How Python 3 Should Have Worked

March 9, 2012

Original link

As a workaday Python developer, it's hard to shake the feeling that the Python 2 to 3 transition isn't working. I get occasional requests to make my libraries work in 3 but it's far from clear how to and when I try to look it up I find all sorts of conflicting advice, none of which sounds very practical.

Indeed, when you see new 3.x versions rolling off the line and no one using them, it's hard to shake the feeling that Python might die in this transition. How will we ever make it across the chasm?

It seems to me that in all the talk about Python 3000 being a new, radical, blue-sky vision of the future, we neglected the proven methods of getting there. In the Python 2 era, we had a clear method for adding language changes:

1. In Python 2.a, support for `from __future__ import new_feature` was added so you could use the new feature if you explicitly declared you wanted it.
2. In Python 2.b, support was added by default so you could just use it without the future declaration.
3. In Python 2.c, warnings begun being issued when you tried to use the old way, explaining you needed to change or your code would stop working.
4. In Python 2.d, it actually did stop working.

It seems to me this process worked pretty well. And I don't see why it couldn't work for the Python 3 transition. This would mean mainly just:

1. A Python 2.x release that added support for `from __future__ import python3`.

Putting this at the top of a file would declare it to be a Python3 file and allow the interpreter to parse it accordingly. (I realize behind the scenes this would mean a lot of work to merge tr 2 and 3 interpreters, but honestly it would always have been better to have a unified codebase to maintain.)

Then if I wanted my Python 2 program to use some 3 modules, I just need to make sure those modules have the import line at the top. If I want to do a new release of my module that works on Python 3, I just need to declare that it only works in Python 2.x and higher and release a version that's been run through

2to3 (with the new import statement). If my project is big, I can even port files to 3 one at a time, leaving the rest as 2 until someone gets around to fixing the rest. Most importantly, I can start porting to Puhon 3 without waiting for all my dependencies to do the same, parallelizing what until now has been a rather serial process.

Users know they can safely upgrade to 2.x since it won't break any existing code. Developers know everyone will eventually upgrade to 2.x so they can drop support for earlier versions. But since 2.x supports code that also runs in 3, they can start writing and releasing code that's future-compatible as well. Eventually the vast major code will work in 3 and users can upgrade to 3. (2.x will issue warnings to the remaining stragglers.) Finally, we can drop support for 2.x and all live happily having crossed the bridge together.

This isn't a radical idea. It's how Python upgrades have always worked. And unless we use it again, I don't see how we're ever going to cross this chasm.