And Now for Something Completely Different

PETER NORTON

Peter works on automating cloud environments. He loves using Python to solve problems. He has contributed to books on Linux and Python, helped with the New York Linux Users Group, and helped to organize past DevOpsDays NYC events. In addition to Python, Peter is slowly improving his knowledge of Rust, Clojure, and maybe other fun things. Even though he is a native New Yorker, he is currently living in and working from home in the northeast of Brazil. pcnorton@rbox.co.

I use Python in my day-to-day work, and I aspire to being able to write about things that I would want to know more about if I were reading this column instead of writing it. I use Python with Saltstack for writing internal APIs, for templating, for writing one-off tools, and for trying out ideas. It's my first choice as a go-to tool for almost anything at this point. I've come to realize that it's the lens that I view my computer and my job through.

Between writing my first column and this one, there was the announcement that Python is changing in a fundamental way. So I feel the need to take this opportunity to reflect on the extraordinary nature of this change: on July 12, 2018, Guido van Rossum elected to step down as the BDFL of the Python language [1].

A lot has been written about the circumstances, and I am not able to add useful commentary or knowledge about Guido's decision to retire from his title and his position in the community. I just want to add my own voice to those who have thanked him for shepherding the language for as long as he has done.

This is also a great chance to give props to all of Python's maintainers who will be guiding the language to its next phase of governance and to discuss what that may mean for those of us who mainly use the language. So while this article will be non-technical, I hope it will at least be informative, interesting, and useful.

Conway's Law

Conway's Law [2] is often invoked when asking how some piece of software developed into its current state. The version at Wikipedia attributed to Melvin Conway says, "organizations which design systems…are constrained to produce designs which are copies of the communication structures of these organizations." If you've ever wondered why a system is written in a byzantine-seeming way including paths in and out of various modules and dependencies that don't appear to make sense to you, it is often because of Conway's Law: the software needed to be worked on is under the constraints imposed by the organization writing it, not just based on the needs of the software.

However, the law also describes systems organized in a clear and sensible, easy-to-follow manner, which often doesn't get noted in describing positive aspects of software.

Python

So let's take a step back and think about Python. The core of the model for changes to Python is most typical of a new programming language: someone wrote it, and that person is in charge. It makes intuitive sense on almost every level. When Python was new, it was mostly simple to defer all decisions to Guido, since he clearly cared and was willing to shoulder the burden. As it developed, in order to accommodate the wishes of its growing community, Python developed a PEP (Python Enhancements Proposal) process for proposing changes to the language, its core modules, the C API, and to make clear what was "standard" (e.g., what other implementations needed to do in order to be considered "an implementation of Python" that could use code written for other Python implementations and not merely be considered "like Python") and what was specific to the C implementation.

The PEP process was there to provide feedback to language maintainers, and Guido was given the jocular title of the BDFL, "Benevolent Dictator for Life." This title has always been conditioned on Guido actually wanting it, and the flexibility given to him and to the language by that nuance has meant that even though Python made that acronym popular, it has become a fairly common term to give to maintainers in programming communities ever since it was coined.

Most of the languages I've seen other people enjoy using have been governed by an involved benevolent leader. Most of these have also been dynamic scripting languages: Ruby, Python, Tcl, and Perl are all somewhat similar as languages, and all follow a similar model: they each have a large audience, core developers, and a single leader whom they flourish/flourished under at some point.

Outside of "scripting languages," some other languages that have the same broad leadership model are Clojure and Scala. When I stopped to think about it, most of the hot languages of the past decade benefitted from having an undisputed leadership and support from the core group of users and maintainers.

In addition to these examples, there is evidence that the BDFL leadership model isn't a critical part of the success of a language—successful languages curated by a company or a committee include Java, C, C++, Haskell, and OCaml, among others. In addition, in a "similar to Python" vein, node.js, for one, is clearly successful, and its governance is managed quite differently. So even though there are many successful models, it's not a stretch to think that the languages that have thrived under a leader have done well solely because of that leadership.

If you use Python as a nontrivial codebase, you've probably considered how Python's organization around a minimal core with many modules matches what enables central language maintainers to do the best job they can. The fun and interesting question is how and whether it has affected the structure and the development of your software.

Going forward, the Python core maintainers and broader committer community have begun the difficult and admirable process of describing what they need in order to feel like they can make good decisions. This means that they are creating documentation on the process and also the context of the decisions that are being made. As you might expect, a new series of PEPs have been produced in order to describe the future of Python governance. Starting at PEP 8000 [3] a series of decisions will be made, and in the end PEP 13 [4] will get filled in with the decisions that are reached.

A deliberate part of the outcome of this will be documentation and data about how other software projects and companies are managed. I understand that they are seeking a common understanding so that everyone participating can make an informed decision towards a common goal of helping Python thrive. This is being acted on as an opportunity to provide future maintainers—themselves and others—with the guidelines and knowledge of how and why they made their decisions. If it's ever necessary to change the governance model again, this will probably make the process easier.

PEP 8002 [5, 6] is absolutely fascinating—the Python community is reaching out to other communities and is asking questions about their governance, which may not be documented clearly enough for outsiders to simply comprehend, and the resulting survey provides material for the Python community to understand where they—where *we*—fit in the broader community of software users. Looking at the Git log of the text of this PEP, I see more and more information being added to it weekly, and each addition is fascinating.

A notable point is that the communities in PEP 8002 are not just other languages. As of this writing, it does include Rust and Typescript, but it also includes Jupyter, Openstack, and Django, as well as Microsoft to add a significantly different and contrasting perspective.

Speculation

I'm now going to put out some very unreliable and probably baseless speculation about what will be done to the language in the future.

First, it is uncontroversial that there is an industry trend that CPU speeds have leveled off. Even though special purpose compute units like GPUs are taking over some workloads, threading that isn't bound to a single CPU is becoming more important, not less. I hope that something new could come to Python to improve its story here, even though it's unlikely considering the current and past state of the language.

In recent 3.x releases, however, the addition of *async* features and libraries emphasize how important it is to have some way of scaling that gets closer to true parallel multithreading. In the long term, could a change in the governance model prioritize multithreaded scheduling?

Another recent change in 3.6+ is type hints and their use for static type checks, even though one of the great things about Python is that the usage of types is very beginner-friendly: that is, flexible and forgiving (as they are in Ruby, Perl, and many other languages!). They are also very expert-friendly! If you know what you are doing, the thinking goes, the lack of compile-time type checking lets you get through prototyping faster.

However, in spite of how friendly Python and similar languages are, it's clear that in many cases strict compile-time checks are a huge benefit. An example of this is the development of HHVM

(HipHop Virtual Machine). In case you're not aware, PHP is also a very flexible dynamically typed language. It is the underpinning of a huge enterprise, and that enterprise created a version of PHP for its own use where they added static type annotations. This feature then made its way to mainstream PHP 7 and above.

I feel that the needs of the business fundamentally altered how they perceived the benefits and difficulties of the language they were using, to the degree that they changed fundamental aspects of that language, trading away some ease of use for what I understand to be a huge benefit. They did this by creating a slightly different language, and while communicating with the maintainer of PHP, and the benefit became a part of mainstream PHP.

If you view this progression as an extension of Conway's Law, that could tell us something about some of the potential directions that Python could go in, and also could perhaps indicate some of the benefits along with the costs. A lot of the benefit of HHVM and PHP derive from the type hints being provided to a JIT, though, and that sounds like something that is closer to **PyPy** than to standard C-Python. But as long as I'm speculating wildly: there you have it.

Changes

I am not trying to predict anything here and now except the obvious: there is potential for huge changes in Python in the long term if the community of maintainers and users come together and agree on the inherent benefits. I am not hoping that anyone try to burn down the amazing system that we have and love! My message is that it will be important to have civil conversations as the maintainers peer into their crystal balls, predict the future, and try to guide the language—but there may be some things that were considered unstoppable, immovable, or invariant that could be called into question now!

It simply seems more possible that there will be a chance to accommodate experiments that haven't been getting done because the opinions of the BDFL were known and would make some suggestions dead on arrival. For the most part, it seems unlikely that the maintainers of Python will want to change the language drastically, but looking at the possibilities with an open mind will benefit everyone greatly.

To follow past, present, and future developments, go to the PEP index at https://www.python.org/dev/peps/, where you will find:

- PEP 8002 describing the governance models of other software projects
- PEP 8010 describing the BDFL governance model
- PEP 8011 describing the council governance model
- PEP 8012 describing the community governance model
- PEP 8013 describing the external council governance model

Ongoing meta-discussion in the community is forming the PEPs above. It's also important to pay attention to the python-committers mailing list (https://mail.python.org/pipermail/python-committers/). At this time there have been discussions about how to time box the discussion so that a decision can be made, though I'm not clear on whether there is an agreement about an actual date just yet.

Decisions are being made in large and small ways constantly, and they always have been. Python sprints (https://python-sprints.github.io/) are places where developers get together and discuss Python in addition to hacking on it. Obviously, Python's past, present, and future are discussed at the sprints and will continue to be discussed there.

Conclusion

For anyone who is considering picking up or becoming involved with Python or a Python-based project, the change in leadership shouldn't discourage you—in fact, the process so far should encourage all of us to understand more about how this language has been governed and how it will be in the future.

This column is being written months before its publication, so when you finally read this, a lot more progress should have been made towards describing how Python's future may be guided, but the process will still be alive and dynamic and in motion. So this is a great opportunity to alert those of you who may not be aware that this is happening, and to invite those of you who may have filed this under "look at how this is going later" to see how things are going now.

References

[1] Guido van Rossum, "Transfer of Power," python-committers list: https://mail.python.org/pipermail/python-committers/2018-July/005664.html.

[2] Conway's Law: https://en.wikipedia.org/wiki/Conway%27s_law.

[3] B. Warsaw, Python Language Governance Proposal Overview: https://www.python.org/dev/peps/pep-8000/.

[4] B. Warsaw, Python Language Governance: https://www.python.org/dev/peps/pep-0013/.

[5] B. Warsaw, L. Langa, A. Pitrou, D. Hellmann, C. Willing, Open Source Governance Survey: https://www.python.org/dev/peps/pep-8002/.

[6] History for PEP 8002: https://github.com/python/peps/commits/master/pep-8002.rst.