Wikipedia

Julia (programming language)

Julia is a high-level, high-performance, dynamic programming language. While it is a general purpose language and can be used to write any application, many of its features are well-suited for numerical analysis and computational science.[15][16][17][18]

Distinctive aspects of Julia's design include a type system with parametric polymorphism in a dynamic programming language; with multiple dispatch as its core programming paradigm. Julia supports concurrent, (composable) parallel and distributed computing (with or without using MPI[19] and/or the built-in corresponding[20] to "OpenMP-style" threads[21]), and direct calling of C and Fortran libraries without glue code. Julia uses a just-in-time (JIT) compiler that is referred to as "just-ahead-of-time" (JAOT) in the Julia community, as Julia compiles (on default settings) to machine code before running it.[22][23]

Julia is garbage-collected,[24] uses eager evaluation, and includes efficient libraries for floating-point calculations, linear algebra, random number generation, and regular expression matching. Many libraries are available, including some (e.g., for fast Fourier transforms) that were previously bundled with Julia and are now separate.[25]

Several development tools support coding in Julia, such as integrated development environments (e.g. Microsoft's Visual Studio Code, with extensions available adding Julia support to IDEs, e.g providing debugging and linting[26] support); with integrated tools, e.g. a profiler (and flame graph support available[27][28] for the built-in one), debugger,[29] and the Rebugger.jl package "supports repeated-execution debugging"[a] and more.[31]

Julia

Paradigm	Multi-paradigm: multiple dispatch (primary paradigm), procedural, functional, meta, multistaged[1]
Designed by	Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah
Developer	Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and other contributors[2][3]
First appeared	2012[4]
Stable release	1.5.2[5] / 24 September 2020
Preview release	1.6.0-DEV with daily updates (and 1.0.6[6] being *worked* on)
Typing discipline	Dynamic, nominative, parametric, optional
Implementation language	Julia, C, C++, Scheme, LLVM[7]
Platform	Tier 1: x86-64,

Contents

	IA-32, 64-bit ARM, CUDA/Nvidia GPUs Tier 2: Windows 32-bit (64-bit is tier 1) Tier 3: 32-bit ARM, PowerPC, AMD GPUs. Also has web browser support (for JavaScript and WebAssembly)[8] and works in Android. For more details see "supported platforms" (https://julialang.org/downloads/#currently_supported_platforms).
OS	Linux, macOS, Windows and FreeBSD
License	MIT (core),[2] GPL v2;[7][9] a makefile option omits GPL libraries[10]
Filename extensions	.jl
Website	JuliaLang.org (https://JuliaLang.org)

Influenced by

C[4] · Dylan[11] · Lisp[4] · Lua[12] · Mathematica[4] (strictly its Wolfram Language[4][13]) · MATLAB[4] · Perl[12] · Python[12] · R[4] · Ruby[12] · Scheme[14]

History

Work on Julia was started in 2009, by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman, who set out to create a free language that was both high-level and fast. On 14 February 2012, the team launched a website with a blog post explaining the language's mission.[32] In an interview with *InfoWorld* in April 2012, Karpinski said of the name "Julia": "There's no good reason, really. It just seemed like a pretty name."[33] Bezanson said he chose the name on the recommendation of a friend.[34]

Since the 2012 launch, the Julia community has grown, and "Julia has been downloaded by users at more than 10,000 companies",[35] with over 20,000,000 downloads as of September 2020, up from 9 million a year prior (and is used at more than 1,500 universities),[36][37][38] The Official Julia Docker images, at Docker Hub, have seen over 4,000,000 downloads as of January 2019.[39][40] The JuliaCon academic conference for Julia users and developers has been held annually since 2014.

Version 0.3 was released in August 2014, version 0.4 in October 2015, version 0.5 in October 2016,[41] and version 0.6 in June 2017.[42] Both Julia 0.7 (a useful release for testing packages, and for knowing how to upgrade them for 1.0[43]) and version 1.0 were released on 8 August 2018. Work on Julia 0.7 was a "huge undertaking" (e.g., because of "entirely new optimizer"), and some changes were made to semantics, e.g. the iteration interface was simplified;[44] and the syntax changed a little (with the syntax now stable, and same for 1.x and 0.7).

The release candidate for Julia 1.0 (Julia 1.0.0-rc1) was released on 7 August 2018, and the final version a day later (and by now, Julia 1.0.x are the oldest versions still supported, having long-term support; for at least a year). Julia 1.1 was released in January 2019 with, e.g., a new "exception stack" language feature. Bugfix releases are expected roughly monthly, for 1.4.x and 1.0.x and Julia 1.0.1 up to 1.0.5 have followed that schedule. Julia 1.2 was released in August 2019, and it has e.g. some built-in support for web browsers (for testing if running in JavaScript VM),[45] and Julia 1.5 in August 2020 (and with it Julia 1.4.x, 1.3.x, 1.2.x and Julia 1.1.x releases are no longer maintained). Julia 1.3 added e.g. composable multi-threaded parallelism and a binary artifacts system for Julia packages.[46]

Julia 1.4 allowed better syntax for array indexing to handle e.g. 0-based arrays, with `A[begin+1]` for the second element of array A.[47] The memory model was also changed.[48] Minor release 1.4.2 fixed e.g. a Zlib issue, doubling decompression speed.[49]

Julia 1.5 adds record and replay debugging support,[50] for Mozilla's rr tool. It's a big release, with changed behavior in the REPL (soft scope), same as used in Jupyter, but fully compatible for non-REPL code. Most of the thread API was marked as stable, and with this release "arbitrary immutable objects—regardless of whether they have fields that reference mutable objects or not—can now be stack allocated",[51] reducing heap allocations, e.g. `views` are no longer allocating. All versions have worked on performance, but especially work on Julia 1.5 targeted so-called "time-to-first-plot" performance, in general, the speed of compilation itself (as opposed to performance of the generated code), and adds tools for developers to improve package loading.[52] Julia 1.6 also improves such performance even more.

Packages that work in Julia 1.0.x should work in 1.1.x or newer, enabled by the forward compatible syntax guarantee. A notable exception was foreign language interface libraries like JavaCall.jl (for JVM languages like Java or Scala) and Rcall.jl (R language) due to some threading-related changes (at a time when all of the threading-functionality in Julia was marked experimental).[53] The issue was especially complicated for Java's JVM, as it has some special expectations around how the stack address space is used. A workaround called `JULIA_ALWAYS_COPY_STACKS` was posted for Julia 1.3.0, while a full fix for Java is pending and has no set due date.[54] In addition, JVM versions since Java 11 do not exhibit this problem.[55] Julia 1.6 has a due date for 30 September 2020. Milestones for Julia 2.0 (and later, e.g. 3.0) currently have no set due dates.[56]

Notable uses

Julia has attracted some high-profile users, from investment manager BlackRock, which uses it for time-series analytics, to the British insurer Aviva, which uses it for risk calculations. In 2015, the Federal Reserve Bank of New York used Julia to make models of the United States economy, noting that the language made model estimation "about 10 times faster" than its previous MATLAB implementation. Julia's co-founders established Julia Computing in 2015 to provide paid support, training, and consulting services to clients, though Julia remains free to use. At the 2017 JuliaCon[57] conference, Jeffrey Regier, Keno Fischer and others announced[58] that the Celeste project[59] used Julia to achieve "peak performance of 1.54 petaFLOPS using 1.3 million threads"[60] on 9300 Knights Landing (KNL) nodes of the Cori II (Cray XC40) supercomputer (then 6th fastest computer in the world).[61] Julia thus joins C, C++, and Fortran as high-level languages in which petaFLOPS computations have been achieved.

Three of the Julia co-creators are the recipients of the 2019 James H. Wilkinson Prize for Numerical Software (awarded every four years) "for the creation of Julia, an innovative environment for the creation of high-performance tools that enable the analysis and solution of computational science problems."[62] Also, Alan Edelman, professor of applied mathematics at MIT, has been selected to receive the 2019 IEEE Computer Society Sidney Fernbach Award "for outstanding breakthroughs in high-performance computing, linear algebra, and computational science and for contributions to the Julia programming language."[63]

Julia Computing and NVIDIA announce "the availability of the Julia programming language as a pre-packaged container on the NVIDIA GPU Cloud (NGC) container registry"[64] with NVIDIA stating "Easily Deploy Julia on x86 and Arm [..] Julia offers a package for a comprehensive HPC ecosystem covering machine learning, data science, various scientific domains and visualization."[65]

Additionally, "Julia was selected by the Climate Modeling Alliance (https://clima.caltech.edu/) as the sole implementation language for their next generation global climate model. This multi-million dollar project aims to build an earth-scale climate model providing insight into the effects and challenges of climate change."[64]

Julia is e.g. used by NASA;[66][67] and Brazilian equivalent (INPE) for space mission planning/satellite simulation[68] (and another user is working on an embedded project to control a satellite in space, i.e. using Julia for attitude control).

Sponsors

The Julia Language became a NumFOCUS Fiscally sponsored project in 2014 in an effort to ensure the projects long term sustainability.[69] Dr. Jeremy Kepner at MIT Lincoln Laboratory was the founding sponsor of the Julia project in its early days. In addition, funds from the Gordon and Betty Moore Foundation, the Alfred P. Sloan Foundation, Intel, and agencies such as NSF, DARPA, NIH, NASA, and FAA have been essential to the development of Julia.[70] Mozilla, the maker of Firefox Web browser, with its research grants for H1 2019, sponsored "a member of the official Julia team" for the project "Bringing Julia to the Browser",[71] meaning to Firefox and other web browsers.[72][73][74][75]

Julia Computing

Julia Computing, Inc. was founded in 2015 by Viral B. Shah, Deepak Vinchhi, Alan Edelman, Jeff Bezanson, Stefan Karpinski and Keno Fischer.[76]

In June 2017, Julia Computing raised $4.6M in seed funding from General Catalyst and Founder Collective,[77] and in the same month was "granted $910,000 by the Alfred P. Sloan Foundation to support open-source Julia development, including $160,000 to promote diversity in the Julia community"[78] and in December 2019 the company got $1.1M funding from the US government to "develop a neural component machine learning tool to reduce the total energy consumption of heating, ventilation, and air conditioning (HVAC) systems in buildings".[79]

Language features

Julia is a general-purpose programming language,[80] while also originally designed for numerical/technical computing. It is also useful for low-level systems programming,[81] as a specification language,[82] and for web programming at both server[83][84] and client[85][8] side.

According to the official website, the main features of the language are:

- Multiple dispatch: providing ability to define function behavior across many combinations of argument types
- Dynamic type system: types for documentation, optimization, and dispatch
- Performance approaching that of statically-typed languages like C
- A built-in package manager
- Lisp-like macros and other metaprogramming facilities
- Call C functions directly: no wrappers or special APIs
- Ability to interface with other languages, e.g. Python with PyCall,[b] R with RCall, and Java/Scala with JavaCall.
- Powerful shell-like abilities to manage other processes
- Designed for parallel and distributed computing
- Coroutines: lightweight *green* threading
- User-defined types are as fast and compact as built-ins
- Automatic generation of efficient, specialized code for different argument types

- Elegant and extensible conversions and promotions for numeric and other types
- Efficient support for Unicode, including but not limited to UTF-8

Multiple dispatch (also termed multimethods in Lisp) is a generalization of single dispatch – the polymorphic mechanism used in common object-oriented programming (OOP) languages – that uses inheritance. In Julia, all concrete types are subtypes of abstract types, directly or indirectly subtypes of the *Any* type, which is the top of the type hierarchy. Concrete types can not themselves be subtyped the way they can in other languages; composition is used instead (see also inheritance vs subtyping).

Julia draws significant inspiration from various dialects of Lisp, including Scheme and Common Lisp, and it shares many features with Dylan, also a multiple-dispatch-oriented dynamic language (which features an ALGOL-like free-form infix syntax rather than a Lisp-like prefix syntax, while in Julia "everything"[89] is an expression), and with Fortress, another numerical programming language (which features multiple dispatch and a sophisticated parametric type system). While Common Lisp Object System (CLOS) adds multiple dispatch to Common Lisp, not all functions are generic functions.

In Julia, Dylan, and Fortress, extensibility is the default, and the system's built-in functions are all generic and extensible. In Dylan, multiple dispatch is as fundamental as it is in Julia: all user-defined functions and even basic built-in operations like + are generic. Dylan's type system, however, does not fully support parametric types, which are more typical of the ML lineage of languages. By default, CLOS does not allow for dispatch on Common Lisp's parametric types; such extended dispatch semantics can only be added as an extension through the CLOS Metaobject Protocol. By convergent design, Fortress also features multiple dispatch on parametric types; unlike Julia, however, Fortress is statically rather than dynamically typed, with separate compiling and executing phases. The language features are summarized in the following table:

Language	Type system	Generic functions	Parametric types
Julia	Dynamic	Default	Yes
Common Lisp	Dynamic	Opt-in	Yes (but no dispatch)
Dylan	Dynamic	Default	Partial (no dispatch)
Fortress	Static	Default	Yes

By default, the Julia runtime must be pre-installed as user-provided source code is run. Alternatively, a standalone executable that needs no Julia source code can be built with *PackageCompiler.jl*.[90]

Julia's syntactic macros (used for metaprogramming), like Lisp macros, are more powerful than text-substitution macros used in the preprocessor of some other languages such as C, because they work at the level of abstract syntax trees (ASTs). Julia's macro system is hygienic, but also supports deliberate capture when desired (like for anaphoric macros) using the `esc` construct.

Interaction

The Julia official distribution includes an interactive command-line read–eval–print loop (REPL),[91] with a searchable history, tab-completion, and dedicated help and shell modes,[92] which can be used to experiment and test code quickly.[93] The following fragment represents a sample session example where strings are concatenated automatically by println:[94]

```
julia> p(x) = 2x^2 + 1; f(x, y) = 1 + 2p(x)y
julia> println("Hello world!", " I'm on cloud ", f(0, 4), " as Julia supports recognizable
syntax!")
Hello world! I'm on cloud 9 as Julia supports recognizable syntax!
```

The REPL gives user access to the system shell and to help mode, by pressing `;` or `?` after the prompt (preceding each command), respectively. It also keeps the history of commands, including between sessions.[95] Code that can be tested inside the Julia's interactive section or saved into a file with a `.jl` extension and run from the command line by typing:[89]

```
$ julia <filename>
```

Julia is supported by Jupyter, an online interactive "notebooks" environment.[96]

Use with other languages

Julia is in practice interoperable with many languages (e.g. majority of top 10-20 languages in popular use). Julia's `ccall` keyword is used to call C-exported or Fortran shared library functions individually, and packages to allow calling other languages, to call e.g. Python, R, MATLAB, Java or Scala,[97] do that indirectly for you. And packages for other languages, e.g. Python (or R or Ruby), i.e. pyjulia, to call *to* Julia do too.

Julia has support for the latest Unicode 13.0,[98] with UTF-8 used for strings (by default) and for Julia source code (only allowing legal UTF-8 in the latest version), meaning also allowing as an option common math symbols for many operators, such as ∈ for the `in` operator.

Julia has packages supporting markup languages such as HTML (and also for HTTP), XML, JSON and BSON, and for databases and Web use in general.

Package system

Julia has a built-in package manager and includes a default registry system.[99] Packages are most often distributed as source code hosted on GitHub, though alternatives can also be used just as well. Packages can also be installed as binaries, using artifacts.[100] Julia's package manager is used to query and compile packages, as well as managing environments. Federated package registries are supported, allowing registries other than the official to be added locally.[101]

Uses

Julia has been used to perform petascale computing with the Celeste library for sky surveys.[102][103] Julia is used by BlackRock Engineering[104] analytical platforms.

Implementation

Julia's core is implemented in Julia and C, together with C++ for the LLVM dependency. The parsing and code-lowering are implemented in FemtoLisp, a Scheme dialect.[105] The LLVM compiler infrastructure project is used as the back end for generation of 64-bit or 32-bit optimized machine code depending on the platform Julia runs on. With some exceptions (e.g., PCRE), the standard library is implemented in Julia. The most notable aspect of Julia's implementation is its speed, which is often within a factor of two relative to fully optimized C code (and thus often an order of magnitude faster than Python or R).[106][107][108] Development of Julia began in 2009 and an open-source version was publicized in February 2012.[4][109]

Current and future platforms

While Julia uses JIT, Julia generates native machine code directly, before a function is first run (i.e. a different approach than compiling to bytecode, that you distribute by default, to be run on a virtual machine (VM), as with e.g. Java/JVM; then translated from the bytecode while running, as done by Dalvik on older versions of Android).

Julia has four support tiers.[110] All 32-bit x86 processors newer than the i686 are supported and 64-bit (Intel) x86-64 (aka amd64), less than about a decade old, are supported. ARMv8 (AArch64) processors are fully supported in first tier, and ARMv7 and ARMv6 (AArch32) are supported with some caveats (lower tier).[111] CUDA (i.e. Nvidia GPUs; implementing PTX) has tier 1 support, with the help of an external package. There are also additionally packages supporting other accelerators, such as Google's TPUs,[112] and AMD's GPUs also have support with e.g. OpenCL; and experimental support for the AMD ROCm stack.[113] Julia's downloads page provides executables (and source) for all the officially supported platforms.

On some platforms, Julia may need to be compiled from source code (e.g., the original Raspberry Pi), with specific build options, which has been done and unofficial pre-built binaries (and build instructions) are available.[114][115] Julia has been built on several ARM platforms. PowerPC (64-bit) has tier 3 support, meaning it "may or may not build". Julia is now supported in Raspbian[116] while support is better for newer Pis, e.g., those with ARMv7 or newer; the Julia support is promoted by the Raspberry Pi Foundation.[117]

There is also support for Web browsers/JavaScript through JSExpr.jl;[85] and the alternative language of Web browsers, WebAssembly, has minimal support[8] for several upcoming external Julia projects. Julia can compile to ARM; thus, in theory, Android apps can be made with the NDK, but for now Julia has been made to run under Android only indirectly, *i.e.* with a Ubuntu chroot on Android.[118]

See also

- Comparison of numerical analysis software
- Comparison of statistical packages

Notes

a. [With Rebugger.jl] you can:
   - test different modifications to the code or arguments as many times as you want; you are never forced to exit “debug mode” and save your file
   - run the same chosen block of code repeatedly (perhaps trying out different ways of fixing a bug) without needing to repeat any of the “setup” work that might have been necessary to get to some deeply nested method in the original call stack.[30]

b. For calling the newer Python 3 (the older default to call Python 2, is also still supported)[86][87] and calling in the other direction, from Python to Julia, is also supported with *pyjulia*.[88]

References

1. "Smoothing data with Julia's @generated functions" (https://medium.com/@acidflask/smoothing-data-with-julia-s-generated-functions-c80e240e05f3#.615wk3dle). 5 November 2015. Retrieved 9 December 2015. "Julia's generated functions are closely related to the multistaged programming (MSP) paradigm popularized by Taha and Sheard, which generalizes the compile time/run time stages of program execution by allowing for multiple stages of delayed code execution."

2. "LICENSE.md" (https://github.com/JuliaLang/julia/blob/master/LICENSE.md). GitHub.
3. "Contributors to JuliaLang/julia" (https://github.com/JuliaLang/julia/graphs/contributors). GitHub.
4. "Why We Created Julia" (https://julialang.org/blog/2012/02/why-we-created-julia). *Julia website*. February 2012. Retrieved 7 February 2013.
5. "v1.5.2" (https://github.com/JuliaLang/julia/releases/tag/v1.5.2). *Github.com*. 24 September 2020. Retrieved 24 September 2020.
6. "WIP: Backports release 1.0.6 by KristofferC · Pull Request #34011 · JuliaLang/julia" (https://github.com/JuliaLang/julia/pull/34011). *GitHub*. Retrieved 14 April 2020.
7. "Julia" (https://julialang.org/). *Julia*. NumFocus project. Retrieved 9 December 2016. "Julia's Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for ..."
8. Fischer, Keno (22 July 2019). "Running julia on wasm" (https://github.com/Keno/julia-wasm). Retrieved 25 July 2019.
9. "Non-GPL Julia?" (https://groups.google.com/forum/#!topic/julia-users/v4OjEK7azBs). *Groups.google.com*. Retrieved 31 May 2017.
10. "Introduce USE_GPL_LIBS Makefile flag to build Julia without GPL libraries" (https://github.com/JuliaLang/julia/pull/10870). "Note that this commit does not remove GPL utilities such as git and busybox that are included in the Julia binary installers on Mac and Windows. It allows building from source with no GPL library dependencies."
11. Stokel-Walker, Chris. "Julia: The Goldilocks language" (https://increment.com/programming-languages/goldilocks-language-history-of-julia/). *Increment*. Stripe. Retrieved 23 August 2020.
12. "Home · The Julia Language" (https://docs.julialang.org/en/v1/). *docs.julialang.org*. Retrieved 15 August 2018.
13. "Programming Language Network" (https://fatiherikli.github.io/programming-language-network/). GitHub. Retrieved 6 December 2016.
14. "JuliaCon 2016" (http://www.juliacon.org). JuliaCon. Retrieved 6 December 2016. "He has co-designed the programming language Scheme, which has greatly influenced the design of Julia"
15. Bryant, Avi (15 October 2012). "Matlab, R, and Julia: Languages for data analysis" (https://web.archive.org/web/20140426110631/https://strata.oreilly.com/2012/10/matlab-r-julia-languages-for-data-analysis.html). O'Reilly Strata. Archived from the original (https://strata.oreilly.com/2012/10/matlab-r-julia-languages-for-data-analysis.html) on 26 April 2014.
16. Singh, Vicky (23 August 2015). "Julia Programming Language – A True Python Alternative" (https://www.technotification.com/2018/08/julia-programming-language.html). Technotification.
17. Krill, Paul (18 April 2012). "New Julia language seeks to be the C for scientists" (https://www.infoworld.com/d/application-development/new-julia-language-seeks-be-the-c-scientists-190818). *InfoWorld*.
18. Finley, Klint (3 February 2014). "Out in the Open: Man Creates One Programming Language to Rule Them All" (https://www.wired.com/2014/02/julia/). *Wired*.
19. "GitHub - JuliaParallel/MPI.jl: MPI wrappers for Julia" (https://github.com/JuliaParallel/MPI.jl). Parallel Julia. Retrieved 22 September 2019.
20. "Questions about getting started with parallel computing" (https://discourse.julialang.org/t/questions-about-getting-started-with-parallel-computing/25341/3?u=palli). *JuliaLang*. 16 June 2019. Retrieved 8 October 2019.
21. "Julia and Concurrency" (https://discourse.julialang.org/t/julia-and-concurrency/25556/2). *JuliaLang*. 24 June 2019. Retrieved 22 September 2019.
22. Fischer, Keno; Nash, Jameson. "Growing a Compiler - Getting to Machine Learning from a General Purpose Compiler" (https://juliacomputing.com/blog/2019/02/19/growing-a-compiler.html). *Julia Computing Blog*. Retrieved 11 April 2019.

23. "Creating a sysimage" (https://julialang.github.io/PackageCompiler.jl/dev/devdocs/sysimages_part_1/). *PackageCompiler Documentation*.
24. "Suspending Garbage Collection for Performance...good idea or bad idea?" (https://groups.google.com/forum/#!topic/julia-users/6_XvoLBzN60). *Groups.google.com*. Retrieved 31 May 2017.
25. now available with `using FFTW` in current versions (That dependency, is one of many which, was moved out of the standard library to a package because it is GPL licensed, and thus is not included in Julia 1.0 by default.) "Remove the FFTW bindings from Base by ararslan · Pull Request #21956 · JuliaLang/julia" (https://github.com/JuliaLang/julia/pull/21956). *GitHub*. Retrieved 1 March 2018.
26. "ANN: linter-julia plugin for Atom / Juno" (https://discourse.julialang.org/t/ann-linter-julia-plugin-for-atom-juno/2118). *JuliaLang*. 15 February 2017. Retrieved 10 April 2019.
27. Holy, Tim (13 September 2019). "GitHub - timholy/ProfileView.jl: Visualization of Julia profiling data" (https://github.com/timholy/ProfileView.jl). Retrieved 22 September 2019.
28. Gregg, Brendan (20 September 2019). "GitHub - brendangregg/FlameGraph: Stack trace visualizer" (https://github.com/brendangregg/FlameGraph). Retrieved 22 September 2019.
29. "A Julia interpreter and debugger" (https://julialang.org/blog/2019/03/debuggers). *julialang.org*. Retrieved 10 April 2019.
30. "[ANN] Rebugger: interactive debugging for Julia 0.7/1.0" (https://discourse.julialang.org/t/ann-rebugger-interactive-debugging-for-julia-0-7-1-0/13843). *JuliaLang*. 21 August 2018. Retrieved 10 April 2019.
31. "Home · Rebugger.jl" (https://timholy.github.io/Rebugger.jl/dev/). *timholy.github.io*. Retrieved 10 April 2019.
32. Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman. "Why We Created Julia" (https://julialang.org/blog/2012/02/why-we-created-julia). *JuliaLang.org*. Retrieved 5 June 2017.
33. Karpinski, Stefan (18 April 2012). "New Julia language seeks to be the C for scientists" (https://www.infoworld.com/article/2616709/application-development/new-julia-language-seeks-to-be-the-c-for-scientists.html). InfoWorld.
34. Torre, Charles. "Stefan Karpinski and Jeff Bezanson on Julia" (https://channel9.msdn.com/Blogs/Charles/Stefan-Karpinski-and-Jeff-Bezanson-Julia-Programming-Language). *Channel 9*. MSDN. Retrieved 4 December 2018.
35. "Newsletter August 2020" (https://juliacomputing.com/blog/2020/08/14/newsletter-aug.html). *juliacomputing.com*. 14 August 2020. Retrieved 7 September 2020.
36. "Julia Computing" (https://juliacomputing.com/). *juliacomputing.com*. Retrieved 12 August 2020.
37. "Newsletter November 2019" (https://juliacomputing.com/blog/2019/11/07/november-newsletter.html). *juliacomputing.com*. 7 November 2019. Retrieved 29 November 2019.
38. "Julia Computing Newsletter, Growth Metrics" (https://juliacomputing.com/blog/2019/01/04/january-newsletter.html). *juliacomputing.com*. Retrieved 11 February 2019.
39. "Newsletter January 2019" (https://juliacomputing.com/blog/2019/01/04/january-newsletter.html). *juliacomputing.com*. 4 January 2019. Retrieved 20 August 2019.
40. "julia - Docker Hub" (https://hub.docker.com/_/julia).
41. "The Julia Blog" (https://julialang.org/blog/).
42. "Julia 0.6 Release Announcement" (https://julialang.org/blog/2017/06/julia-0.6-release).
43. "What is Julia 0.7? How does it relate to 1.0?" (https://discourse.julialang.org/t/what-is-julia-0-7-how-does-it-relate-to-1-0/9994). *JuliaLang*. Retrieved 17 October 2018.
44. Davies, Eric. "Writing Iterators in Julia 0.7" (https://julialang.org/blog/2018/07/iterators-in-julia-0.7). *julialang.org*. Retrieved 5 August 2018.

45. "Sys.isjsvm([os])" (https://github.com/JuliaLang/julia/blob/75c10e435b2b9c947422ad38fa0b020595d3f747/base/sysinfo.jl#L401). The Julia Language. 20 August 2019. Retrieved 20 August 2019. "Predicate for testing if Julia is running in a JavaScript VM (JSVM), including e.g. a WebAssembly JavaScript embedding in a web browser."
46. Bezanson, Jeff; Karpinski, Stefan; Shah, Viral; Edelman, Alan. "The Julia Language" (https://julialang.org/blog/2019/11/artifacts). *julialang.org*. Retrieved 13 December 2019.
47. "support a[begin] for a[firstindex(a)] by stevengj · Pull Request #33946 · JuliaLang/julia" (https://github.com/JuliaLang/julia/pull/33946). *GitHub*. Retrieved 7 April 2020.
48. quinnj. "For structs with all isbits or isbitsunion fields, allow them to be stored inline in arrays · Pull Request #32448 · JuliaLang/julia" (https://github.com/JuliaLang/julia/pull/32448). *GitHub*. Retrieved 7 April 2020. "I still keep running into problems that this causes internally because it was a breaking change that changes assumptions made by some users and inference/codegen."
49. "Bump Zlib BB release to 'v1.2.11+10' which enables '-O3' optimisation by giordano · Pull Request #35979 · JuliaLang/julia" (https://github.com/JuliaLang/julia/pull/35979). *GitHub*. Retrieved 25 May 2020.
50. Fischer, Keno (2 May 2020). "Coming in Julia 1.5: Time Traveling (Linux) Bug Reporting" (https://julialang.org/blog/2020/05/rr/). *julialang.org*. Retrieved 5 May 2020. "Overhead for recording of single threaded processes is generally below 2x, most often between 2% and 50% (lower for purely numerical calculations, higher for workloads that interact with the OS). Recording multiple threads or processes that share memory (as opposed to using kernel-based message passing) is harder. [..] As expected, the threads test is the worst offender with about 600% overhead."
51. Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman et al. "The Julia Language" (https://julialang.org/blog/2020/08/julia-1.5-highlights/). *julialang.org*. Retrieved 14 August 2020. "There are some size-based limits to which structs can be stack allocated, but they are unlikely to be exceeded in practice."
52. Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman et al. "The Julia Language" (https://julialang.org/blog/2020/08/invalidations/). *julialang.org*. Retrieved 16 September 2020.
53. "Fix for C stack checking issues on 1.1 by simonbyrne · Pull Request #293 · JuliaInterop/RCall.jl" (https://github.com/JuliaInterop/RCall.jl/pull/293). *GitHub*. Retrieved 10 August 2019.
54. "JVM fails to load in 1.1 (JavaCall.jl) · Issue #31104 · JuliaLang/julia" (https://github.com/JuliaLang/julia/issues/31104). *GitHub*. Retrieved 18 August 2019. "JeffBezanson modified the milestones: 1.3, 1.4"
55. "StackOverflowError in 'JavaCall.init' for Julia 1.1.0 · Issue #96 · JuliaInterop/JavaCall.jl" (https://github.com/JuliaInterop/JavaCall.jl/issues/96#issuecomment-534178269). *GitHub*. Retrieved 21 October 2019.
56. "Milestones - JuliaLang/julia" (https://github.com/JuliaLang/julia/milestones). The Julia Language. Retrieved 13 December 2019.
57. "JuliaCon 2017" (http://juliacon.org/2017/). *juliacon.org*. Retrieved 4 June 2017.
58. Fisher, Keno. "The Celeste Project" (https://juliacon2017.sched.com/speaker/thecelesteproject). *juliacon.org*. Retrieved 24 June 2017.
59. Regier, Jeffrey; Pamnany, Kiran; Giordano, Ryan; Thomas, Rollin; Schlegel, David; McAulife, Jon; Prabat (2016). "Learning an Astronomical Catalog of the Visible Universe through Scalable Bayesian Inference". arXiv:1611.03404 (https://arxiv.org/abs/1611.03404) [cs.DC (https://arxiv.org/archive/cs.DC)].

60. Claster, Andrew (12 September 2017). "Julia Joins Petaflop Club" (https://juliacomputing.com/press/2017/09/12/julia-joins-petaflop-club.html). *Julia Computing* (Press release). "Celeste is written entirely in Julia, and the Celeste team loaded an aggregate of 178 terabytes of image data to produce the most accurate catalog of 188 million astronomical objects in just 14.6 minutes [..] a performance improvement of 1,000x in single-threaded execution."
61. Shah, Viral B. (15 November 2017). "Viral B. Shah on Twitter" (https://twitter.com/Viral_B_Shah/status/930928375400751105). Retrieved 15 September 2019. "@KenoFischer is speaking on Celeste in the @Intel theatre at @Supercomputing. 0.65M cores, 56 TB of data, Cori - world's 6th largest supercomputer."
62. "Julia language co-creators win James H. Wilkinson Prize for Numerical Software" (https://news.mit.edu/2018/julia-language-co-creators-win-james-wilkinson-prize-numerical-software-1226). *MIT News*. Retrieved 22 January 2019.
63. "Alan Edelman of MIT Recognized with Prestigious 2019 IEEE Computer Society Sidney Fernbach Award | IEEE Computer Society" (https://www.computer.org/press-room/2019-news/2019-ieee-fernbach-award-edelman) (Press release). 1 October 2019. Retrieved 9 October 2019.
64. "Julia Computing and NVIDIA Bring Julia GPU Computing to Arm" (https://juliacomputing.com/blog/2019/12/03/nvidia-ngc-arm.html). *juliacomputing.com*. 3 December 2019. Retrieved 3 December 2019.
65. Patel, Chintan (19 November 2019). "NVIDIA Expands Support for Arm with HPC, AI, Visualization Containers on NGC | NVIDIA Blog" (https://blogs.nvidia.com/blog/2019/11/18/ngc-containers-arm/). *The Official NVIDIA Blog*. Retrieved 3 December 2019.
66. *Circuitscape/Circuitscape.jl* (https://github.com/Circuitscape/Circuitscape.jl), Circuitscape, 25 February 2020, retrieved 26 May 2020
67. "Conservation through Coding: 5 Questions with Viral Shah | Science Mission Directorate" (https://science.nasa.gov/earth-science/applied-sciences/making-space-for-earth/5-questions-with-viral-shah). *science.nasa.gov*. Retrieved 26 May 2020.
68. *JuliaSpace/SatelliteToolbox.jl* (https://github.com/JuliaSpace/SatelliteToolbox.jl), JuliaSpace, 20 May 2020, retrieved 26 May 2020
69. "Julia: NumFOCUS Sponsored Project since 2014" (https://numfocus.org/project/julia). *numfocus.org*. Retrieved 29 September 2020.
70. "The Julia Language" (https://julialang.org/research/). *julialang.org*. Retrieved 22 September 2019.
71. Cimpanu, Catalin. "Mozilla is funding a way to support Julia in Firefox" (https://www.zdnet.com/article/mozilla-is-funding-a-way-to-support-julia-in-firefox/). *ZDNet*. Retrieved 22 September 2019.
72. "Julia in Iodide" (https://alpha.iodide.io/notebooks/225/). *alpha.iodide.io*. Retrieved 22 September 2019.
73. "Language plugins - Iodide Documentation" (https://iodide-project.github.io/docs/language_plugins/). *iodide-project.github.io*. Retrieved 22 September 2019.
74. "Mozilla Research Grants 2019H1" (https://mozilla-research.forms.fm/mozilla-research-grants-2019h1/forms/6510). *Mozilla*. Retrieved 22 September 2019. "running language interpreters in WebAssembly. To further increase access to leading data science tools, we're looking for someone to port R or Julia to WebAssembly and to attempt to provide a level 3 language plugin for Iodide: automatic conversion of data basic types between R/Julia and Javascript, and the ability to share class instances between R/Julia and Javascript."
75. "Literate scientific computing and communication for the web: iodide-project/iodide" (https://github.com/iodide-project/iodide). iodide. 20 September 2019. Retrieved 22 September 2019. "We envision a future workflow that allows you to do your data munging in Python, fit a quick model in R or JAGS, solve some differential equations in Julia, and then display your results with a live interactive d3+JavaScript visualization ... and all that within within a single, portable, sharable, and hackable file."

76. "About Us – Julia Computing" (https://juliacomputing.com/about-us). *juliacomputing.com*. Retrieved 12 September 2017.
77. "Julia Computing Raises $4.6M in Seed Funding" (https://web.archive.org/web/20190510040656/https://juliacomputing.com/communication/2017/06/19/seed-funding.html). Archived from the original (https://juliacomputing.com/communication/2017/06/19/seed-funding.html) on 10 May 2019.
78. "Julia Computing Awarded $910,000 Grant by Alfred P. Sloan Foundation, Including $160,000 for STEM Diversity" (https://juliacomputing.com/media/2017/06/26/sloan-grant.html). *juliacomputing.com*. 26 June 2017. Retrieved 28 July 2020.
79. "DIFFERENTIATE—Design Intelligence Fostering Formidable Energy Reduction (and) Enabling Novel Totally Impactful Advanced Technology Enhancements" (https://arpa-e.energy.gov/sites/default/files/documents/files/DIFFERENTIATE_Project_Descriptions_FINAL.pdf) (PDF).
80. "The Julia Language" (https://julialang.org/) (official website). "General Purpose [..] Julia lets you write UIs, statically compile your code, or even deploy it on a webserver."
81. Green, Todd (10 August 2018). "Low-Level Systems Programming in High-Level Julia" (https://web.archive.org/web/20181105083419/http://juliacon.org/2018/talks_workshops/42/). Archived from the original (http://juliacon.org/2018/talks_workshops/42/) on 5 November 2018. Retrieved 5 November 2018.
82. Moss, Robert (26 June 2015). "Using Julia as a Specification Language for the Next-Generation Airborne Collision Avoidance System" (https://juliacon.org/2015/images/juliacon2015_moss_v3.pdf) (PDF). Archived (https://web.archive.org/web/20150701182804/http://juliacon.org/talks.html) from the original on 1 July 2015. Retrieved 29 June 2015. "Airborne collision avoidance system"
83. Anaya, Richard (28 April 2019). "How to create a multi-threaded HTTP server in Julia" (https://medium.com/@richardanaya/how-to-create-a-multi-threaded-http-server-in-julia-ca12dca09c35). *Medium*. Retrieved 25 July 2019. "In summary, even though Julia lacks a multi-threaded server solution currently out of box, we can easily take advantage of its process distribution features and a highly popular load balancing tech to get full CPU utilization for HTTP handling."
84. Anthoff, David (1 June 2019). "Node.js installation for julia" (https://github.com/davidanthoff/NodeJS.jl). Retrieved 25 July 2019.
85. "Translate Julia to JavaScript" (https://github.com/JuliaGizmos/JSExpr.jl). JuliaGizmos. 7 July 2019. Retrieved 25 July 2019.
86. "PyCall.jl" (https://github.com/JuliaPy/PyCall.jl). *stevengj*. github.com.
87. "Using PyCall in julia on Ubuntu with python3" (https://groups.google.com/forum/#!topic/julia-users/lDM7-YXT2LU). *julia-users at Google Groups*. "to import modules (e.g., python3-numpy)"
88. "python interface to julia" (https://github.com/JuliaPy/pyjulia).
89. "Learn Julia in Y Minutes" (https://learnxinyminutes.com/docs/julia/). *Learnxinyminutes.com*. Retrieved 31 May 2017.
90. "GitHub - JuliaLang/PackageCompiler.jl: Compile your Julia Package" (https://github.com/JuliaLang/PackageCompiler.jl). The Julia Language. 14 February 2019. Retrieved 15 February 2019.
91. "The Julia REPL · The Julia Language" (https://docs.julialang.org/en/v1/stdlib/REPL/). *docs.julialang.org*. Retrieved 22 September 2019.
92. "Introducing Julia/The REPL - Wikibooks, open books for an open world" (https://en.wikibooks.org/wiki/Introducing_Julia/The_REPL). *en.wikibooks.org*. Retrieved 22 September 2019. "you can install the Julia package OhMyREPL.jl (github.com/KristofferC/OhMyREPL.jl (https://github.com/KristofferC/OhMyREPL.jl)) which lets you customize the REPL's appearance and behaviour"

93. "Getting Started · The Julia Language" (https://docs.julialang.org/en/v1/manual/getting-started/). *docs.julialang.org*. Retrieved 15 August 2018.
94. See also: docs.julialang.org/en/v1/manual/strings/ (https://docs.julialang.org/en/v1/manual/strings/) for string interpolation and the `string(greet, ", ", whom, ".\n")` example for preferred ways to concatenate strings. Julia has the println and print functions, but also a @printf macro (i.e., not in function form) to eliminate run-time overhead of formatting (unlike the same function in C).
95. "Julia Documentation" (https://docs.julialang.org). *JuliaLang.org*. Retrieved 18 November 2014.
96. "Project Jupyter" (https://jupyter.org/).
97. "Julia and Spark, Better Together" (https://juliacomputing.com/blog/2020/06/02/julia-spark.html). *juliacomputing.com*. 2 June 2020. Retrieved 14 July 2020.
98. "Unicode 13 support by stevengj · Pull Request #179 · JuliaStrings/utf8proc" (https://github.com/JuliaStrings/utf8proc/pull/179). *GitHub*. Retrieved 29 March 2020.
99. "JuliaRegistries / General" (https://github.com/JuliaRegistries/General). Retrieved 30 April 2020.
00. "Pkg.jl - Artifacts" (https://julialang.github.io/Pkg.jl/dev/artifacts/). Retrieved 4 June 2020.
01. "Pkg.jl - Registries" (https://julialang.github.io/Pkg.jl/v1/registries/). Retrieved 30 April 2020.
02. Farber, Rob (28 November 2017). "Julia Language Delivers Petascale HPC Performance" (https://www.nextplatform.com/2017/11/28/julia-language-delivers-petascale-hpc-performance/). *The Next Platform*. Retrieved 22 April 2020.
03. Kincade, Kathy (11 November 2016). "Celeste Enhancements Create New Opportunities in Sky Surveys" (https://cs.lbl.gov/news-media/news/2016/celeste-enhancements-create-new-opportunities-in-sky-surveys/). *Berkeley Lab*. Retrieved 22 April 2020.
04. Francis, Michael (9 May 2017). "OS@BLK: Julia NamedTuples" (http://rockthecode.io/blog/osblk-julia-namedtuples/). *BlackRock Engineering*. Retrieved 22 April 2020.
05. Bezanson, Jeff (6 June 2019). "JeffBezanson/femtolisp" (https://github.com/JeffBezanson/femtolisp). *GitHub*. Retrieved 16 June 2019.
06. "Julia: A Fast Dynamic Language for Technical Computing" (https://julialang.org/images/julia-dynamic-2012-tr.pdf) (PDF). 2012.
07. "How To Make Python Run As Fast As Julia" (https://www.ibm.com/developerworks/community/blogs/jfp/entry/Python_Meets_Julia_Micro_Performance?lang=en). 2015.
08. "Basic Comparison of Python, Julia, R, Matlab and IDL" (https://modelingguru.nasa.gov/docs/DOC-2625). 2015.
09. Gibbs, Mark (9 January 2013). "Pure and Julia are cool languages worth checking out" (https://www.networkworld.com/columnists/2013/010913-gearhead.html). *Network World* (column). Retrieved 7 February 2013.
10. "Julia Downloads" (https://julialang.org/downloads/#support-tiers). *julialang.org*. Retrieved 17 May 2019.
11. "julia/arm.md" (https://github.com/JuliaLang/julia/blob/master/doc/build/arm.md). The Julia Language. 29 November 2019. Retrieved 29 November 2019. "A list of known issues (https://github.com/JuliaLang/julia/labels/arm) for ARM is available."
12. "Julia on TPUs" (https://github.com/JuliaTPU/XLA.jl). JuliaTPU. 26 November 2019. Retrieved 29 November 2019.
13. "AMD ROCm · JuliaGPU" (https://juliagpu.org/rocm/). *juliagpu.org*. Retrieved 20 April 2020.
14. 262588213843476. "Build Julia for RaspberryPi Zero" (https://gist.github.com/terasakisatoshi/3f8a55391b1fc22a5db4a43da8d92c98). *Gist*. Retrieved 14 August 2020.
15. "JuliaBerry: Julia on the Raspberry Pi" (https://juliaberry.github.io/). *juliaberry.github.io*. Retrieved 14 August 2020.
16. "Julia available in Raspbian on the Raspberry Pi" (https://julialang.org/blog/2017/05/raspberry-pi-julia). "Julia works on all the Pi variants, we recommend using the Pi 3."

17. "Julia language for Raspberry Pi" (https://www.raspberrypi.org/blog/julia-language-raspberry-pi/). *Raspberry Pi Foundation*.
18. "Using Julia on Android?" (https://discourse.julialang.org/t/using-julia-on-android/8086/7). *JuliaLang*. 27 September 2019. Retrieved 2 October 2019.

Further reading

- Nagar, Sandeep (2017). *Beginning Julia Programming-For Engineers and Scientists*. Springer.
- Bezanson, J; Edelman, A; Karpinski, S; Shah, V. B (2017). "Julia: A fresh approach to numerical computing". **59** (1). SIAM Review: 65–98.
- Joshi, Anshul (2016). *Julia for Data Science — Explore the world of data science from scratch with Julia by your side*. Packt Publishing.

External links

- Official website (https://julialang.org)
- julia (https://github.com/JuliaLang/julia) on GitHub