Introduction to Julia

Alessandro Conigli

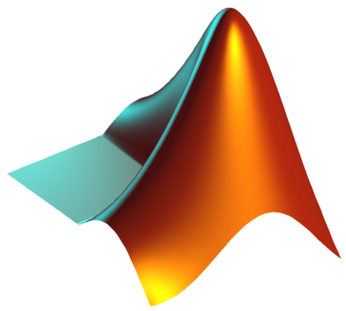
IFT UAM-CSIC

December 13th -17th Tor Vergata

Nowadays there's a plethora of programming languages

Static - Compiled - Speed

Dynamic - Interpreted - Convenience



The two language problem

- The two language problem is a trade-off that developers typically make when choosing a language: it can either be relatively easy for computer to run, or relatively easy for humans to write, but not both

Prototype

User friendly

Production

Developers

What does physics need from a programming language?

Easy to write and read!

Fast and scalable!

Interactive!

Dynamic

Compiled

User types and standard types

Standalone or glue

Introduction

- Julia is a high-level, high-performance and dynamic programming language
- Adequate for data science, research, scientific computer, among others

First stable release 0.4.6
June 2016

Current release 1.6
Winter 2021

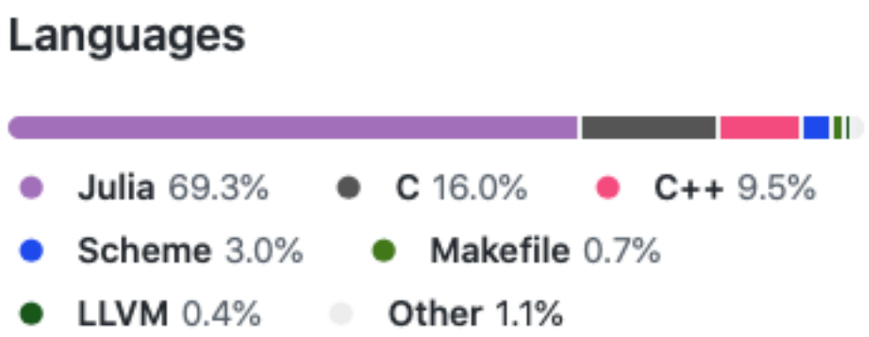
Julia: A fresh approach to numerical computing

Jeff Bezanson Alan Edelman Stefan Karpinski Viral B. Shah

MIT and Julia Computing*
July 7, 2015

Abstract

Bridging cultures that have often been distant, Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing. Julia is designed to be easy and fast. Julia questions notions generally held as "laws of nature" by practitioners of numerical computing:



Julia in a Nutshell

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

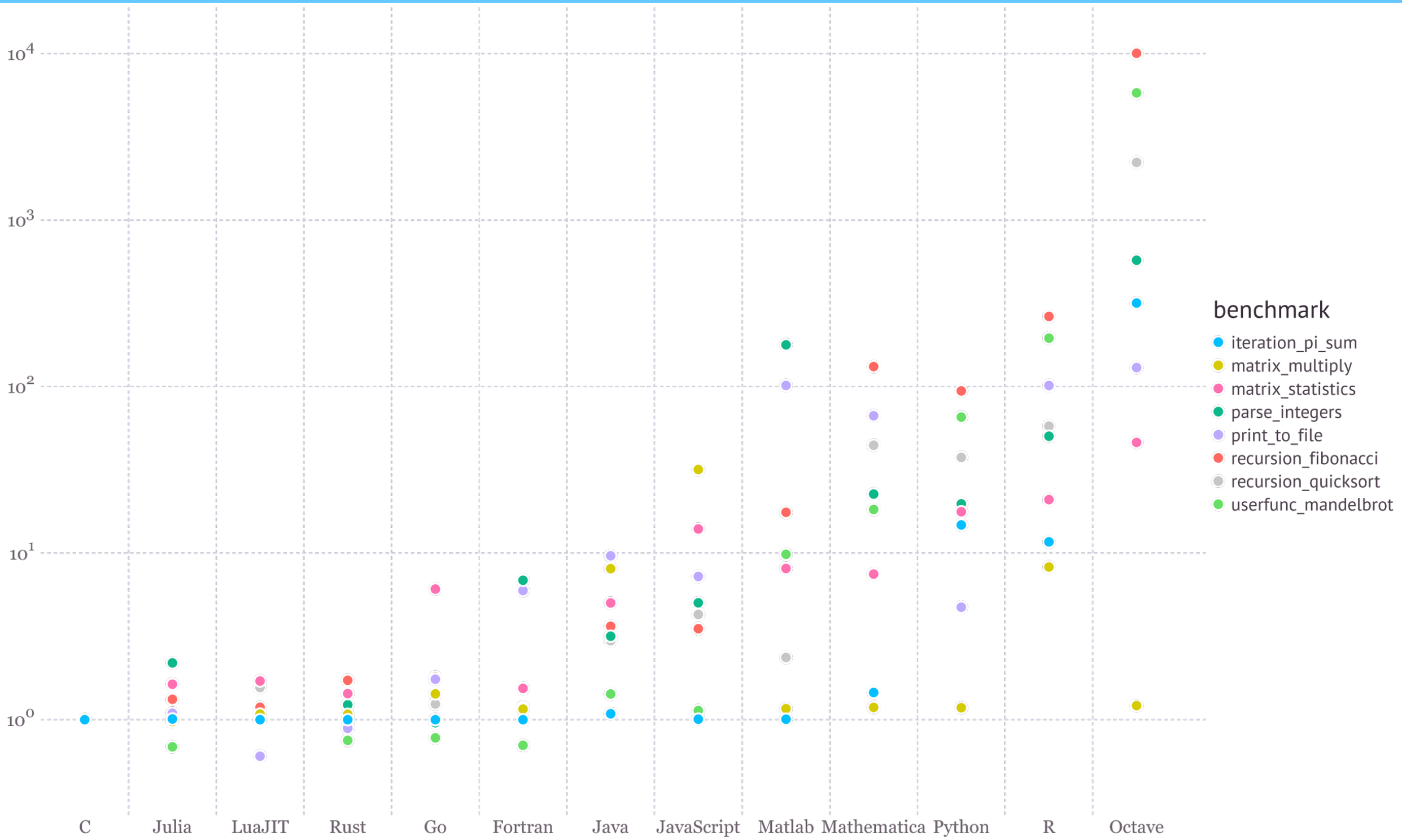
Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

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.

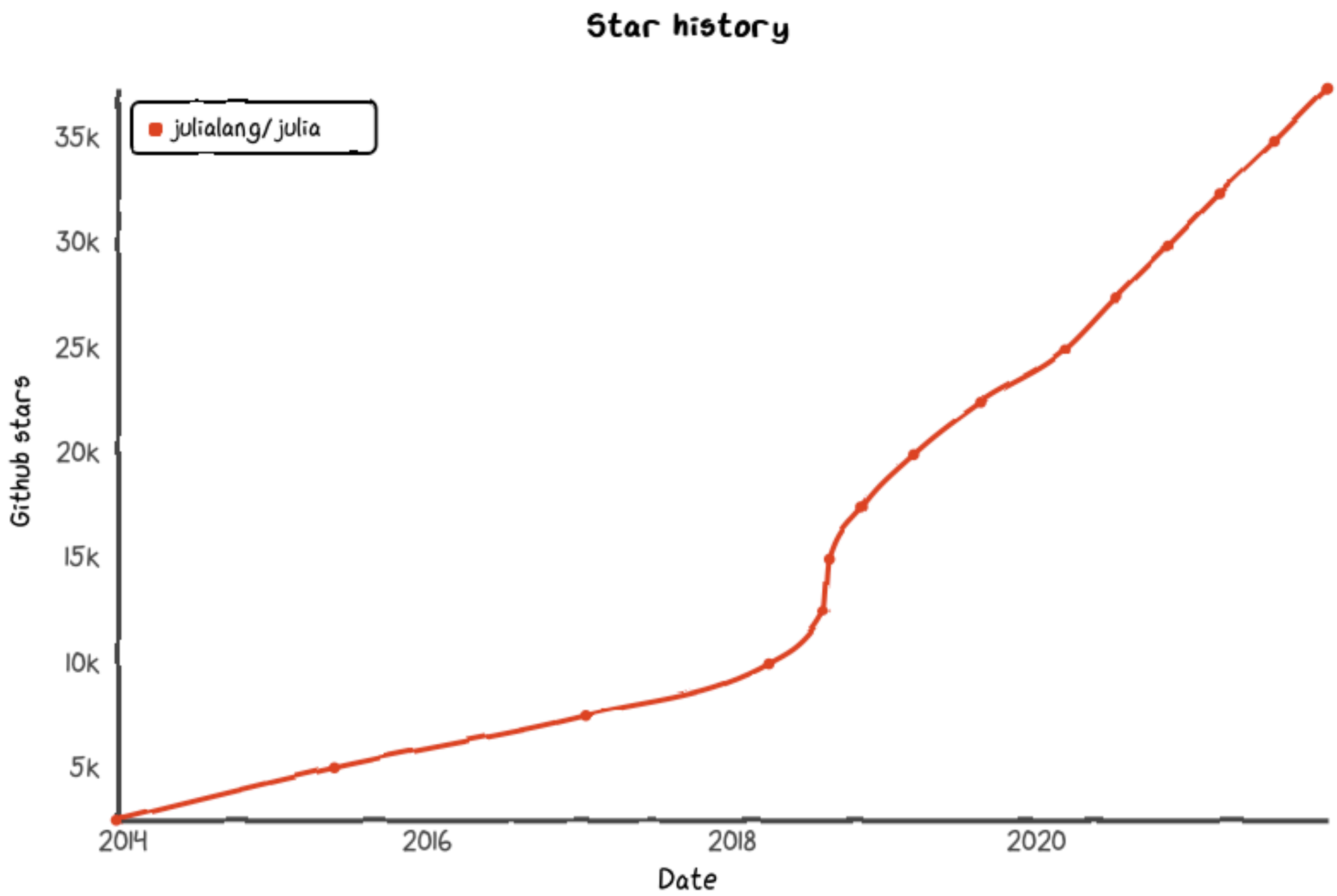
[https://julialang.org]

Benchmark



[https://julialang.org/benchmarks]

GitHub stars



How to install Julia

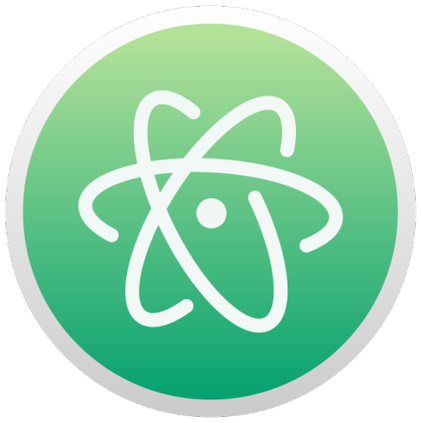
- Install Julia version 1.6.4 (latest stable version) click here to download

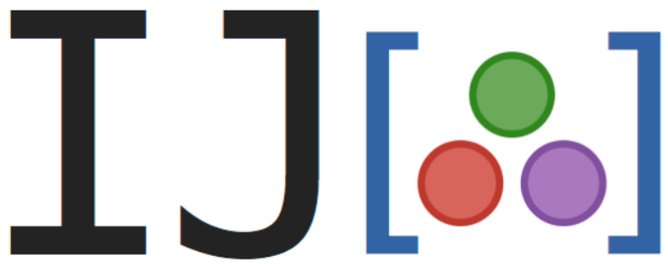
Windows [help]	64-bit (installer), 64-bit (portable)		32-bit (installer), 32-bit (portable)
macOS [help]	64-bit		
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl)[1] (GPG)		32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)		32-bit (ARMv7-a hard float) (GPG)
Generic FreeBSD on x86 [help]	64-bit (GPG)		
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

- Finally, create a symbolic link to Julia inside the /usr/local/bin folder

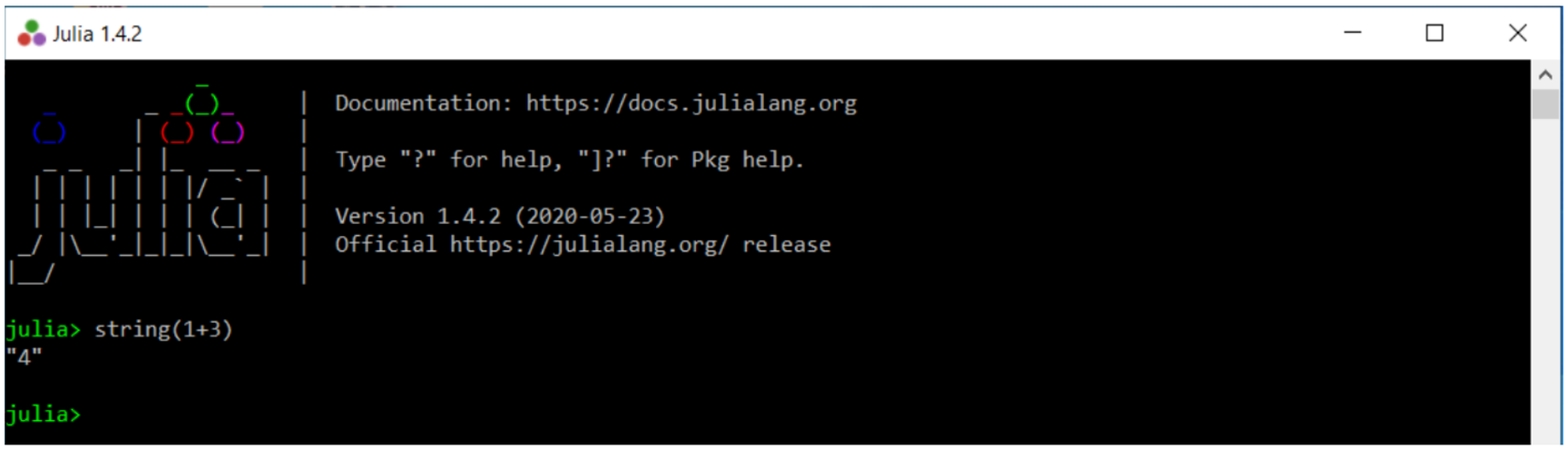
```
sudo ln -s /opt/julia-1.6.3/bin/julia /usr/local/bin/julia
```

Integrated development environments





Visual Studio Code



Built-in types

Integer

Int8

Int16

Int32

Int64

Int128

Unsigned

Uint8

Uint16

Uint32

Uint64

Uint128

Float

Float16

Float32

Float64

Other

String

Char

Complex

Bool

```
julia> struct Foo
           bar
           baz::Int
           qux::Float64
       end
```

Functions

```
function _drop(column::Symbol, labels::Array{Symbol}, columns::Array,
        coldata::Array)
        pos = findall(x->x==column, labels)[1]
        deleteat!(labels, pos)
        deleteat!(coldata, pos)
        deleteat!(columns, pos)
end

function _drop(row::Int64, columns::Array)
        [deleteat!(col, row) for col in columns]
end

function _drop(row::Array, columns::Array)
        [deleteat!(col, row) for col in columns]
end
```

Figure from townrdsdatascience

Package Manager

```
julia> import Pkg

julia> Pkg.add("Example");
    Updating registry at `~/.julia/registries/General`
    Updating git-repo `https://github.com/JuliaRegistries/General.git`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`

julia> using Example
```

```
(@v1.6) pkg> add Example;
    Updating registry at `~/.julia/registries/General`
    Updating git-repo `https://github.com/JuliaRegistries/General.git`
   Resolving package versions...
    Updating `~/.julia/environments/v1.6/Project.toml`
  [7876af07] + Example v0.5.3
    Updating `~/.julia/environments/v1.6/Manifest.toml`
  [7876af07] + Example v0.5.3

(@v1.6) pkg> rm Example
    Updating `~/.julia/environments/v1.6/Project.toml`
  [7876af07] - Example v0.5.3
    Updating `~/.julia/environments/v1.6/Manifest.toml`
  [7876af07] - Example v0.5.3
```

Let’s get some practice!