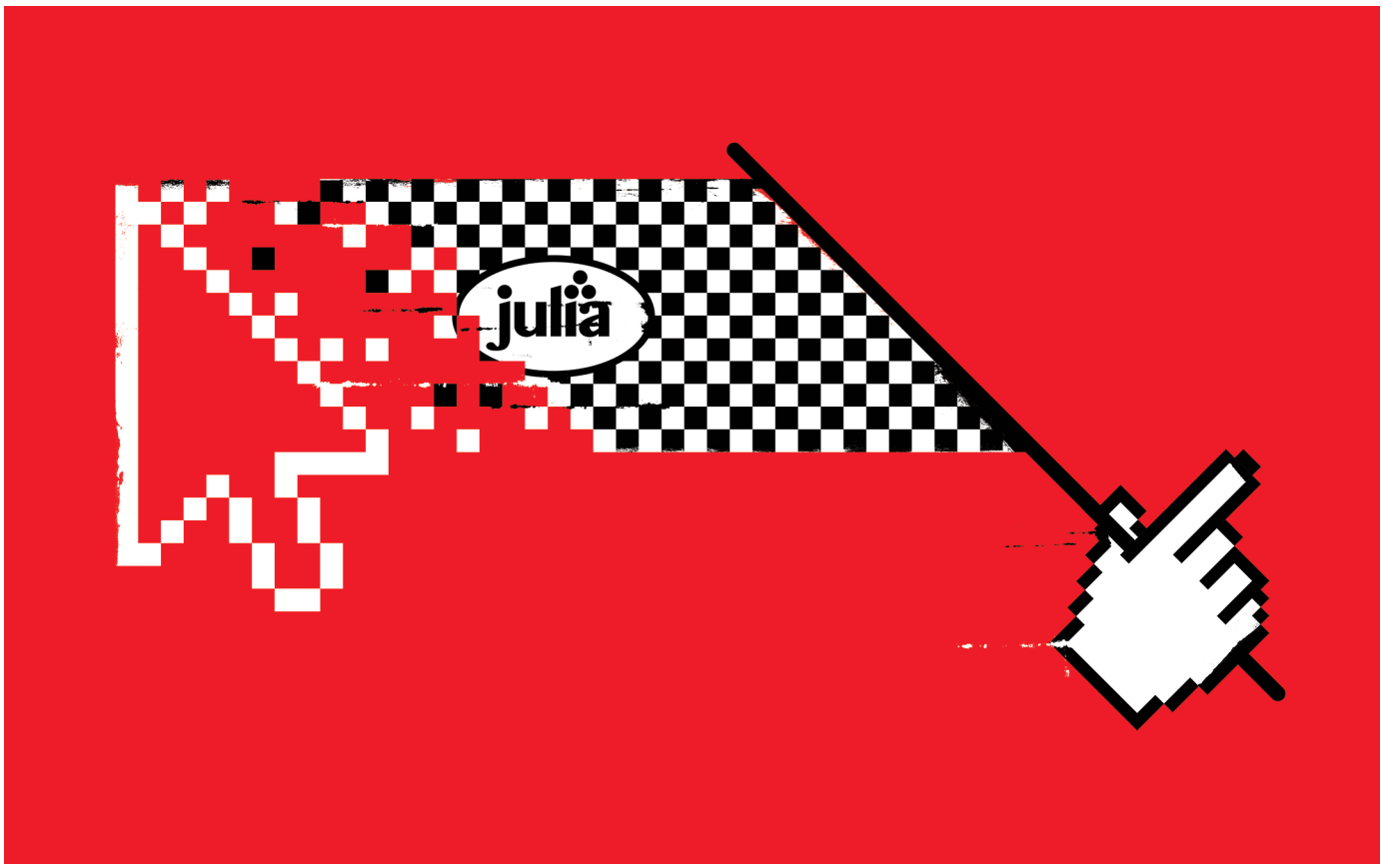
TOOLBOX

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.

ILLUSTRATION BY THE PROJECT TWINS



BY JEFFREY M. PERKEL

When it comes to climate modelling, every computational second counts. Designed to account for air, land, sun and sea, and the complicated physics that links them, these models can run to millions of lines of code, which are executed on the world's most powerful computers. So when the coder-climatologists of the Climate Modeling Alliance (CliMA) — a coalition of US-based scientists, engineers and mathematicians — set out to build a model from the ground up, they opted for a language that could handle their needs. They opted for 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.

Among climate scientists, the lingua franca is Fortran: speedy, but — with roots dating to the 1950s — not terribly exciting. "A lot of people, when they hear 'Fortran', are like, 'Oh, my God, I don't want to program in that'," says Frank Giraldo, a mathematician at the Naval Postgraduate School in Monterey, California, and a co-principal investigator on the CliMA project. Younger programmers prefer languages that can accommodate the latest trends in software and hardware design, Giraldo says, and since adopting Julia he has seen an uptick in interest. "Some of them are really interested in climate modelling, but others are intrigued by the idea of using Julia for some large-scale application," he says.

Jane Herriman, who is studying materials science at the California Institute of Technology in Pasadena, says that she has seen tenfold-faster runs since rewriting her Python codes in Julia. Michael Stumpf, a systems biologist and self-styled Julia proselytizer at the University of Melbourne, Australia, who has ported computational models from R, has seen an 800-fold improvement. "You can do things in an hour that would otherwise take weeks or months," he says.

That acceleration, combined with Julia's user-friendly syntax and its promise to tackle the 'two-language problem' — researchers often prototype algorithms in a user-friendly language such as Python but then have to rewrite them in a faster language — is raising the language's profile, particularly among those dealing with computationally intensive problems. Besides climate modelling, the language is being adopted in disciplines such as artificial intelligence, finance and bioinformatics.

According to Alan Edelman, a computer scientist at the Massachusetts Institute of Technology in Cambridge who co-created Julia, the language has been downloaded some 9 million times so far. Julia is now listed among the world's 50 most-popular programming languages, according to one index. It's still relatively niche — the 2019 index ranks Julia 50th, and Python 3rd — but it has a passionate user base.

"People are just tired of rewriting code," Edelman says. "They're tired of codes that obscure what their intent is, they're tired of some researcher or graduate student or employee writing code and then moving on to their next job and nobody knows what to do with their code anymore. Those are the people who are using Julia — people who want performance, portability, flexibility."

BEST OF BOTH WORLDS

Julia — the name puts the 'Ju' in 'Jupyter', a computational notebook system popular among data scientists, alongside Python and R — is essentially a compiled language in scripting-language clothing. In scripting languages such as Python, users type code into an interactive editor line by line, and the language interprets and executes it, returning the result immediately. With languages such as C and Fortran, code must be compiled into machine-readable instructions before it can be executed. The former is easier to use, but the latter produces faster code. As a result, programmers for whom speed counts often develop algorithms in scripting languages and then translate them into C or Fortran, a laborious and error-prone process.

Julia circumvents that two-language problem because it runs like C, but reads like Python. And it includes built-in features to accelerate computationally intensive problems, such as distributed computing, that otherwise require multiple languages. (Distributed computing allows programmers to split difficult problems across multiple processors and computers.) Vijay Ivaturi, a quantitative clinical pharmacologist at the University of Maryland in Baltimore, used Julia to create a tool for personalizing drug-dosing decisions. His previous go-to language, Fortran, required him to use several ancillary tools. "I fell in love with the speed of Julia," he says. "But overall, I think I fell in love with the fact that I don't have to switch [language] tools to get my work done: I can live in one environment through and through."

Users typically code in Julia using the REPL (read–evaluate–print loop) console — an interactive text-based interface that takes the input, evaluates it and returns the results to the user. (They can also use a standard programming text editor, or the Jupyter notebook.) To all appearances, using Julia is like coding in Python: type a line, get a result. But in the background, the code is compiled. Consequently, the first time a function is keyed in, it might be slow, but subsequent runs are faster. And once the code is working correctly, users can optimize it (see 'Get started').

According to Giraldo, one reason that CliMA selected Julia for its work was its performance in a Christmas 'bake-off'-style coding challenge against C and Fortran, using Giraldo — then a Julia novice — as a guinea pig. "The Julia code, right out of the box, was really within a couple of per cent away from these highly optimized Fortran codes," he says.

And it's easier to read, he adds. With features such as multiple dispatch (allowing multiple functions to have the same name) and metaprogramming (programs that can modify themselves), the language emphasizes simplicity. Julia also supports Unicode symbols, allowing programmers to use Greek letters as variables, rather than Roman equivalents. This means that they can write code that resembles the maths in their papers, with $C = 2*\pi*r$ for the circumference of a circle, instead of $C = 2*pi*r$. "You could express things exactly how your mind thinks about them," says Edelman. "You want the machine to bend to your will, not you to bend to the machine's will."

Get started

Set up

- Julia: julialang.org
- Juno, a free Julia language 'integrated development environment' including a code editor, debugging tools and interactive console: junolab.org
- Debugger: go.nature.com/2jdfr5g
- IJulia, a 'kernel' for writing Julia code in Jupyter: go.nature.com/2jldaj2
- Packages: go.nature.com/30brtxe

Learn

- julialang.org/learning/
- Julia language documentation: go.nature.com/2nxrqup
- Think Julia: go.nature.com/2y7skii

Get help

- Slack: julialang.slack.com
- Discourse: discourse.julialang.org
- Gitter: gitter.im/JuliaLang/julia
- An interactive and executable Julia notebook, highlighting some key features, is available at go.nature.com/2lxllfd. J.P.

FAST, POWERFUL AND EASY

Michael Borregaard, a biodiversity researcher at the University of Copenhagen, says Julia has accelerated his codes by two orders of magnitude compared with R — a result of both computational speed and linguistic clarity. "Coding it into Julia made it a lot easier for me to refactor it for speed, or to rethink how I implemented it to make it go faster," he says.

For George Tollefson, a clinical research assistant at the Women and Infants Hospital of Rhode Island in Providence, it was Julia's blend of user-friendliness and computational power that made it ideal for writing a data viewer for large genomic data sets. "Julia was an attractive language to us in the beginning because it is very fast and powerful," he says. "But it's also very easy to learn to write in." And it has a supportive community, Tollefson adds. Because the language has a relatively small user base, it can be difficult to find answers online. But developer communities on Slack, Discourse and GitHub can fill the gap. "In some cases we found that people hadn't encountered the problem [we had], but they were able to help us within half an hour," Stumpf says.

That said, a smaller user base also translates to a correspondingly smaller package ecosystem — the collection of external code libraries that programmers use to extend a language into new disciplines. According to Edelman, the Julia ecosystem has upwards of 2,600 packages, including Flux (machine learning), BioJulia (DNA sequence analysis), DifferentialEquations (computational simulations) and JuMP (mathematical modelling). By comparison, the CRAN R language repository has more than 14,000 packages, and Python's PyPI index exceeds 187,000.

Researchers who require libraries that haven't been translated into Julia can use the code directly using packages such as Pycall (Python) and Rcall (R). As an undergraduate at the Massachusetts Institute of Technology, Lydia Krasilnikova, now a computer-science graduate student at Harvard University in Cambridge, Massachusetts, created a Matlab-to-Julia translator, which is available online. "A lot of people have messaged me saying that the translator eased their transition and let them test code in Julia and tinker with their existing code base in ways they wouldn't have been able to before," she says.

Ultimately, the choice of language comes down to personal preference, project requirements and your colleagues. In many cases, any language will do. But for "performant code", says Giraldo, "then, honestly, right now I see Julia as really the best choice. You have to suck it up and just dive in. It's not really that difficult." ■

Jeffrey M. Perkel *is technology editor at* Nature.