The Dark Side of the Python

Karnav Popat

Python is the perfect programming language, isn’t it? It’s reliable, it’s readable, and it has a set of libraries bigger than most operating systems. There’s a reason that Guido van Rossum’s masterpiece is currently not only the most popular language on Stack Overflow, but the most beloved, throughout the community. Unfortunately, the reason it’s beloved is not that it’s the fastest or the most compatible with enterprise systems. The primary advantage of Python, it’s readability and low dev time, is not something that’s very conducive to adoption by anyone other than idealistic developers.

For the same reason that COBOL is still used by hundreds and thousands of legacy high-performance systems, particularly in banking, finance, and such applications where speed and power is everything. In a capitalist and highly-competitive set-up like the software industry, managers and executives are rarely concerned with technical debt or ease of development.

The problem with Python is, it’s second-best at everything. While it’s true that this isn’t a problem specific to Python, it is one that is exacerbated by the fact that Python is just good enough to be great at everything, but really not the best at anything at all. When you want to hammer a nail, you bring out a hammer; and when you want to tighten a bolt, you need a wrench. What you don’t do, is use one screwdriver, to tighten screws, hammer nails, and solder pipes. A good developer needs a complete toolbox, and having one, wide-scoped, general-purpose language that’s not ideal for any task is not a great look.

If you’re looking to build close-to-the-OS or bare metal projects, C and C++ is the way to go. The fact is, Python is slow. Like, really slow. On average, Python needs upto an order of magnitude longer to accomplish a task as compared to a lower-level, lightning-fast language like C. This creates a strong and often problematic trade-off between development time and execution time, a problem that is solved by expanding your toolkit to include specialized languages instead of the one-size-fits-all approach of a Python developer.

If you’re looking for long-term scalability and eventual enterprise-support, your best shot will be Java (or JavaScript in the case of a lot of edge-cases. Though there’s really not much reason to use Java other than that it’s supported everywhere, its scalability makes it a good bit better than Python in its niche, striking the right balance between speed and development speed.

The one thing that Python has always had in its corner is data science. The truth is, Python is no longer the best choice for that, either. Julia was created in 2009 by a four-person team and unveiled to the public in 2012, as a new, high-level abstracted language meant to address the shortcomings in Python and other languages and applications used for scientific computing and data processing. Since then it has firmly become the best choice, if not the most popular, for all manner of data applications, not only because it’s much faster than Python, but also because it’s much better suited and specifically designed for data science.

What can’t so easily be replaced by specialized languages and a varied toolkit is Python’s philosophy. The reason Python sprung up to so much popularity and acclaim with the development community is that it has a unique philosophy, the “Zen” of Python: “Beautiful is better than ugly”, “Explicit is better than implicit”, and 17 other aphorisms that lay down a guide not just to good Python programming but also to clean and functional code in general.

The fact is, Python just isn’t the language of the future that it was billed as (or atleast, it shouldn’t be). What it is, is the first step towards developing a concrete ideology of programming. The best developers would (or should) apply Python’s principles to the faster languages, getting the best, as they say, of both worlds.