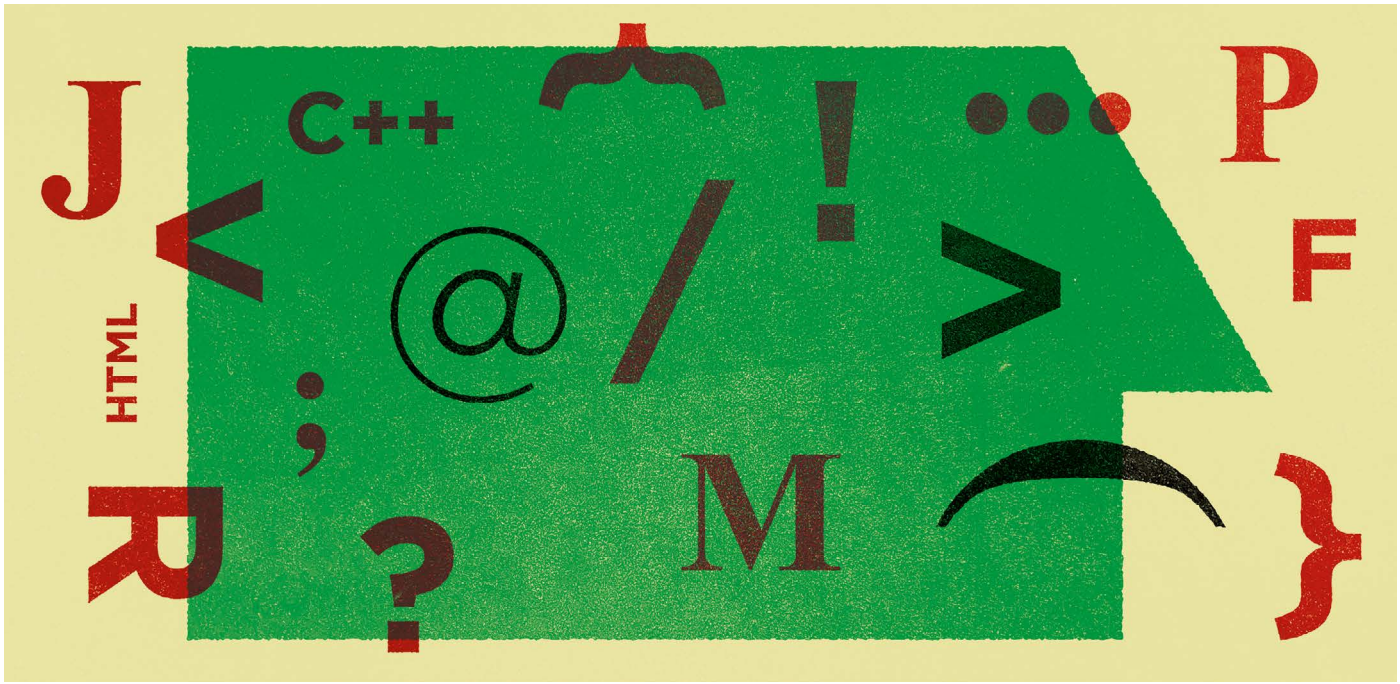

ILLUSTRATION: THE PROJECT TWINS

HOW TO PICK A PROGRAMMING LANGUAGE

Computer scientists and bioinformaticians address four key questions to help rookie coders to make the right choice. **By Jeffrey M. Perkel**

Each year, the code-sharing platform GitHub releases its 'State of the Octoverse' report, which among other things ranks the popularity of programming languages. The latest report, released in October 2024, had some good news for pythonistas, as Python programmers are called: for the first time in ten years, the language JavaScript had been bumped from the top of the leader board and replaced by Python.

"This is the first large-scale change we've seen in the top two languages since 2019 – and it speaks to the rise in Python that's accompanied the generative [artificial intelligence] boom we've seen over the past two years," the report says.

For researchers who have watched the growing fusion of science and coding, that news perhaps answers a basic but rarely asked question: with so many programming languages to choose from, which one should I learn?

But the choice is not that simple.

"For a very long time in computer science, a lot of people who work in programming languages have had the ostensible goal of [creating] the 'one language to rule them all'," says Rob Patro, a computational biologist at the University of Maryland in College Park. But that's a bit like a carpenter who, armed only with a hammer, treats everything as a nail: different situations might call for different tools, and there is no single 'best' language.

Nature asked computer scientists and bioinformaticians what advice they would give to researchers who recognize the need to pick up some coding skills but don't know where to start. Here are four key questions to help you decide.

What do you mean by 'programming'?

Some researchers build tools, others use them. Both are 'programmers', but the style of programming and the skills required are different.

"Somebody has to make the lathe; somebody has to make the electron microscope," says Greg Wilson, a software engineering manager at Plotly, a company in Montreal, Canada, that develops interactive graphing tools. "But most scientists don't need to know how to do that – they need to know how to use those tools, not how to make them from scratch."

The computational 'lathe' in this analogy is software designed to solve a given problem accurately and efficiently – say, aligning DNA-sequencing readouts to a reference genome. The code to do that is often mathematics-heavy and memory-intensive; it can require multiple processors working in tandem; and it is often written in languages such as C/C++, Rust and Fortran. These are 'compiled' languages – they require a compilation step to translate human-written code into instructions the computer can execute, and demand a deeper understanding of how computers work, but they produce fast, highly optimized software.

Most scientists, however, are data wranglers who want, for instance, to quantify gene expression by aligning RNA sequences to a reference genome rather than building a tool to do the alignment. This data workflow is typically accomplished using 'scripting' languages such as Python, R or Matlab, often in concert with computational notebooks such as Jupyter, Quarto or marimo (see 'A notebook for reproducible Python code'). Such languages do not need to be compiled and are interpreted by computers directly, line by line. That makes this workflow interactive and easy to learn – type a command, get a result, repeat – but relatively slow, because the computer has no opportunity to optimize what it's doing.

Web interfaces that help to make those tools broadly available to users are often written in JavaScript, and the databases underlying those interfaces might use a different language, such

as SQL. And then there are tools that tie these pieces together — another form of programming. You can do a lot of data manipulation at the text-based command line, for instance. Workflow languages such as Snakemake and Nextflow make it easy to string tools together into sophisticated computational pipelines.

What are your colleagues using?

For many programming tasks, almost any language will do. But for beginners, it's good to choose one that a colleague can help with. Furthermore, if everyone in your field is using a particular language, it helps to be using the same one, too.

Edoardo Saccenti, who specializes in systems-data analysis and applied statistics at Wageningen University & Research in the Netherlands, has a good command of multiple programming languages. For transcriptome analysis, he uses R. "All of the most-used packages and tools have been developed in R," he explains. But Saccenti also studies psychometrics, a branch of psychology that evaluates how psychological traits are measured and quantified. In that case, he turns mostly to Matlab. "I've never seen a psychometry paper written in Python," he says.

Which tools are available?

Coders can extend the core capabilities of programming languages using 'libraries' — collections of software routines that provide further functions. Many popular machine-learning libraries were developed in Python; the Bioconductor collection of bioinformatics tools works in R; and alevin-fry, a tool for processing single-cell RNA-sequencing data, was written in Rust.

Yanina Bellini Saibene is a devotee of R. Aside from her role as community manager for rOpenSci, a non-profit initiative in Berkeley, California, that provides open-source software tools for scientists, she is a member of the leadership committee for R-Ladies+ Global, a group that supports women and programmers from minority gender groups. "I think the only thing that I don't do using R is my toast for my breakfast," she jokes. But as a graduate student at the National Institute of Agricultural Technology in La Pampa, Argentina, she used Python because the weather radar data she was studying could not easily be manipulated any other way. "At that time, in Python we had more libraries to work with that kind of data than the ones I had in R," she says.

How big are your data?

Fast and slow are relative in computational research, and even 'slow' software can be reasonably zippy. Sometimes, however, the distinction matters. Patro, for instance, builds tools for genomics studies that can involve thousands of gigabyte-sized data sets. At that scale, many scripting languages can't keep up.

Every program has to allocate memory, for instance — it's what happens when you create a variable in the program to hold data. In compiled languages such as Rust, coders must specifically allocate the memory they want, then 'free' it up when it is no longer needed; scripting languages usually hide those details. As a result, scripting languages are easier to learn but less memory-efficient. "When you're building data structures designed to barely fit into the working memory of a large server, a factor of two takes something from being completely infeasible to being something that you could run if you have dedicated access to the machine," Patro explains. So, his team develops its algorithms in Rust.

Titus Brown, a bioinformatician at the University of California, Davis, says newcomers to his group often start with R, because it's easy to learn and has all the required bioinformatics tools. But many reach a stage at which they want to analyse thousands of genomes rather than a handful, which then produces hundreds of thousands of output files. "At that point, people will often be directed more towards Python, because Python has a broader array of abilities to deal with that scope of data," says Brown.

Help is at hand

The user community is also a key consideration — Saibene says that's one of the things she loves most about R. The language's large, welcoming and engaged user community means that tools are developed and updated frequently; that local user groups exist around the world; and that tutorials and other resources are available in many spoken languages, including her native Spanish. The alternative — having resources available only in English — imposes a "cognitive load" on many users, she says. "You are going to remove a lot of barriers if those resources are in Spanish."

Another consideration is economics. Although many languages are free to use, some require users to pay for a licence, which could put them out of reach of many researchers. Some scientists also prefer to use open-source software instead of proprietary systems. That said, tool developers often create 'bindings' so that tools they have written in one language can be used directly in multiple others — and many programming languages include ways to execute code in others. Artificial-intelligence tools, such as GitHub Copilot, make it easy to translate code from one language to another or to generate code in an unfamiliar programming language.

There's no shortage of online help, whatever language you choose. Useful resources include The Carpentries, the Data Science Learning Community and Stack Overflow.

Jeffrey M. Perkel is Technology Editor at *Nature*.

A notebook for reproducible Python code

An open-source platform called marimo is designed for computational reproducibility and sharing.

Interactive code-writing environments such as Jupyter Notebook are beloved by data analysts because they combine software code, computational output and explanatory text in a single document. However, reproducibility advocates find them more of a problem: code cells can be executed out of order, and changing a value in one cell has no impact on other cells that depend on it. Thus, the 'state' of a notebook can easily get out of sync with its code. And the file format of Jupyter Notebooks means that version control can be difficult.

To try to overcome these reproducibility issues, in 2022, machine-learning researcher Akshay Agrawal, a newly minted PhD graduate from Stanford University in California, began work on an alternative: marimo. With funding from the SLAC National Accelerator Laboratory in Menlo Park, California, Agrawal and software developer Myles Scolnick launched the notebook in January 2024.

Like a spreadsheet, marimo is reactive — changing a value in one cell causes dependent cells to update themselves. It's also interactive, with user-interface elements such as slider bars and drop-down boxes making it easy to fiddle with variables. Thanks to Pyodide, a version of Python that runs in a web browser, researchers can share executable forms of their notebooks and publish them online. And because the format is that of a simple Python file, version control is easier, too.

Bennet Meyers, a staff scientist at the SLAC National Accelerator Laboratory, uses marimo to create interactive tutorials, promote computational reproducibility and analyse data. The tool has largely displaced Jupyter Notebooks in his day-to-day work, he says. "The notebook style of marimo is extremely useful for quickly putting together exploratory data visualizations."

Like Jupyter, marimo is free and open-source. According to Agrawal, it has been downloaded more than two million times since its launch.