Julia vs. Python: Julia language rises for data science

Python has turned into a data science and machine learning mainstay, while Julia was built from the ground up to do the job

By Serdar Yegulalp
Senior Writer, InfoWorld

DEC 20, 2017

Table of Contents

Of the many use cases Python covers, data analytics has become perhaps the biggest and most significant. The Python ecosystem is loaded with libraries, tools, and applications that make the work of scientific computing and data analysis fast and convenient.

But for the developers behind the Julia language — aimed specifically at "scientific computing, machine learning, data mining, large-scale linear algebra, distributed and parallel computing"—Python isn't fast or convenient *enough*. It's a trade-off, good for some parts of this work but terrible for others.

What is the Julia language?

Created in 2009 by a four-person team and unveiled to the public in 2012, Julia is meant to address the shortcomings in Python and other languages and applications used for scientific computing and data processing. "We are greedy," they wrote. They wanted more:

> *We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.*
>
> *(Did we mention it should be as fast as C?)*

Here are some of the ways Julia implements those aspirations:

- **Compiled, not interpreted, for speed.** Julia is just-in-time (JIT) compiled using the LLVM compiler framework. At its best, Julia can approach or match the speed of C.
- **Straightforward but useful syntax.** Julia's syntax is similar to Python's—terse, but also expressive and powerful.
- **Dynamic typing with static type benefits.** You can specify types for variables, like "unsigned 32-bit integer." But you can also create hierarchies of types to allow general cases for handling variables of specific types—for instance, to write a function that accepts integers generally without specifying the length or signing of the integer. And, finally, you can do without typing entirely if it isn't needed in a particular context.
- **Python, C, and Fortran libraries are just a call away.** Julia can interface directly with external libraries written in C and Fortran. It's also possible to interface with Python code by way of the PyCall library, and even share data between Python and Julia.
- **Metaprogramming.** Julia programs can generate other Julia programs, and even modify their own code, in a way that is reminiscent of languages like Lisp.

Julia vs Python: Julia language advantages

Julia was designed from the start for scientific and numerical computation. Thus it's no surprise that Julia has many features advantageous for such use cases:

- **Faster by default.** Julia's JIT compilation and type declarations mean it can routinely beat "pure," unoptimized Python by orders of magnitude. Python can be *made* faster by way of external libraries, third-party JIT compilers (PyPy), and optimizations with tools like Cython, but Julia is designed to be faster right out of the gate.
- **A math-friendly syntax.** A major target audience for Julia is users of scientific computing languages and environments like Matlab, R, Mathematica, and Octave. Julia's syntax for math operations looks more like the way math formulas are written outside of the computing world, making it easier for non-programmers to pick up on.
- **Automatic memory management.** Like Python, Julia doesn't burden the user with the details of allocating and freeing memory, and it provides some measure of manual control over garbage collection. The idea is that if you switch to Julia, you don't lose one of Python's common conveniences.
- **Parallelism.** Math and scientific computing thrive when you can make use of the full resources available on a given machine, especially multiple cores. Both Python and Julia can run operations in parallel. But Julia's syntax is slightly less top-heavy than Python's, lowering the threshold to its use.

Python vs Julia: Python advantages

Python is a general-purpose computing language that is easy to learn, and that has developed into a leading language for scientific computing. Some of the reasons Python may still be the better choice for data science work:

- **Julia arrays are 1-indexed.** This might seem like an obscure issue, but it's a potentially jarring one. In most languages, Python and C included, the first element of an array is accessed with a zero—e.g., `string[0]` in Python for the first character in a string. Julia uses 1 for the first element in an array. This isn't an arbitrary decision; many other math and science applications, like Mathematica, use 1-indexing, and Julia is intended to appeal to that audience. It's possible to support zero-indexing in Julia with an experimental feature, but 1-indexing by default may stand in the way of adoption by a more general-use audience with ingrained programming habits.

- **Julia is still young.** The Julia language has been under development since 2009, and has undergone a fair amount of feature churn along the way. It still doesn't have a 1.0 release, although the developers are getting close.
- **Python has far more third-party packages.** The breadth and usefulness of Python's culture of third-party packages remains one of the language's biggest attractions. Again, Julia's relative newness means the culture of software around it is still small. Some of that is offset by the ability to use existing C and Python libraries, but Julia needs libraries of its own to thrive.
- **Python's huge community is a huge advantage.** A language is nothing without a large, devoted, and active community around it. Python enjoys just such a community right now. The community around Julia is enthusiastic and growing, but it is still only a fraction of the size of the Python community.

Serdar Yegulalp is a senior writer at InfoWorld, focused on machine learning, containerization, devops, the Python ecosystem, and periodic reviews.

Follow