OPINION

Bye-bye Python. Hello Julia!

As Python's lifetime grinds to a halt, a hot new competitor is emerging

Rhea Moutafis
May 2 · 8 min read ★

If Julia is still a mystery to you, don't worry. Photo by Julia Caesar on Unsplash

Don't get me wrong. Python's popularity is still backed by a rock-solid community of computer scientists, data scientists and AI specialists.

But if you've ever been at a dinner table with these people, you also know how much they rant about the weaknesses of Python. From being slow to requiring excessive testing, to producing runtime errors despite prior testing — there's enough to be pissed off about.

Which is why more and more programmers are adopting other languages — the top players being Julia, Go, and Rust. Julia is great for mathematical and technical tasks, while Go is awesome for modular programs, and Rust is the top choice for systems programming.

Since data scientists and AI specialists deal with lots of mathematical problems, Julia is the winner for them. And even upon critical scrutiny, Julia has upsides that Python can't beat.

Why Python is not the programming language of the future

Even though it will be in high demand for a few more years

towardsdatascience.com

The Zen of Python versus the Greed of Julia

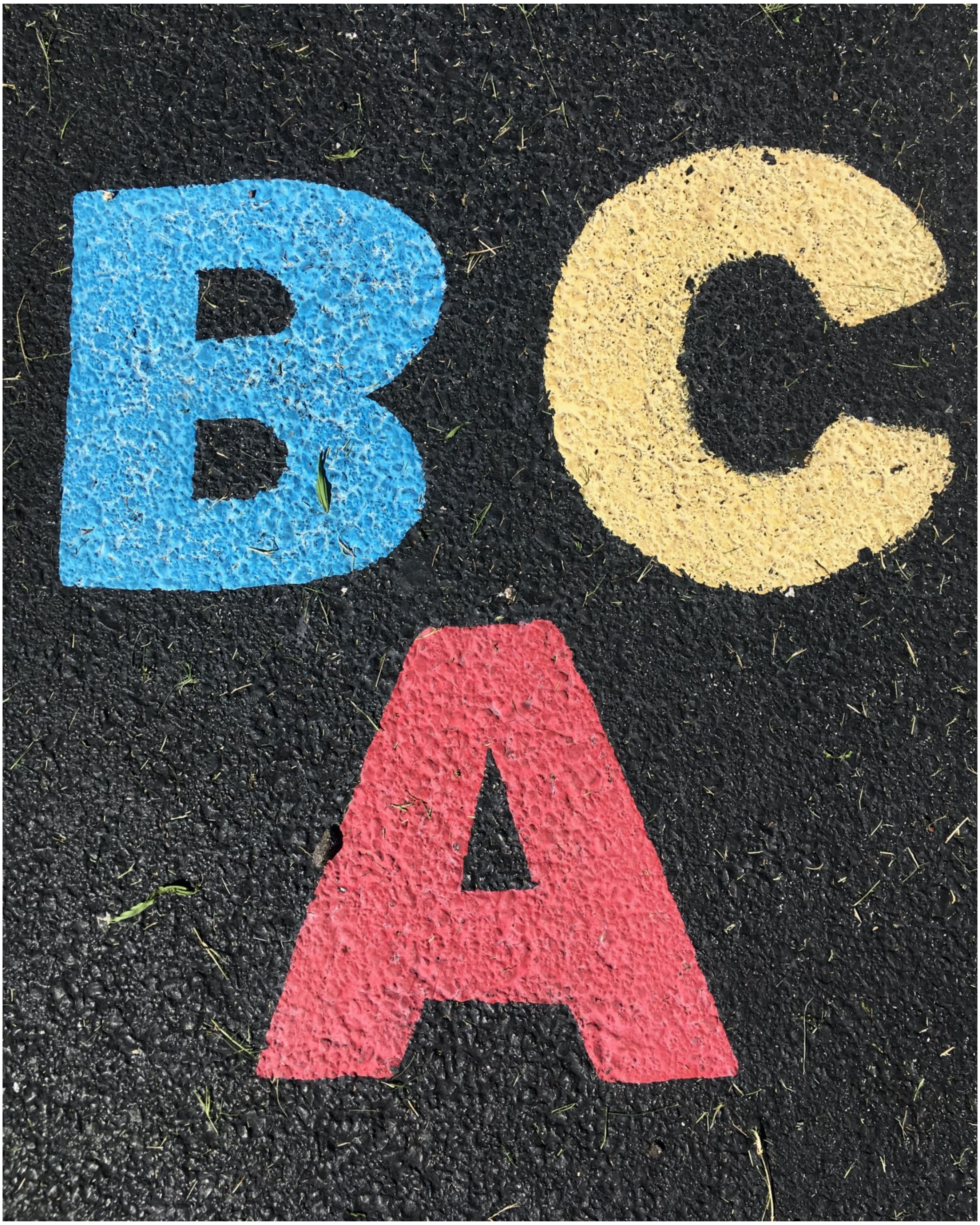
When people create a new programming language, they do so because they want to keep the good features of old languages and fix the bad ones.

In this sense, Guido van Rossum created Python in the late 1980s to improve ABC. The latter was too perfect for a programming language — while its rigidity made it easy to teach, it was hard to use in real life.

In contrast, Python is quite pragmatic. You can see this in the Zen of Python, which reflects the intention that the creators have:

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
[...]
```

Python still kept the good features of ABC: Readability, simplicity, and beginner-friendliness for example. But Python is far more robust and adapted to real life than ABC ever was.



ABC paved the way for Python, which is paving the way for Julia. Photo by David Ballew on Unsplash

In the same sense, the creators of Julia want to keep the good parts of other languages and ditch the bad ones. But Julia is a lot more ambitious: instead of replacing one language, it wants to beat them all.

This is how Julia's creators say it:

```
We are greedy: we want 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.
```

Julia wants to blend all upsides that currently exist, and not trade them off for the downsides in other languages. And even though Julia is a young language, it has already achieved a lot of the goals that the creators set.

What Julia developers are loving

Versatility

Julia can be used for everything from simple machine learning applications to enormous supercomputer simulations. To some extent, Python can do this, too — but Python somehow grew into the job.

In contrast, Julia was built precisely for this stuff. From the bottom up.

Speed

Julia's creators wanted to make a language that is as fast as C — but what they created is even faster. Even though Python has become easier to speed up in recent years, its performance is still a far cry from what Julia can do.

In 2017, Julia even joined the Petaflop Club — the small club of languages who can exceed speeds of one petaflop per second at peak performance. Apart from Julia, only C, C++ and Fortran are in the club right now.

Tiny improvement at each step, great leap as a whole

towardsdatascience.com

Community

With its more than 30 years of age, Python has an enormous and supportive community. There is hardly a Python-related question that you can't get answered within one Google search.

In contrast, the Julia community is pretty tiny. While this means that you might need to dig a bit further to find an answer, you might link up with the same people again and again. And this can turn into programmer-relationships that are beyond value.

Code conversion

You don't even need to know a single Julia-command to code in Julia. Not only can you use Python and C code within Julia. You can even use Julia within Python!

Needless to say, this makes it extremely easy to patch up the weaknesses of your Python code. Or to stay productive while you're still getting to know Julia.

Libraries are still a strong point of Python. Photo by Susan Yin on Unsplash

Libraries

This is one of the strongest points of Python — its zillion well-maintained libraries. Julia doesn't have many libraries, and users have complained that they're not amazingly maintained (yet).

But when you consider that Julia is a very young language with a limited amount of resources, the number of libraries that they already have is pretty impressive. Apart from the fact that Julia's amount of libraries is growing, it can also interface with libraries from C and Fortran to handle plots, for example.

Dynamic and static types

Python is 100% dynamically typed. This means that the program decides at runtime whether a variable is a float or an integer, for example.

While this is extremely beginner-friendly, it also introduces a whole host of possible bugs. This means that you need to test Python code in all possible scenarios — which is quite a dumb task that takes a lot of time.

Since the Julia-creators also wanted it to be easy to learn, Julia fully supports dynamical typing. But in contrast to Python, you can introduce static types if you like — in the way they are present in C or Fortran, for example.

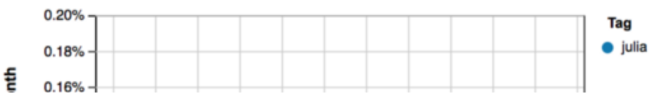
This can save you a ton of time: Instead of finding excuses for not testing your code, you can specify the type wherever it makes sense.

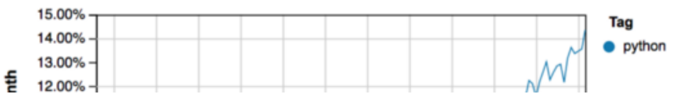
5 Ways Julia Is Better Than Python

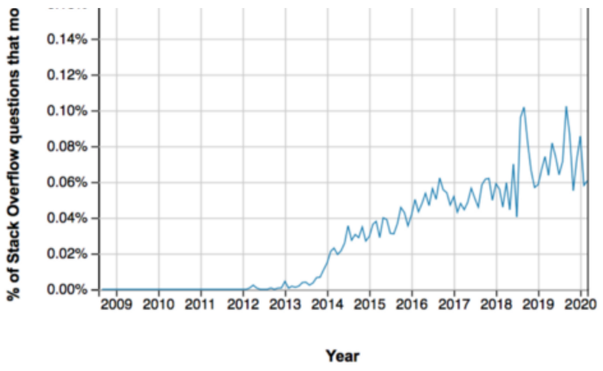
Why Julia is better than Python for DS/ML

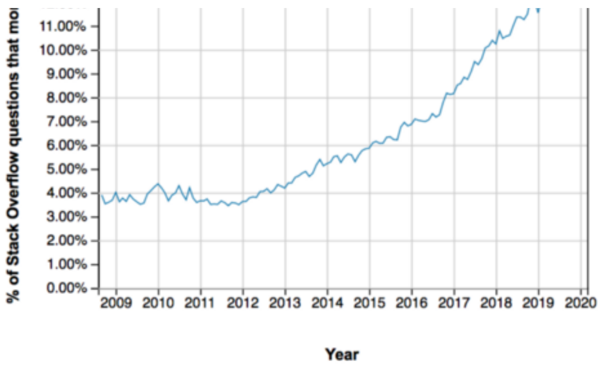
towardsdatascience.com

The data: Invest in things while they're small









Number of questions tagged Julia (left) and Python (right) on StackOverflow.

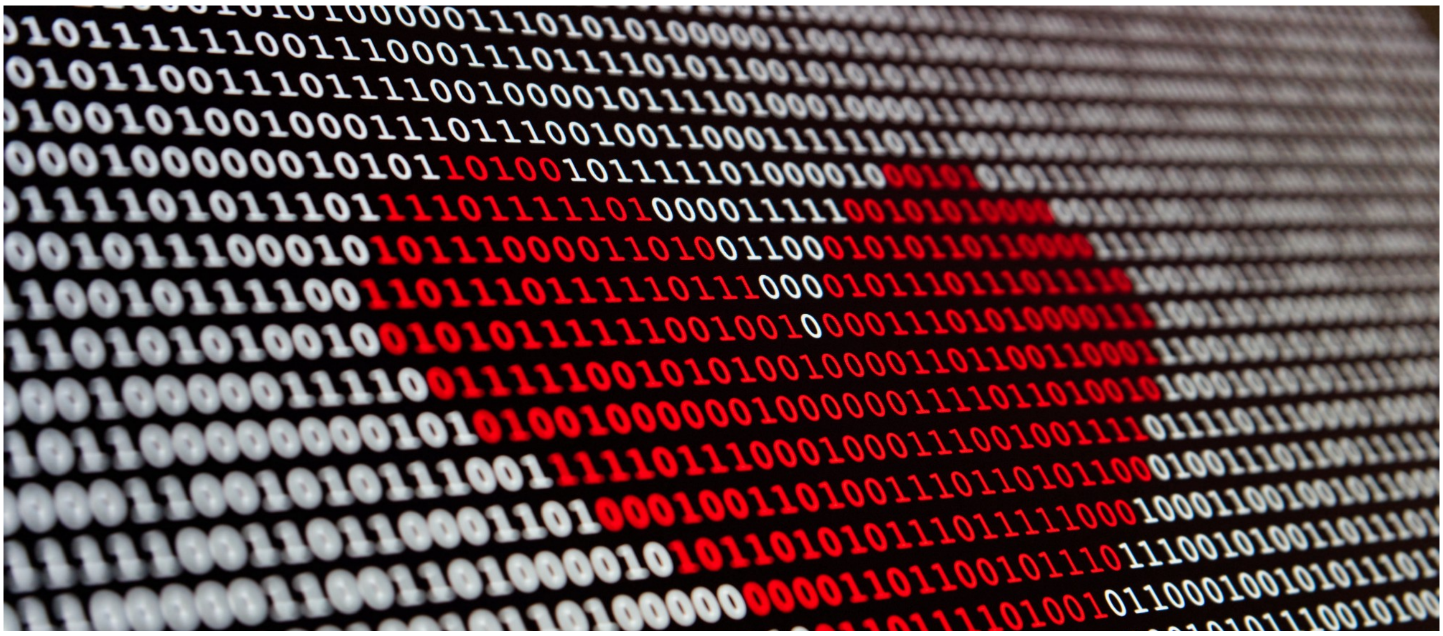
While all these things sound pretty great, it's important to keep in mind that Julia is still tiny compared to Python.

One pretty good metric is the number of questions on StackOverflow: At this point in time, Python is tagged about twenty more often than Julia!

This doesn't mean that Julia is unpopular — rather, it's naturally taking some time to get adopted by programmers.

Think about it — would you really want to write your whole code in a different language? No, you'd rather try a new language in some future project. This creates a time lag that every programming language faces between its release and its adoption.

But if you adopt it now — which is easy because Julia allows an enormous amount of language conversion — you're investing in the future. As more and more people adopt Julia, you'll already have gained enough experience to answer their questions. Also, your code will be more durable as more and more Python code is replaced by Julia.



It's time to show Julia some love. Photo by Alexander Sinn on Unsplash

Bottom line: Do Julia and let it be your edge

Forty years ago, artificial intelligence was nothing but a niche phenomenon. The industry and investors didn't believe in it, and many technologies were clunky and hard to use. But those who learned it back then are the giants of today — those that are so high in demand that their salary matches that of an NFL player.

Similarly, Julia is still very niche now. But when it grows, the big winners will be those who adopted it early.

I'm not saying that you're guaranteed to make a shitload of money in ten years if you adopt Julia now. But you're increasing your chances.

Think about it: Most programmers out there have Python on their CV. And in the next few years, we'll see even more Python programmers on the job market. But if the demand of enterprises for Python slows, the perspectives for Python programmers are going to go down. Slowly at first, but inevitably.

On the other hand, you have a real edge if you can put Julia on your CV. Because let's be honest, what distinguishes you from any other Pythonista out there? Not much. But there won't be that many Julia-programmers out there, even in three years' time.

With Julia-skills, not only are you showing that you have interests beyond the job requirements. You're also demonstrating that you're eager to learn and that you have a broader sense of what it means to be a programmer. In other words, you're fit for the job.

You — and the other Julia programmers — are future rockstars, and you know it. Or, as Julia's creators said it in 2012:

```
Even though we recognize that we are inexcusably greedy, we still
want to have it all. About two and a half years ago, we set out to
create the language of our greed. It's not complete, but it's time
for a 1.0 release — the language we've created is called Julia. It
already delivers on 90% of our ungracious demands, and now it needs
the ungracious demands of others to shape it further. So, if you are
also a greedy, unreasonable, demanding programmer, we want you to
give it a try.
```

Python is still insanely popular. But if you learn Julia now, that could be your golden ticket later on. In this sense: Bye-bye Python. Hello Julia!

Thanks to Ludovic Benistant.

Towards Data Science Programming Programming Languages Python Data Science