10 Myths of Enterprise Python

2016 Update: Whether you enjoy myth busting, Python, or just all enterprise software, you will also likely enjoy Enterprise Software with Python, presented by the author of the article below, and published by O'Reilly.

PayPal enjoys a remarkable amount of linguistic pluralism in its programming culture. In addition to the long-standing popularity of C++ and Java, an increasing number of teams are choosing JavaScript and Scala, and Braintree's acquisition has introduced a sophisticated Ruby community.

One language in particular has both a long history at eBay and PayPal and a growing mindshare among developers: **Python**.

Python has enjoyed many years of grassroots usage and support from developers across eBay. Even before official support from management, technologists of all walks went the extra mile to reap the rewards of developing in Python. I joined PayPal a few years ago, and chose Python to work on internal applications, but I've personally found production PayPal Python code from nearly **15 years ago**.

Today, Python powers **over 50 projects**, including:

- **Features and products**, like RedLaser
- **Operations and infrastructure**, both OpenStack and proprietary
- **Mid-tier services and applications**, like the one used to set PayPal's prices and check customer feature eligibility
- **Monitoring agents and interfaces**, used for several deployment and security use cases
- **Batch jobs for data import**, price adjustment, and more
- And far too many developer tools to count

In the coming series of posts I'll detail the initiatives and technologies that led the eBay/PayPal Python community to grow from just under 25 engineers in 2011 to **over 260** in 2014. For this introductory post, I'll be focusing on the 10 myths I've had to debunk the most in eBay and PayPal's enterprise environments.

Myth #1: Python is a new language

What with all the start-ups using it and kids learning it these days, it's easy to see how this myth still persists. Python is actually over 23 years old, originally released in 1991, 5-years before HTTP 1.0 and 4-years before Java. A now-famous early usage of Python was in 1996: Google's first successful web crawler.

If you're curious about the long history of Python, Guido van Rossum, Python's creator, has taken the care to tell the whole story.

Myth #2: Python is not compiled

While not requiring a separate compiler toolchain like C++, Python is in fact compiled to bytecode, much like Java and many other compiled languages. Further compilation steps, if any, are at the discretion of the runtime, be it CPython, PyPy, Jython/JVM, IronPython/CLR, or some other process virtual machine. See Myth #6 for more info.

The general principle at PayPal and elsewhere is that the compilation status of code should not be relied on for security. It is much more important to secure the runtime environment, as virtually every language has a decompiler, or can be intercepted to dump protected state. See the next myth for even more Python security implications.

Myth #3: Python is not secure

Python's affinity for the lightweight may not make it seem formidable, but the intuition here can be misleading. One central tenet of security is to present as small a target as possible. Big systems are anti-secure, as they tend to overly centralize behaviors, as well as undercut developer comprehension. Python keeps these demons at bay by encouraging simplicity. Furthermore, CPython addresses these issues by being a simple, stable, and easily-auditable virtual machine. In fact, a recent analysis by Coverity Software resulted in CPython receiving their highest quality rating.

Python also features an extensive array of open-source, industry-standard security libraries. At PayPal, where we take security and trust very seriously, we find that a combination of hashlib, PyCrypto, and OpenSSL, via PyOpenSSL and our own custom bindings, cover all of PayPal's diverse security and performance needs.

For these reasons and more, Python has seen some of its fastest adoption at PayPal (and eBay) within the application security group. Here are just a few security-based applications utilizing Python for PayPal's security-first environment:

- Creating security agents for facilitating key rotation and consolidating cryptographic implementations
- Integrating with industry-leading HSM technologies
- Constructing TLS-secured wrapper proxies for less-compliant stacks
- Generating keys and certificates for our internal mutual-authentication schemes
- Developing active vulnerability scanners

Plus, myriad Python-built operations-oriented systems with security implications, such as firewall and connection management. In the future we'll definitely try to put together

a deep dive on PayPal Python security particulars.

Myth #4: Python is a scripting language

Python can indeed be used for scripting, and is one of the forerunners of the domain due to its simple syntax, cross-platform support, and ubiquity among Linux, Macs, and other Unix machines.

In fact, Python may be one of the most flexible technologies among general-use programming languages. To list just a few:

1. Telephony infrastructure (Twilio)
2. Payments systems (PayPal, Balanced Payments)
3. Neuroscience and psychology (many, many, examples)
4. Numerical analysis and engineering (numpy, numba, and many more)
5. Animation (LucasArts, Disney, Dreamworks)
6. Gaming backends (Eve Online, Second Life, Battlefield, and so many others)
7. Email infrastructure (Mailman, Mailgun)
8. Media storage and processing (YouTube, Instagram, Dropbox)
9. Operations and systems management (Rackspace, OpenStack)
10. Natural language processing (NLTK)
11. Machine learning and computer vision (scikit-learn, Orange, SimpleCV)
12. Security and penetration testing (so many and eBay/PayPal
13. Big Data (Disco, Hadoop support)
14. Calendaring (Calendar Server, which powers Apple iCal)
15. Search systems (ITA, Ultraseek, and Google)
16. Internet infrastructure (DNS) (BIND 10)

Not to mention web sites and web services aplenty. In fact, PayPal engineers seem to have a penchant for going on to start Python-based web properties, YouTube and Yelp, for instance. For an even bigger list of Python success stories, check out the official list.

Myth #5: Python is weakly-typed

Python's type system is characterized by strong, dynamic typing. Wikipedia can explain more.

Not that it is a competition, but as a fun fact, Python is more strongly-typed than Java. Java has a split type system for primitives and objects, with `null` lying in a sort of gray area. On the other hand, modern Python has a unified strong type system, where the type of `None` is well-specified. Furthermore, the JVM itself is also dynamically-typed, as it traces its roots back to an implementation of a Smalltalk VM acquired by Sun.

Python's type system is very nice, but for enterprise use there are much bigger concerns at hand.

Myth #6: Python is slow

First, a critical distinction: Python is a programming language, not a runtime. There are several Python implementations:

1. **CPython** is the reference implementation, and also the most widely distributed and used.
2. **Jython** is a mature implementation of Python for usage with the JVM.
3. **IronPython** is Microsoft's Python for the Common Language Runtime, aka .NET.
4. **PyPy** is an up-and-coming implementation of Python, with advanced features such as JIT compilation, incremental garbage collection, and more.

Each runtime has its own performance characteristics, and none of them are slow per se. The more important point here is that it is a mistake to assign performance assessments to a programming languages. Always assess an application runtime, most preferably against a particular use case.

Having cleared that up, here is a small selection of cases where Python has offered significant performance advantages:

1. Using NumPy as an interface to Intel's MKL SIMD
2. PyPy's JIT compilation achieves faster-than-C performance
3. Disqus scales from 250 to 500 million users on the same 100 boxes

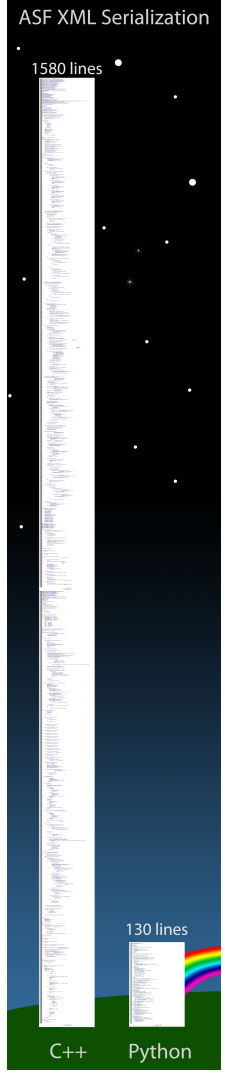
Admittedly these are not the newest examples, just my favorites. It would be easy to get side-tracked into the wide world of high-performance Python and the unique offerings of runtimes. Instead of addressing individual special cases, attention should be drawn to the generalizable impact of developer productivity on end-product performance, especially in an enterprise setting.

Given enough time, a disciplined developer can execute the only proven approach to achieving accurate and performant software:

1. **Engineer** for correct behavior, including the development of respective tests
2. **Profile** and measure performance, identifying bottlenecks
3. **Optimize**, paying proper respect to the test suite and Amdahl's Law, and taking advantage of Python's strong roots in C.

It might sound simple, but even for seasoned engineers, this can be a very time-consuming process. Python was designed from the ground up with developer timelines in mind. In our experience, it's not uncommon for Python projects to undergo three or

more iterations in the time it C++ and Java to do just one. Today, PayPal and eBay have seen multiple success stories wherein Python projects outperformed their C++ and Java counterparts, with less code (see right), all thanks to fast development times enabling careful tailoring and optimization. You know, the fun stuff.



C++ vs Python,. Two languages, one output, as apples to apples as it gets.

Myth #7: Python does not scale

Scale has many definitions, but by any definition, YouTube is a web site at scale. More than 1 billion unique visitors per month, over 100 hours of uploaded video per minute, and going on 20 pecent of peak Internet bandwidth, all with Python as a core technology. Dropbox, Disqus, Eventbrite, Reddit, Twilio, Instagram, Yelp, EVE Online, Second Life, and, yes, eBay and PayPal all have Python scaling stories that prove scale is more than just possible: it's a pattern.

The key to success is simplicity and consistency. CPython, the primary Python virtual machine, maximizes these characteristics, which in turn makes for a very predictable runtime. One would be hard pressed to find Python programmers concerned about garbage collection pauses or application startup time. With strong platform and networking support, Python naturally lends itself to smart horizontal scalability, as manifested in systems like BitTorrent.

Additionally, scaling is all about measurement and iteration. Python is built with profiling and optimization in mind. See Myth #6 for more details on how to vertically scale Python.

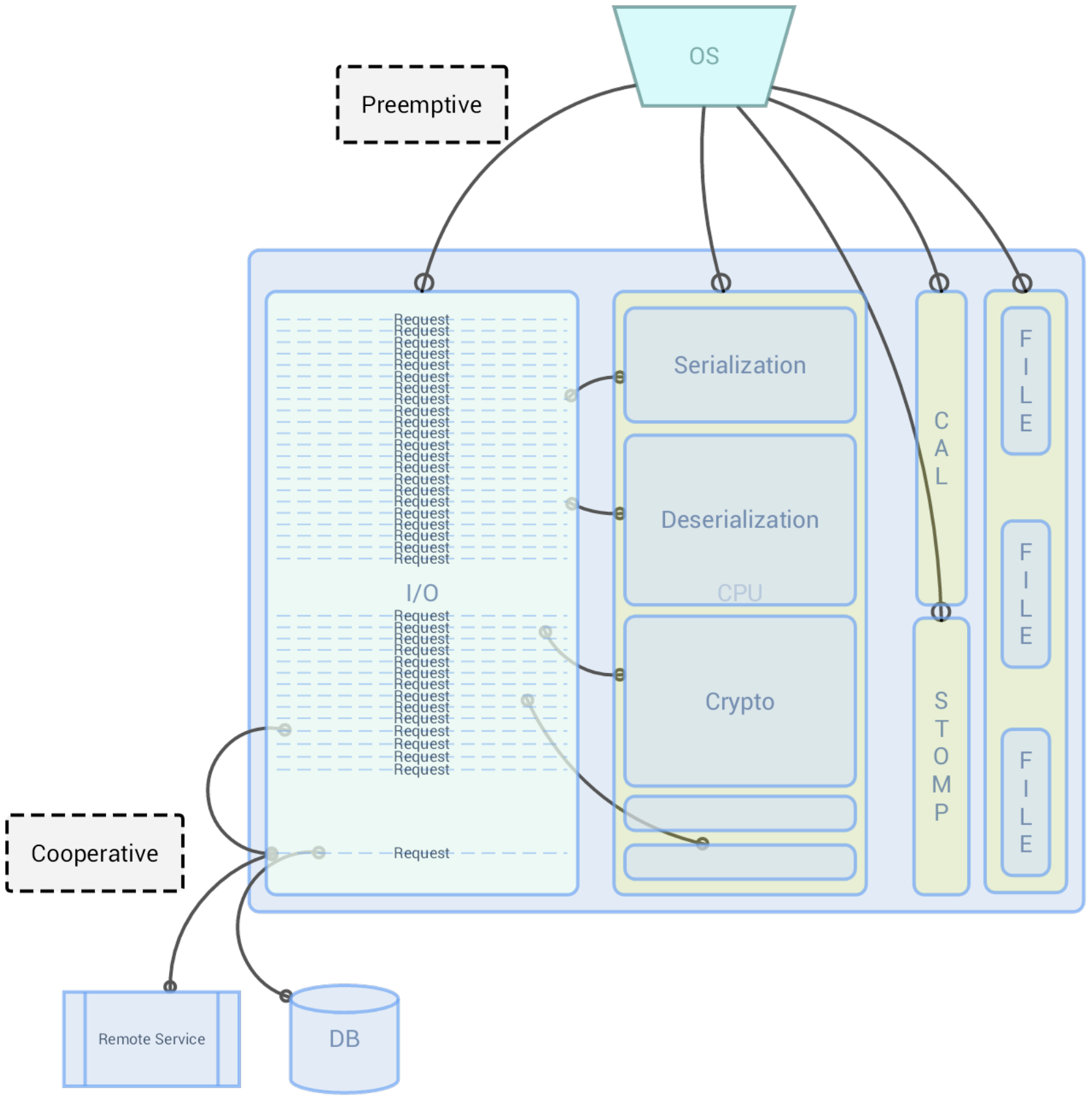
Myth #8: Python lacks good concurrency support

Occasionally debunking performance and scaling myths, and someone tries to get technical, "Python lacks concurrency," or, "What about the GIL?" If dozens of counterexamples are insufficient to bolster one's confidence in Python's ability to scale vertically and horizontally, then an extended explanation of a CPython implementation detail probably won't help, so I'll keep it brief.

Python has great concurrency primitives, including generators, greenlets, Deferreds, and futures. Python has great concurrency frameworks, including eventlet, gevent, and Twisted. Python has had some amazing work put into customizing runtimes for concurrency, including Stackless and PyPy. All of these and more show that there is no shortage of engineers effectively and unapologetically using Python for concurrent programming. Also, all of these are officially support and/or used in enterprise-level production environments. For examples, refer to Myth #7.

The Global Interpreter Lock, or GIL, is a performance optimization for most use cases of Python, and a development ease optimization for virtually all CPython code. The GIL makes it much easier to use OS threads or green threads (greenlets usually), and does not affect using multiple processes. For more information, see this great Q&A on the topic and this overview from the Python docs.

Here at PayPal, a typical service deployment entails multiple machines, with multiple processes, multiple threads, and a very large number of greenlets, amounting to a very robust and scalable concurrent environment (see figure below). In most enterprise environments, parties tends to prefer a fairly high degree of overprovisioning, for general prudence and disaster recovery. Nevertheless, in some cases Python services still see millions of requests per machine per day, handled with ease.



A rough sketch of a single worker within our coroutine-based asynchronous architecture. The outermost box is the process, the next level is threads, and within those threads are green threads. The OS handles preemption between threads, whereas I/O coroutines are cooperative.

Myth #9: Python programmers are scarce

There is some truth to this myth. There are not as many Python web developers as PHP or Java web developers. This is probably mostly due to a combined interaction of industry demand and education, though trends in education suggest that this may change.

That said, Python developers are far from scarce. There are millions worldwide, as evidenced by the dozens of Python conferences, tens of thousands of StackOverflow

questions, and companies like YouTube, Bank of America, and LucasArts/Dreamworks employing Python developers by the hundreds and thousands. At eBay and PayPal we have hundreds of developers who use Python on a regular basis, so what's the trick?

Well, why scavenge when one can create? Python is exceptionally easy to learn, and is a first programming language for children, university students, and professionals alike. At eBay, it only takes one week to show real results for a new Python programmer, and they often really start to shine as quickly as 2-3 months, all made possible by the Internet's rich cache of interactive tutorials, books, documentation, and open-source codebases.

Another important factor to consider is that projects using Python simply do not require as many developers as other projects. As mentioned in Myth #6 and Myth #9, lean, effective teams like Instagram are a common trope in Python projects, and this has certainly been our experience at eBay and PayPal.

Myth #10: Python is not for big projects

Myth #7 discussed running Python projects at scale, but what about *developing* Python projects at scale? As mentioned in Myth #9, most Python projects tend not to be people-hungry. while Instagram reached hundreds of millions of hits a day at the time of their billion dollar acquisition, the whole company was still only a group of a dozen or so people. Dropbox in 2011 only had 70 engineers, and other teams were similarly lean. So, can Python scale to large teams?

Bank of America actually has over 5,000 Python developers, with over 10 million lines of Python in one project alone. JP Morgan underwent a similar transformation. YouTube also has engineers in the thousands and lines of code in the millions. Big products and big teams use Python every day, and while it has excellent modularity and packaging characteristics, beyond a certain point much of the general development scaling advice stays the same. Tooling, strong conventions, and code review are what make big projects a manageable reality.

Luckily, Python starts with a good baseline on those fronts as well. We use PyFlakes and other tools to perform static analysis of Python code before it gets checked in, as well as adhering to PEP8, Python's language-wide base style guide.

Finally, it should be noted that, in addition to the scheduling speedups mentioned in Myth #6 and #7, projects using Python generally require fewer developers, as well. Our most common success story starts with a Java or C++ project slated to take a team of 3-5 developers somewhere between 2-6 *months*, and ends with a single motivated developer completing the project in 2-6 **weeks** (or hours, for that matter).

A miracle for some, but a fact of modern development, and often a necessity for a competitive business.

A clean slate

Mythology can be a fun pastime. Discussions around these myths remain some of the most active and educational, both internally and externally, because implied in every myth is a recognition of Python's strengths. Also, remember that the appearance of these seemingly tedious and troublesome concerns is a sign of steadily growing interest, and with steady influx of interested parties comes the constant job of education. Here's hoping that this post manages to extinguish a flame war and enable a project or two to talk about the real work that can be achieved with Python.

Keep an eye out for future posts where I'll dive deeper into the details touched on in this overview. If you absolutely must have details before then, or have corrections or comments, shoot me an email at mahmoud@paypal.com. Until then, happy coding!