

Key features Overview



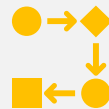
Syntax



Speed



Package

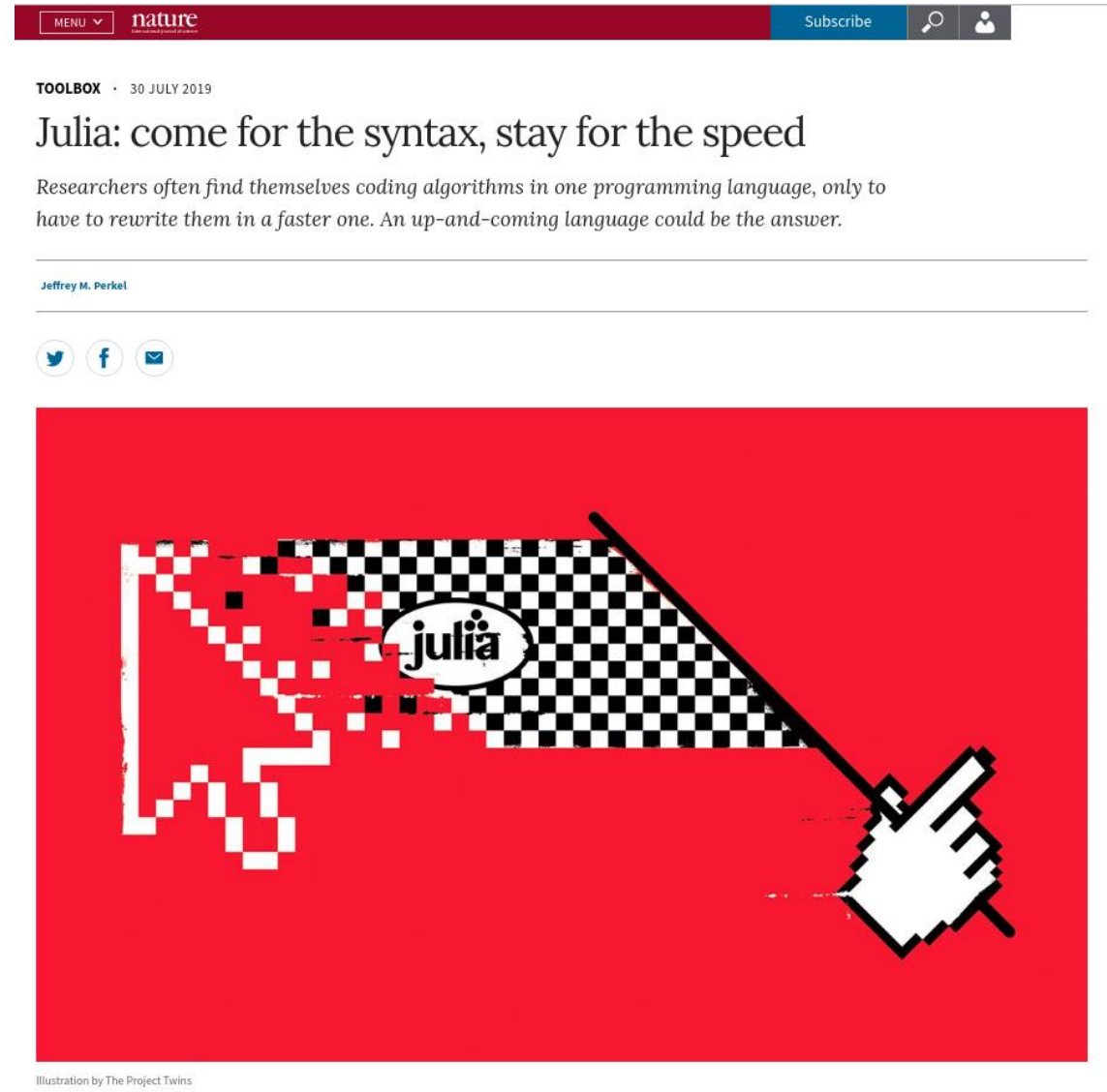


Language binding

Julia Syntax

Article (30 July 2019) in Nature journal about Julia:

"Launched in 2012, Julia is an open-source language that combines the interactivity and syntax of 'scripting' languages, such as Python, Matlab and R, with the speed of 'compiled' languages such as Fortran and C."(Jeffrey M. Perkel)



Jeffrey M. Perkel, "Julia: come for the syntax, stay for the speed", Nature. 2019 Aug;572(7767):141-142. doi: 10.1038/d41586-019-02310-3.

Julia Syntax



Programming language comparison resources:

- [Noteworthy Differences from other Languages](#)
- [Python vs R vs Julia for Data Sciences](#)
- [Julia Vs R Comparison Cheat Sheet](#)
- [MATLAB–Python–Julia cheatsheet](#)
- [The Matrix Cheatsheet Matlab–Python–R–Julia comparison](#)



Libraries/Packages

To add a library/package:

```
install.packages("tidyverse")
```

```
conda install pandas  
pip install pandas
```

```
Pkg.add("DataFrames")
```

To load a library/package:

```
library(tidyverse)
```

```
import numpy as np  
import pandas as pd
```

```
using DataFrames, CSV, LinearAlgebra using Statistics,  
StatsBase
```

Control Structures

if / else if /else

```
if (x == a) {  
  ...  
} else if (x == b) {  
  ...  
} else {  
  ...  
}
```

```
if x == a:  
  ...  
elif x == b:  
  ...  
else:  
  ...
```

```
if x == a  
  ...  
elseif x == b  
  ...  
else  
  ...  
end
```

for loop

```
for (i in I) {  
  ...  
}
```

```
for i in I:  
  ...
```

```
for i in I  
  ...  
end
```



Matrix

To create a matrix $A_{2 \times 3}$

```
A <- matrix(c(1,2,3,4,5,6), 2, 3)
```

```
A = np.array([[1,2,3],[4,5,6]])
```

```
A = [1 2 3; 4 5 6]
```

2 x 2 matrix of zeros

```
A <- zeros(2,2)
```

```
A = np.zeros(2,2)
```

```
A = zeros(2,2)
```

2 x 2 identity matrix

```
A <- eye(2,2)
```

```
A = np.eye(2)
```

```
A = I
```

Select an element in matrix A

```
A[i,j]
```

```
A[i,j]
```

```
A[i,j]
```

Select a row in matrix A

```
A[i,]
```

```
A[i,:]
```

```
A[i,:]
```

Select a column in matrix A

```
A[:,j]
```

```
A[:,j]
```

```
A[:,j]
```



Function

To declare

```
f <- function(x1, x2 = 3) {  
  y <- x2 - x1  
  y  
}
```

```
def f(x1, x2 = 3):  
    y = x2 - x1  
    return y
```

```
function f(x1, x2 = 3)  
    y = x2 - x1  
end  
  
# math notation  
f(x1, x2) = x2 - x1
```

To call

```
f(5,12)      # position matters  
  
f(x2=12, x1=5) # order free
```

```
f(5,12)      # position matters  
  
f(x2=12, x1=5) # order free
```

```
f(5,12)      # position matters  
  
f(x2=12, x1=5) # if named arguments
```

To pass multiple arguments or keyword arguments to a function

```
f <- function(x1, x2 = 7, ...) {  
  y <- func_foo(x1, x2, ...)  
  y  
}
```

```
def f(x1, x2 = 7, *args, **kwargs):  
    y = func_foo(x1, x2, args, kwargs)  
    return y
```

```
function f(x1, x2 = 7, args...;kwargs...)  
    y = func_foo(x1,x2,args;kwargs)  
end
```

Numeric Types

Julia supports a range of numeric types, including integer, floating-point, and complex numbers. Here's a list of the most common numeric types:

- Int8, Int16, Int32, **Int64**, Int128: signed integers of 8, 16, 32, 64, and 128 bits, respectively
- UInt8, UInt16, UInt32, UInt64, UInt128: unsigned integers of 8, 16, 32, 64, and 128 bits, respectively
- Float16, Float32, **Float64**: floating-point numbers of 16, 32, and 64 bits, respectively
- ComplexF32, ComplexF64: complex numbers of 32 and 64 bits, respectively

String, Character, and Boolean Type

- The String type in Julia represents a sequence of Unicode characters. Here's an example of how to define and use a String type:

```
julia> my_string = "Hello World!"
```

- The Char type in Julia represents a single Unicode character. Here's an example of how to define and use a Char type:

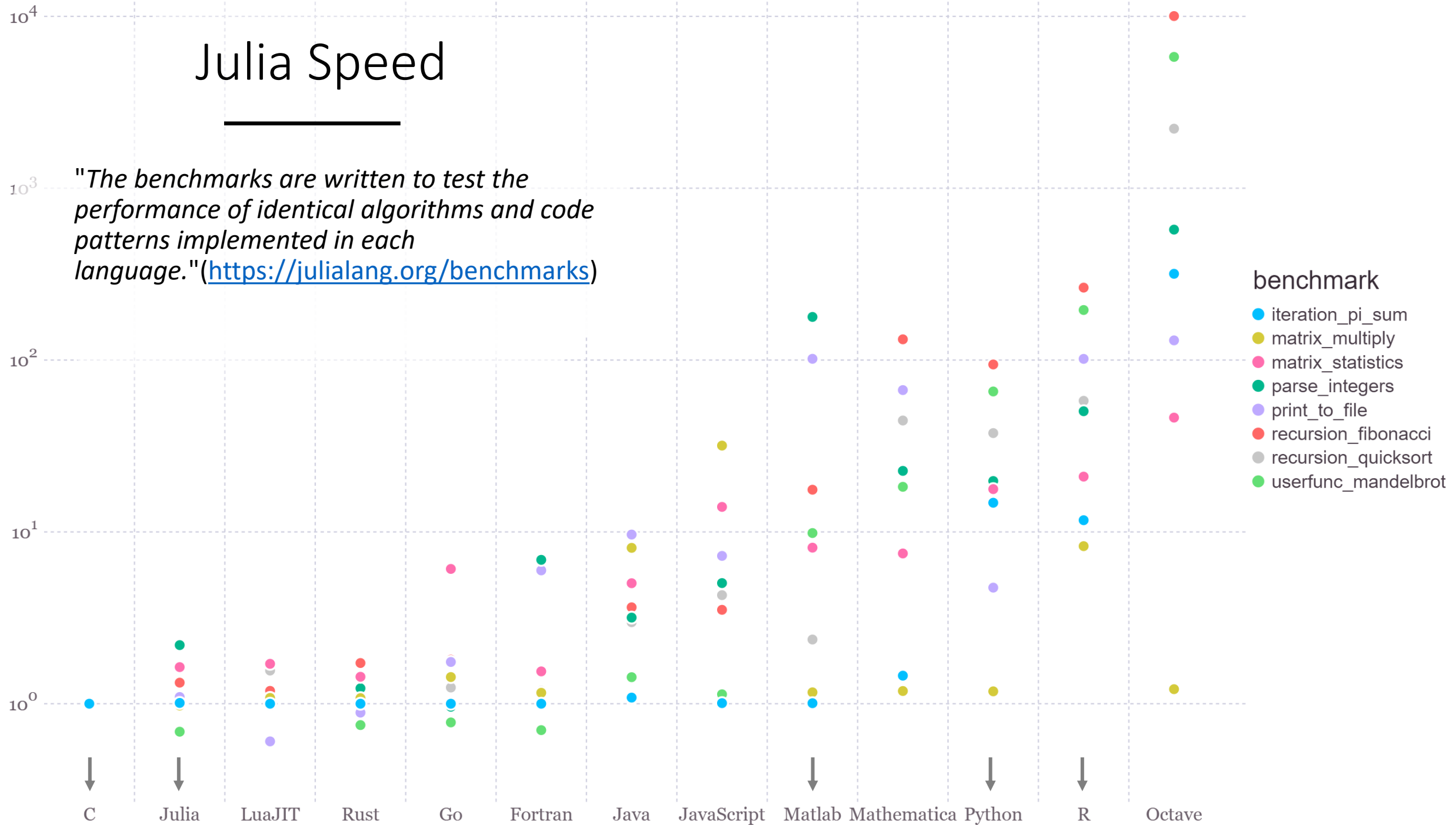
```
julia> my_char = 'Hello World!'
```

- The Boolean type in Julia represents logical values, which can be either **true** or **false**. Here's an example of how to define and use a Boolean type:

```
julia> isvalid = true
```


Julia Speed

"The benchmarks are written to test the performance of identical algorithms and code patterns implemented in each language." (<https://julialang.org/benchmarks>)



Programming languages :

- Julia 1.8.5
- Python 3.11.1
- C gcc (Ubuntu 12.2.0-3ubuntu1) 12.2.0
- Lisp SBCL 2.3.0
- Java openjdk 19 2023-02-14

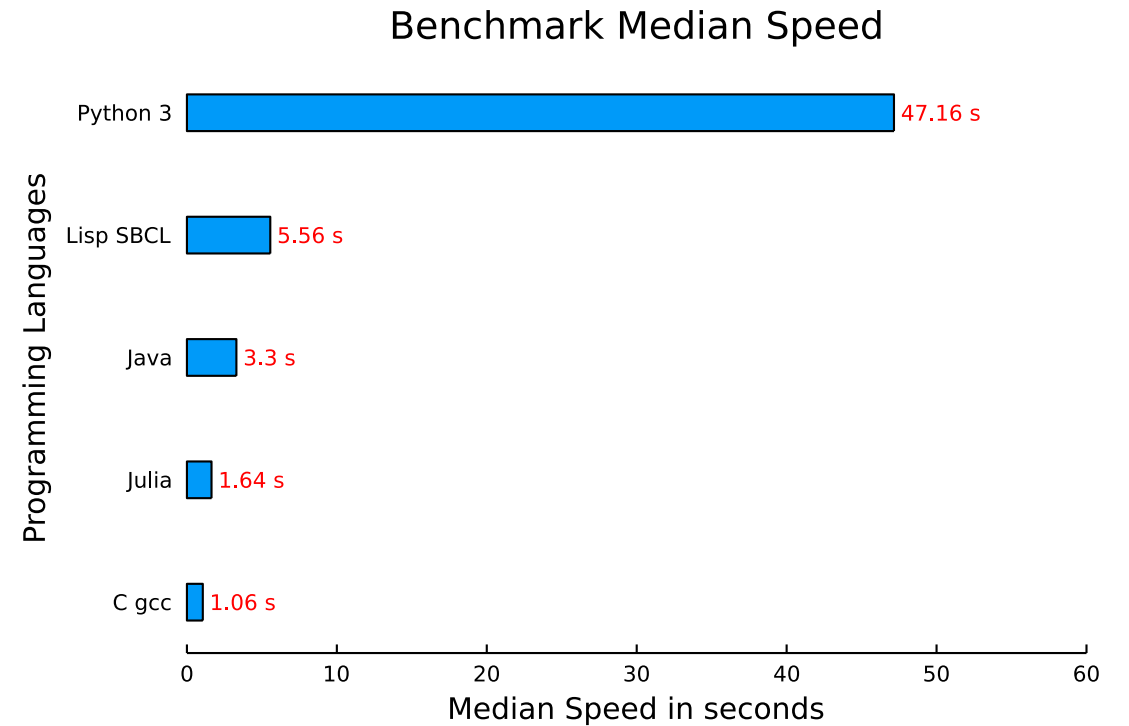
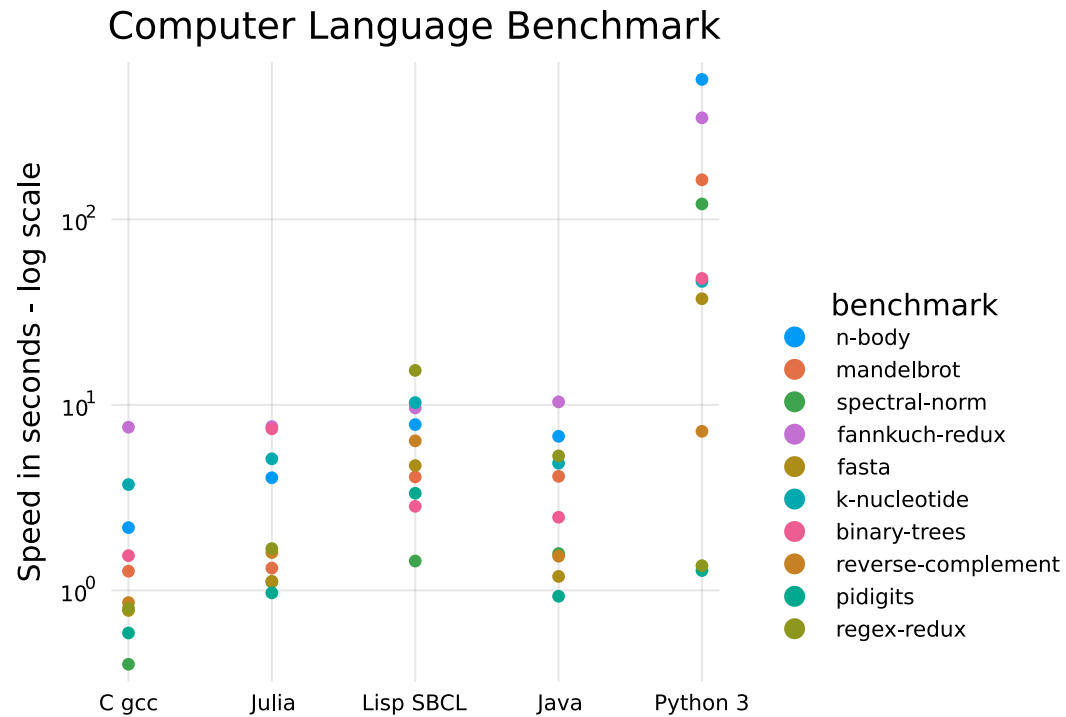
CPU: quad-core 3.0GHz Intel® i5-3330®

RAM: 15.8 GiB

HDD: 2TB SATA disk drive

OS: Ubuntu™ 22.10 x86_64 GNU/Linux 5.19.0-29-generic

The Computer Language Benchmarks Games

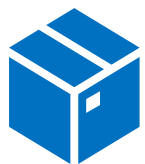


Julia Packages

Julia has over 9114 registered packages. They are a substantial part of the Julia ecosystem.

To explore existing packages related to your area of interest you can check out the following link: <https://juliahub.com/ui/Packages>

Julia comes with a built-in package manager named `Pkg`, and also an interactive Package Mode: [Example](#)



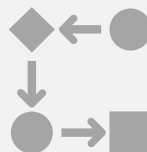
Julia Packages Worth Mentioning

- **Statistics and Math:** **StatsBase** (Basic functionalities for statistics), **LinearAlgebra** (Linear algebra operations), **Distributions** (Probability distributions), **GLM** (Generalized linear models), **Turing** (Bayesian Inference)
- **Data tools:** **DataFrames** (Essential tools for tabular data), **Queryverse** (A meta package for data science), **CSV** (For working with CSV files), **XLSX** (Excel file reader and writer), **Tidier** ([100% Julia implementation of the R tidyverse mini-language](#))
- **Biology:** **Bio** (Framework for computational biology and bioinformatics)
- **Machine Learning:** **Flux** (ML library), **TensorFlow** (A wrapper around TensorFlow), **Knet** (Deep learning framework), **ScikitLearn** (Scikit-learn framework), **MLJ** (A Machine Learning Framework for Julia)
- **Plot:** **Plots** (Julia visualizations and data analysis), **PyPlot** (matplotlib plotting like), **Gadfly** (ggplot2 plotting like), **UnicodePlots** (Unicode-based scientific plotting for working in the terminal), **Makie** (data visualization ecosystem)
- **Language binding:** **PyCall** (For Python calling), **RCall** (For R calling)
- **Notebooks:** **IJulia** (Jupyter Julia kernel), **Pluto** ([Interactive Julia native notebooks](#))

Language Binding

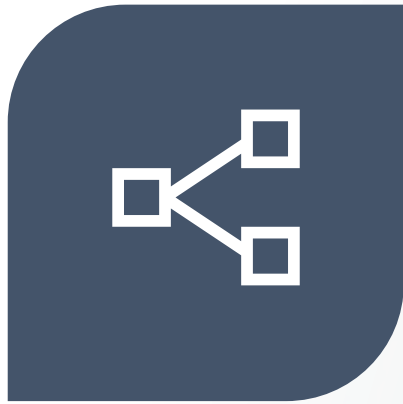


Even though, the package ecosystem still has room to grow. Julia has excellent foreign function interfaces. Easily call into other languages such as python with `PyCall` or R with `Rcall`.



This means that you don't have to wait until the Julia ecosystem is fully mature, and that you don't have to give up your favorite package/library from another language when moving to Julia!

Notebooks



GETTING STARTED



MACROS



DATA STRUCTURES