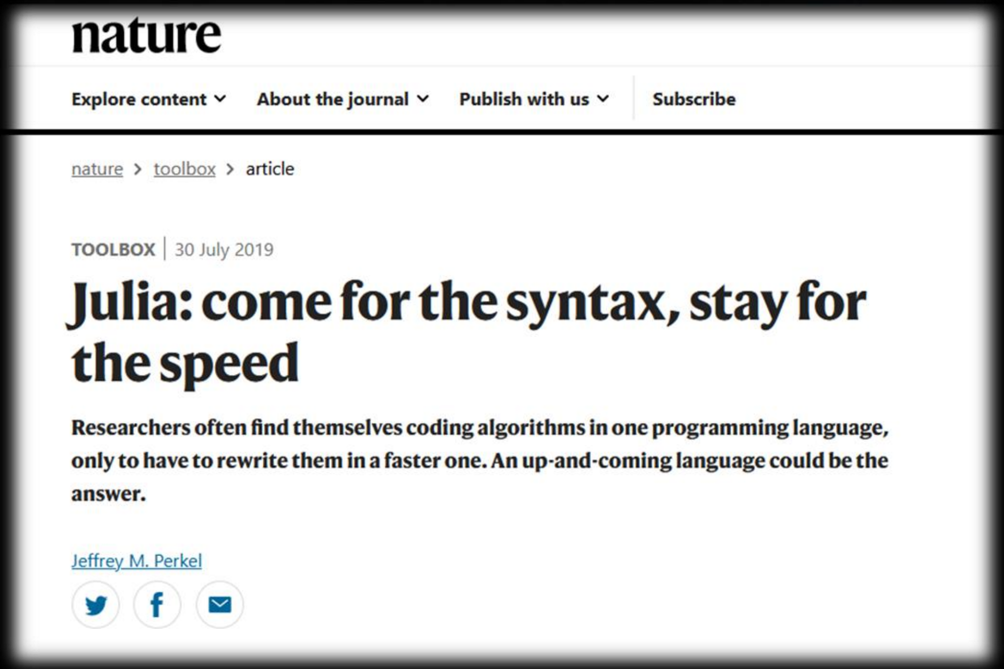
- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation
  - ↘ Dynamic recompilation
- ❑ *Type-oriented* programming
  - ↘ Multiple dispatch
- ❑ Self-hosted packages
  - ↘ Even base functions



```
julia> methods(sin)
# 14 methods for generic function "sin":
[1] sin(x::T) where T<:Union{Float32, Float64} in
[2] sin(D::LinearAlgebra.Diagonal) in LinearAlgeb
3\share\julia\stdlib\v1.8\LinearAlgebra\src\diago
[3] sin(A::LinearAlgebra.Hermitian{var"#s884", S}
#s884"})}) in LinearAlgebra at C:\Users\robbe\App
8\LinearAlgebra\src\symmetric.jl:731
[4] sin(A::Union{LinearAlgebra.Hermitian{var"#s88
{var"#s885"<:Real, S}) in LinearAlgebra at C:\Use
a\stdlib\v1.8\LinearAlgebra\src\symmetric.jl:727
[5] sin(A::AbstractMatrix{<:Real}) in LinearAlgeb
3\share\julia\stdlib\v1.8\LinearAlgebra\src\dense
[6] sin(A::AbstractMatrix{<:Complex}) in LinearAl
.8.3\share\julia\stdlib\v1.8\LinearAlgebra\src\de
[7] sin(J::LinearAlgebra.UniformScaling) in Linea
a-1.8.3\share\julia\stdlib\v1.8\LinearAlgebra\src
[8] sin(a::ComplexF16) in Base.Math at math.jl:13
[9] sin(z::Complex{T}) where T in Base at complex
[10] sin(::Missing) in Base.Math at math.jl:1374
[11] sin(x::BigFloat) in Base.MPFR at mpfr.jl:750
[12] sin(::Irrational{:π}) in Base.MathConstants
[13] sin(a::Float16) in Base.Math at math.jl:1352
[14] sin(x::Real) in Base.Math at math.jl:1369
```

# Key properties

- ❑ User friendly syntax
- ❑ Just-in-time (JIT) compilation
  - ↘ Dynamic recompilation
- ❑ *Type-oriented* programming
  - ↘ Multiple dispatch
- ❑ Self-hosted packages
  - ↘ Even base functions



nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature > toolbox > article

TOOLBOX | 30 July 2019

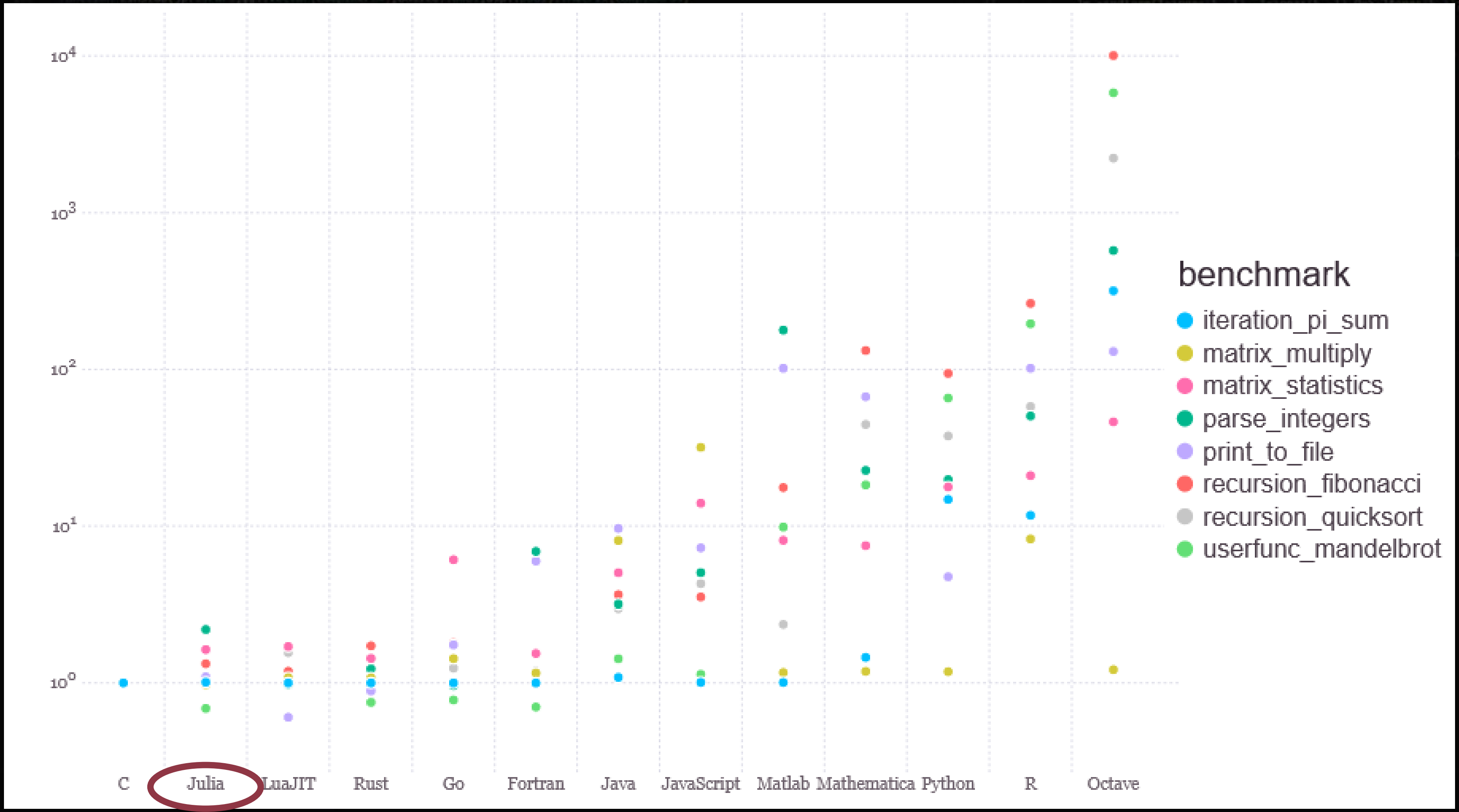
## Julia: come for the syntax, stay for the speed

**Researchers often find themselves coding algorithms in one programming language, only to have to rewrite them in a faster one. An up-and-coming language could be the answer.**

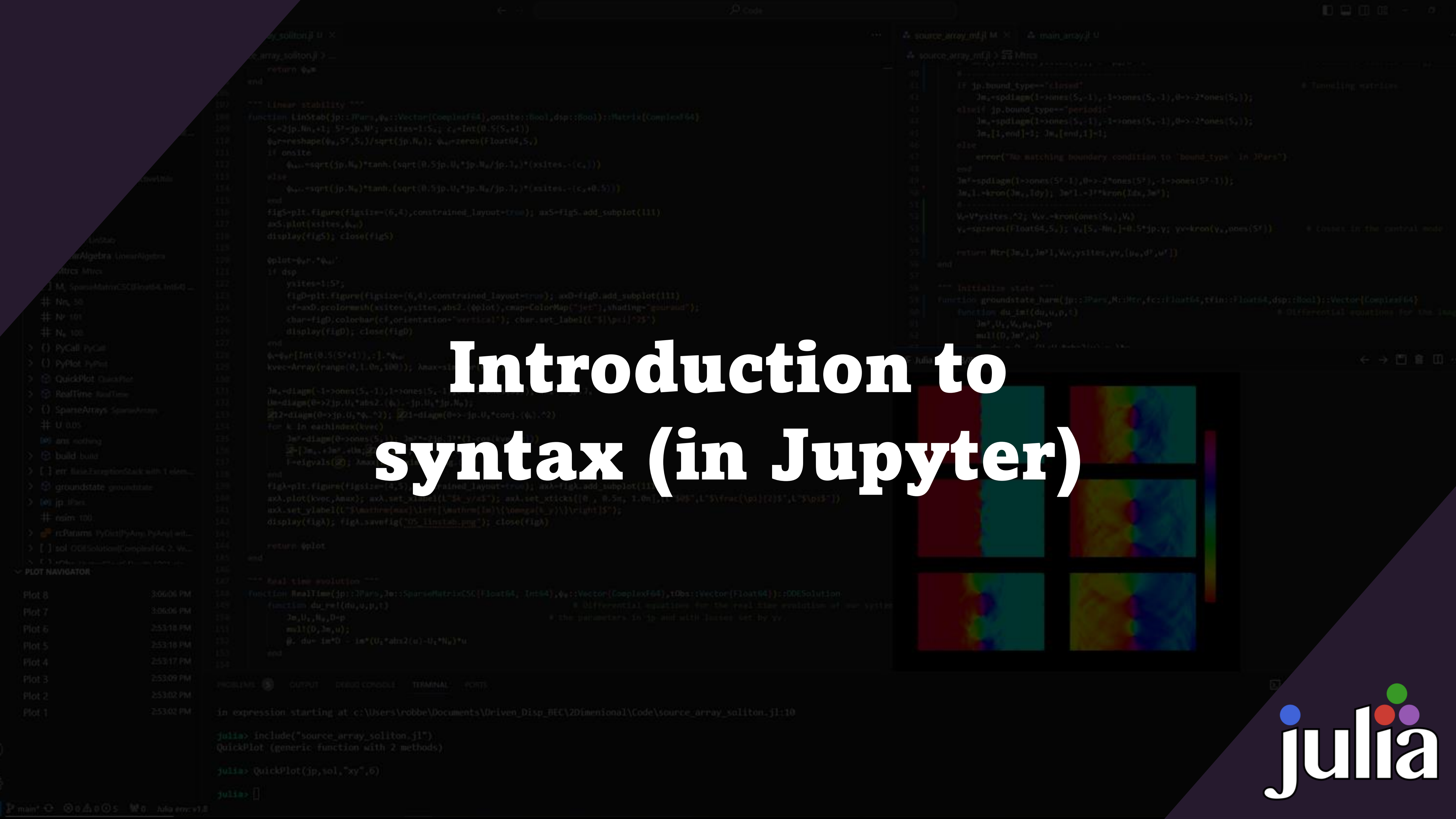
Jeffrey M. Perkel

Source: J.M. Perkel, *Nature* **572**, 141-142 (2019)

# Key properties

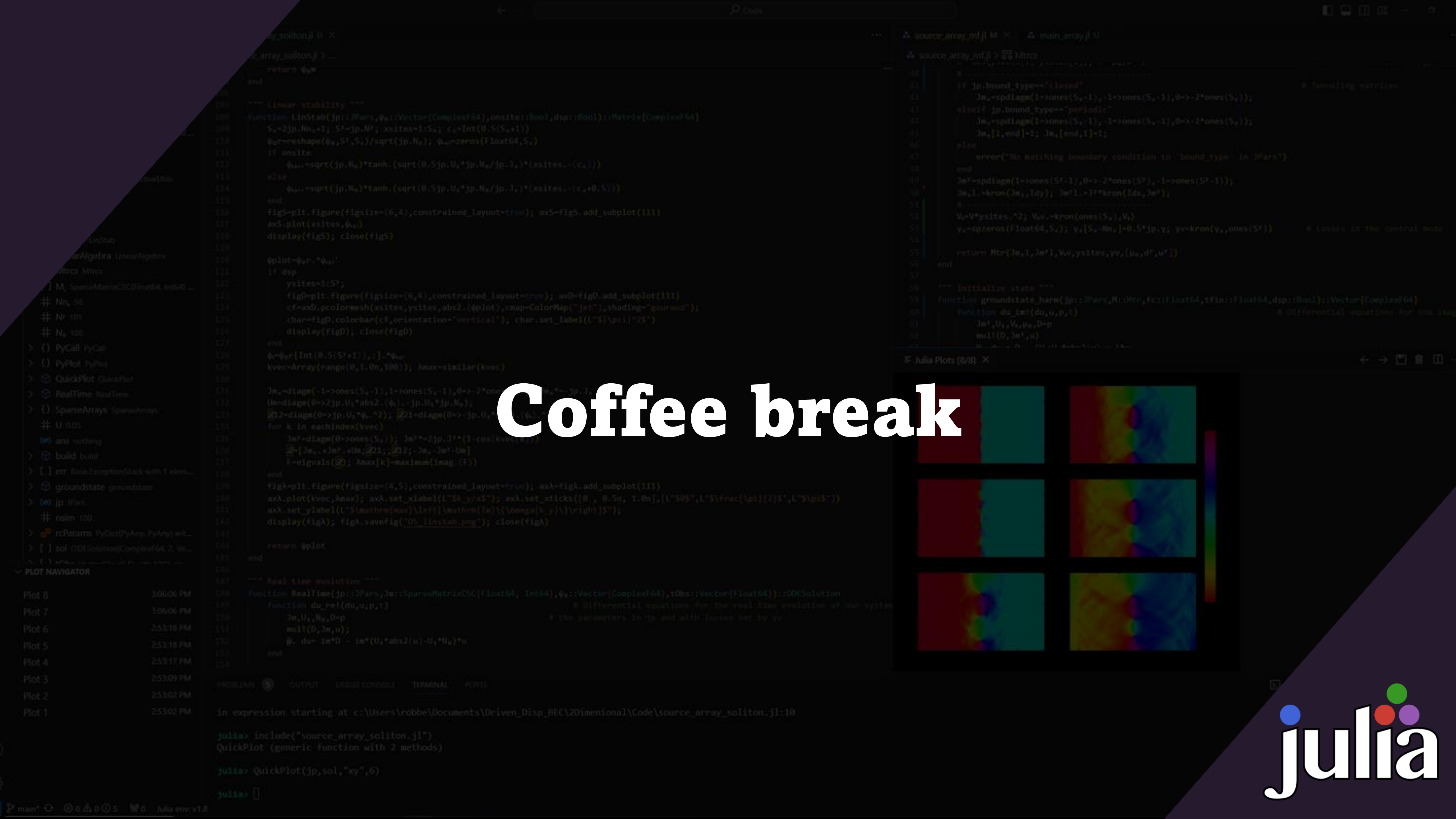


https://julialang.org/benchmarks/

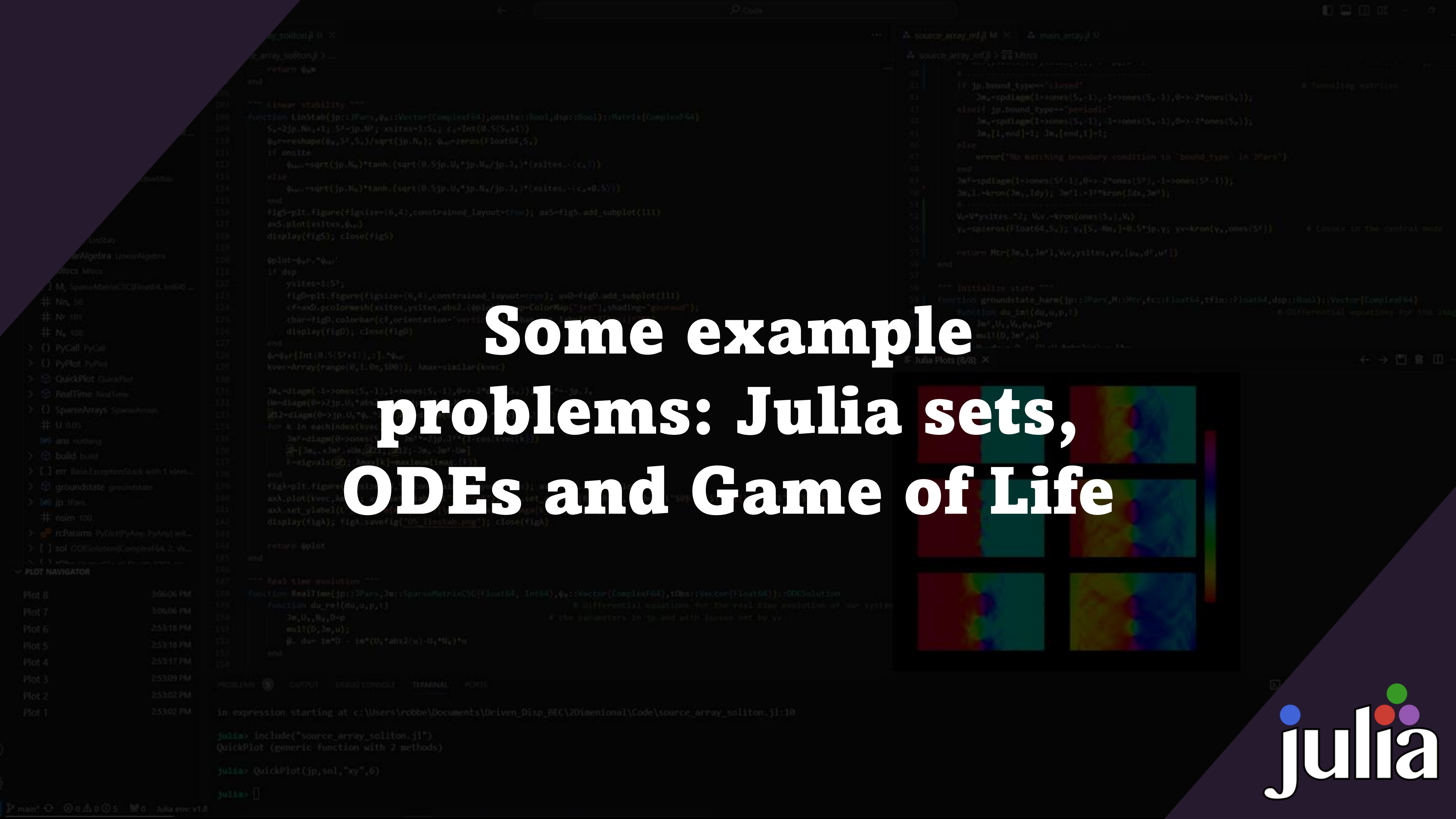


# Introduction to syntax (in Jupyter)

# Coffee break



```
in expression starting at c:\Users\robbe\Documents\Driven_Disp_BEC\2Dimenional\Code\source_array_soliton.jl:10

julia> include("source_array_soliton.jl")
QuickPlot (generic function with 2 methods)

julia> QuickPlot(jp,sol,"xy",6)

julia> 
```



# Some example problems: Julia sets, ODEs and Game of Life

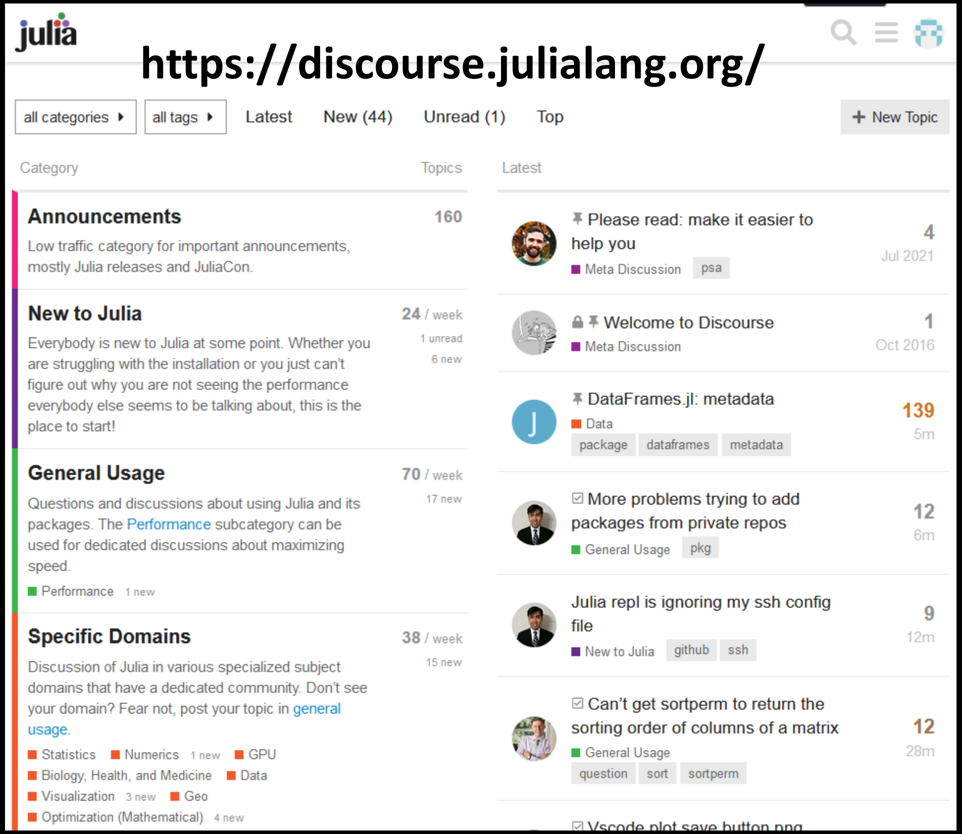
# Community today: Who uses Julia?

```
in expression starting at c:\Users\robbe\Documents\Driven_Disp_BEC\2Dimenional\Code\source_array_soliton.jl:10

julia> include("source_array_soliton.jl")
QuickPlot (generic function with 2 methods)

julia> QuickPlot(jp,sol,"xy",6)

julia> 
```

# julia community

**Some stats[1]:**

- ❑ 50 million+ downloads
- ❑ Over 10.000 packages
- ❑ Annual growth of 30%
- ❑ Active community
- ❑ Conferences globally



https://discourse.julialang.org/

| Category | Topics |
|---|---|
| **Announcements**<br>Low traffic category for important announcements, mostly Julia releases and JuliaCon. | 160 |
| **New to Julia**<br>Everybody is new to Julia at some point. Whether you are struggling with the installation or you just can't figure out why you are not seeing the performance everybody else seems to be talking about, this is the place to start! | 24 / week<br>1 unread<br>6 new |
| **General Usage**<br>Questions and discussions about using Julia and its packages. The Performance subcategory can be used for dedicated discussions about maximizing speed.<br>■ Performance 1 new | 70 / week<br>17 new |
| **Specific Domains**<br>Discussion of Julia in various specialized subject domains that have a dedicated community. Don't see your domain? Fear not, post your topic in general usage.<br>■ Statistics ■ Numerics 1 new ■ GPU<br>■ Biology, Health, and Medicine ■ Data<br>■ Visualization 3 new ■ Geo<br>■ Optimization (Mathematical) 4 new | 38 / week<br>15 new |

| Latest | | |
|---|---|---|
| Please read: make it easier to help you<br>■ Meta Discussion psa | 4 | Jul 2021 |
| Welcome to Discourse<br>■ Meta Discussion | 1 | Oct 2016 |
| DataFrames.jl: metadata<br>■ Data<br>package dataframes metadata | 139 | 5m |
| ☑ More problems trying to add packages from private repos<br>■ General Usage pkg | 12 | 6m |
| Julia repl is ignoring my ssh config file<br>■ New to Julia github ssh | 9 | 12m |
| ☑ Can't get sortperm to return the sorting order of columns of a matrix<br>■ General Usage<br>question sort sortperm | 12 | 28m |
| ☑ Vscode plot save button png | | |

juliacon
Local Eindhoven 2023

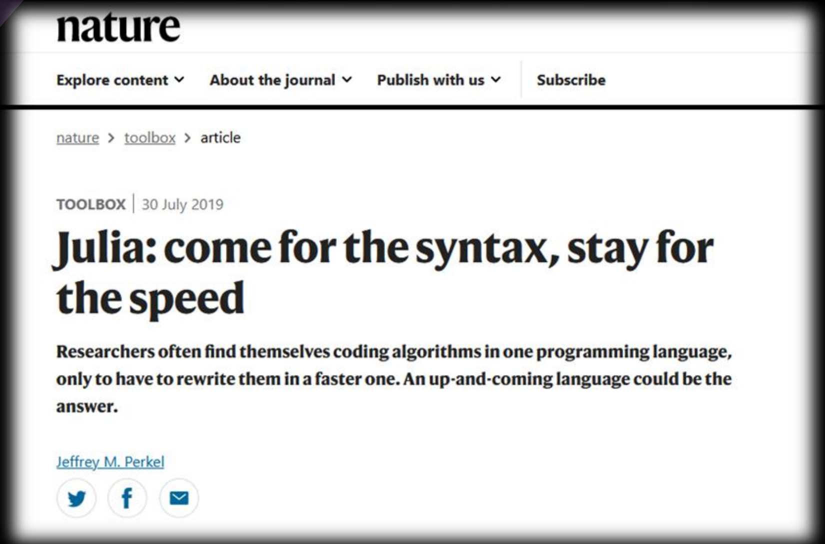
juliacon

julia

[1] https://juliahub.com/

# Job opportunities?

## Academia



nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature > toolbox > article

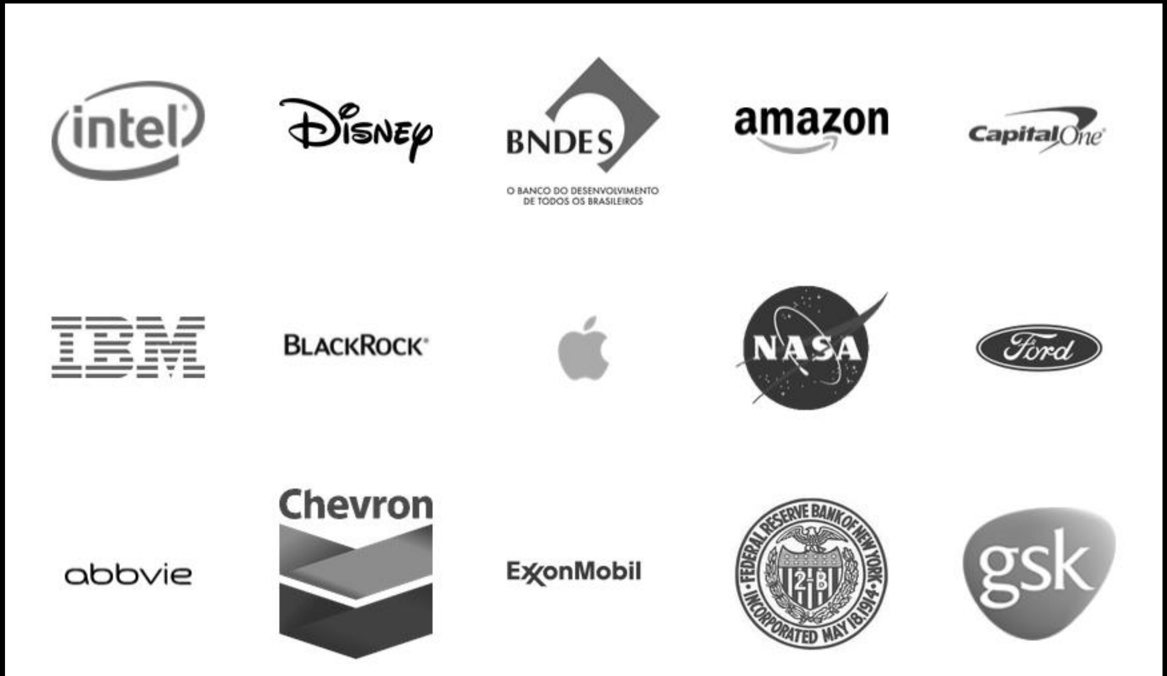
TOOLBOX | 30 July 2019

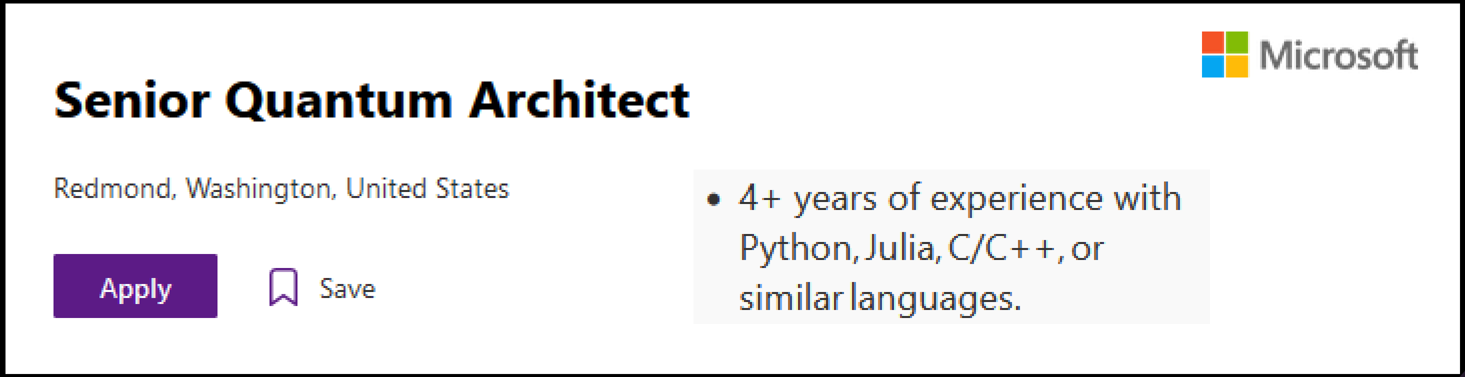
### Julia: come for the syntax, stay for the speed

Researchers often find themselves coding algorithms in one programming language, only to have to rewrite them in a faster one. An up-and-coming language could be the answer.

Jeffrey M. Perkel

## Private sector


intel
Disney
BNDES
O BANCO DO DESENVOLVIMENTO DE TODOS OS BRASILEIROS
amazon
CapitalOne
IBM
BLACKROCK
NASA
Ford
abbvie
Chevron
ExxonMobil
FEDERAL RESERVE BANK OF NEW YORK INCORPORATED MAY 18 1914
2-B
gsk



Microsoft

### Senior Quantum Architect

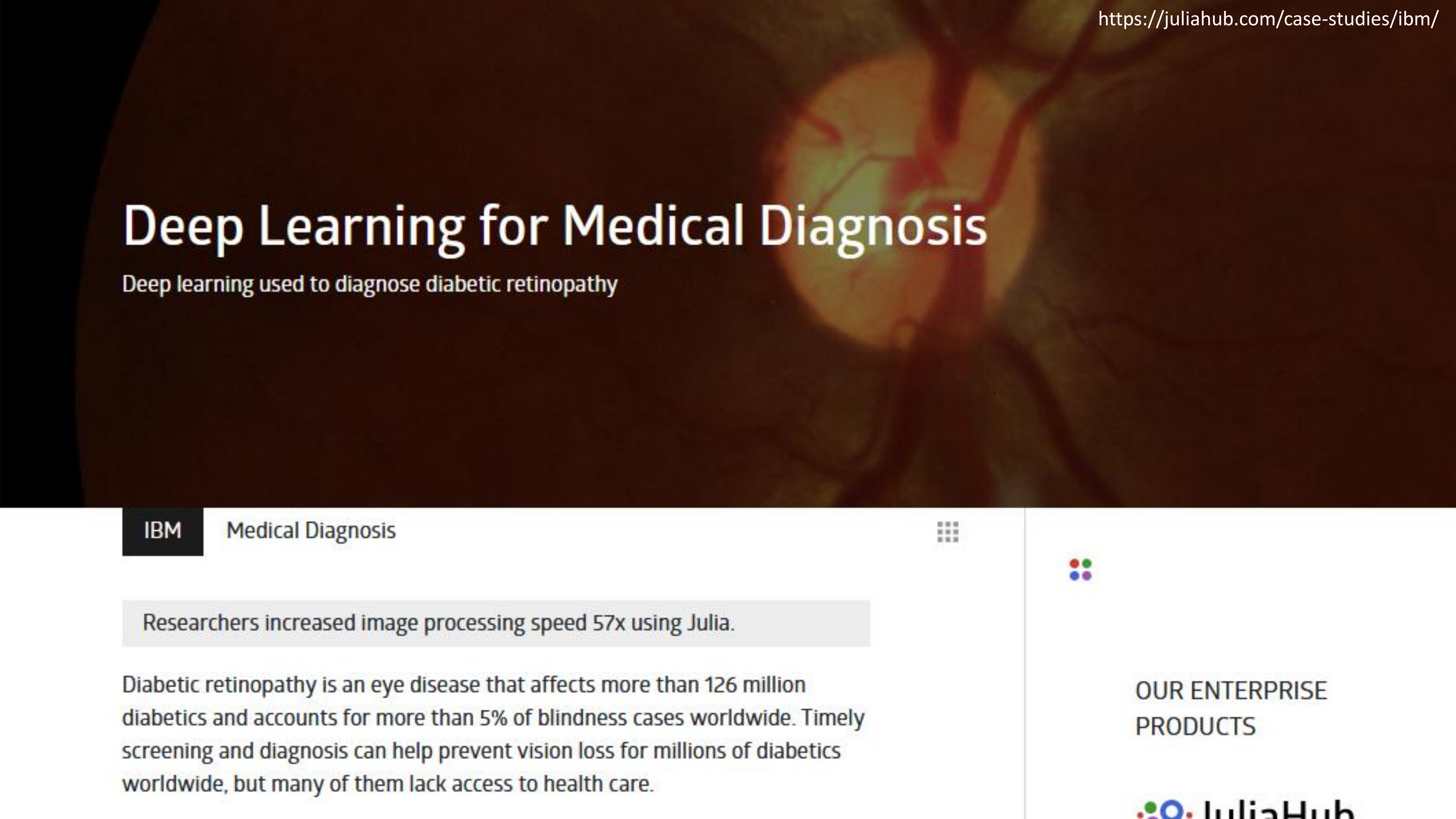
Redmond, Washington, United States

Apply    Save

- 4+ years of experience with Python, Julia, C/C++, or similar languages.

https://juliahub.com/

https://juliahub.com/case-studies/ibm/



# Deep Learning for Medical Diagnosis

Deep learning used to diagnose diabetic retinopathy

IBM

Medical Diagnosis

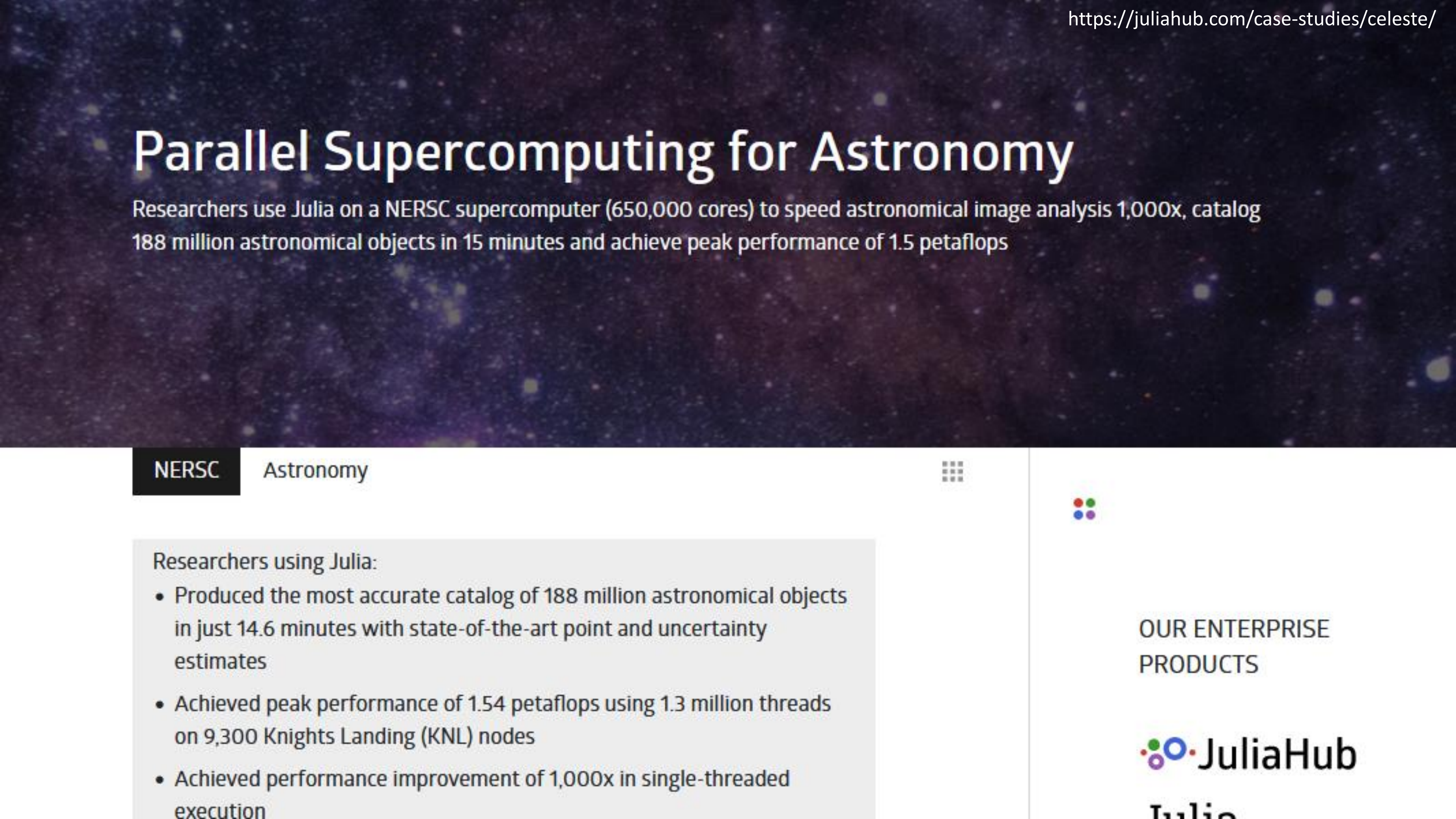
Researchers increased image processing speed 57x using Julia.

Diabetic retinopathy is an eye disease that affects more than 126 million diabetics and accounts for more than 5% of blindness cases worldwide. Timely screening and diagnosis can help prevent vision loss for millions of diabetics worldwide, but many of them lack access to health care.

OUR ENTERPRISE PRODUCTS

JuliaHub

https://juliahub.com/case-studies/celeste/



# Parallel Supercomputing for Astronomy

Researchers use Julia on a NERSC supercomputer (650,000 cores) to speed astronomical image analysis 1,000x, catalog 188 million astronomical objects in 15 minutes and achieve peak performance of 1.5 petaflops

NERSC Astronomy

Researchers using Julia:

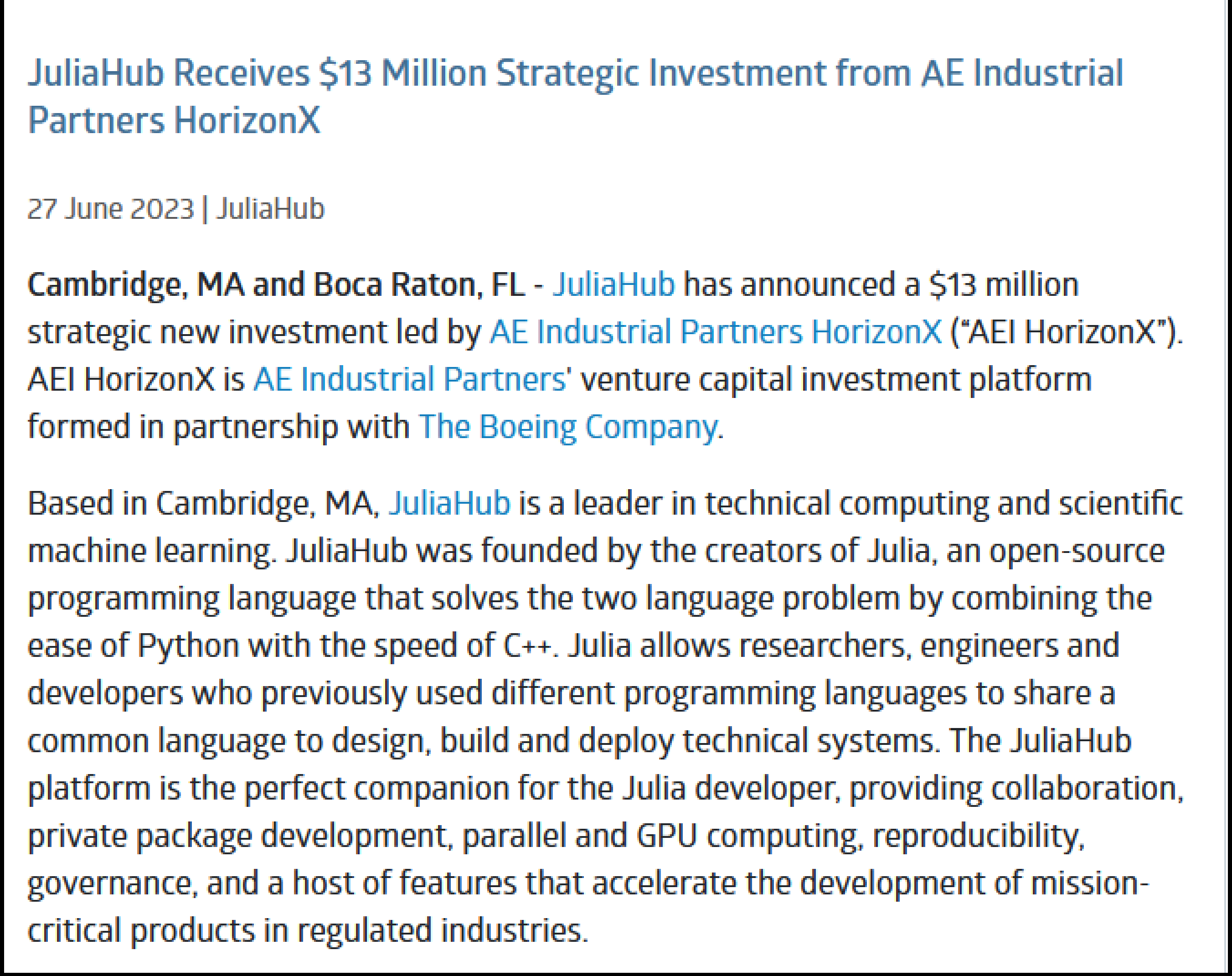
- Produced the most accurate catalog of 188 million astronomical objects in just 14.6 minutes with state-of-the-art point and uncertainty estimates
- Achieved peak performance of 1.54 petaflops using 1.3 million threads on 9,300 Knights Landing (KNL) nodes
- Achieved performance improvement of 1,000x in single-threaded execution

OUR ENTERPRISE PRODUCTS

JuliaHub

Julia

# Job opportunities?



## JuliaHub Receives $13 Million Strategic Investment from AE Industrial Partners HorizonX

27 June 2023 | JuliaHub

**Cambridge, MA and Boca Raton, FL** - JuliaHub has announced a $13 million strategic new investment led by AE Industrial Partners HorizonX ("AEI HorizonX"). AEI HorizonX is AE Industrial Partners' venture capital investment platform formed in partnership with The Boeing Company.

Based in Cambridge, MA, JuliaHub is a leader in technical computing and scientific machine learning. JuliaHub was founded by the creators of Julia, an open-source programming language that solves the two language problem by combining the ease of Python with the speed of C++. Julia allows researchers, engineers and developers who previously used different programming languages to share a common language to design, build and deploy technical systems. The JuliaHub platform is the perfect companion for the Julia developer, providing collaboration, private package development, parallel and GPU computing, reproducibility, governance, and a host of features that accelerate the development of mission-critical products in regulated industries.

https://juliahub.com/

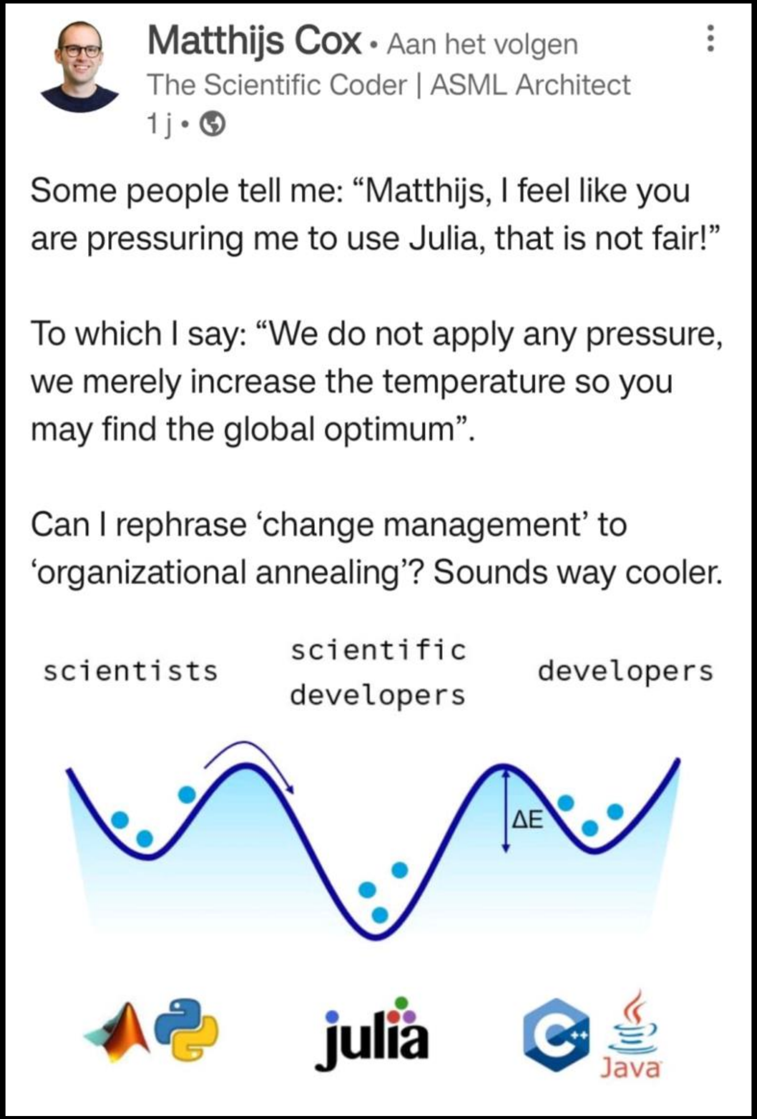
# References:

[1] J. Bezanson, S Karpinski, V.B. Shah, and A. Edelman, *Why we created Julia*, The Julia Language Blog, https://julialang.org/blog/2012/02/why-we-created-julia/ (2012).

[2] A. Buscemi, *A comparative study of code generation using ChatGPT 3.5 across 10 programming languages*, arXiv:2308.04477v1 (2023).

[3] M. Cox, *How to solve the two language problem?*, The Scientific coder, https://scientificcoder.com/how-to-solve-the-two-language-problem (2023).

[4] J.M. Perkel, *Julia: come for the syntax, stay for the speed*, Nature 572, 141-142 (2019).



Matthijs Cox • Aan het volgen

The Scientific Coder | ASML Architect

1 j •

Some people tell me: “Matthijs, I feel like you are pressuring me to use Julia, that is not fair!”

To which I say: “We do not apply any pressure, we merely increase the temperature so you may find the global optimum”.

Can I rephrase ‘change management’ to ‘organizational annealing’? Sounds way cooler.