All Things Pythonic

The fate of reduce() in Python 3000

by Guido van van Rossum

March 10, 2005

Summary
I received an email from a compatriot lamenting the planned demise of reduce() and lambda in Python 3000. After a few exchanges I think even he agreed that they can go. Here's a summary, including my reasons for dropping lambda, map() and filter(). I expect tons of disagreement in the feedback, all from ex-Lisp-or-Scheme folks. :-)

ADVERTISEMENT

About 12 years ago, Python aquired lambda, reduce(), filter() and map(), courtesy of (I believe) a Lisp hacker who missed them and submitted working patches. But, despite of the PR value, I think these features should be cut from Python 3000.

Update: lambda, filter and map will stay (the latter two with small changes, returning iterators instead of lists). Only reduce will be removed from the 3.0 standard library. You can import it from functools.

I think dropping filter() and map() is pretty uncontroversial; filter(P, S) is almost always written clearer as [x for x in S if P(x)], and this has the huge advantage that the most common usages involve predicates that are comparisons, e.g. x==42, and defining a lambda for that just requires much more effort for the reader (plus the lambda is slower than the list comprehension). Even more so for map(F, S) which becomes [F(x) for x in S]. Of course, in many cases you'd be able to use generator expressions instead.

Why drop lambda? Most Python users are unfamiliar with Lisp or Scheme, so the name is confusing; also, there is a widespread misunderstanding that lambda can do things that a nested function can't -- I still recall Laura Creighton's Aha!-erlebnis after I showed her there was no difference! Even with a better name, I think having the two choices side-by-side just requires programmers to think about making a choice that's irrelevant for their program; not having the choice streamlines the thought process. Also, once map(), filter() and reduce() are gone, there aren't a whole lot of places where you really need to write very short local functions; Tkinter callbacks come to mind, but I find that more often than not the callbacks should be methods of some state-carrying object anyway (the exception being toy programs).

So now reduce(). This is actually the one I've always hated most, because, apart from a few examples involving + or *, almost every time I see a reduce() call with a non-trivial function argument, I need to grab pen and paper to diagram what's actually being fed into that function before I understand what the reduce() is supposed to do. So in my mind, the applicability of reduce() is pretty much limited to associative operators, and in all other cases it's better to write out the accumulation loop explicitly.

There aren't a whole lot of associative operators. (Those are operators X for which (a X b) X c equals a X (b X c).) I think it's just about limited to +, *, &, |, ^, and shortcut and/or. We already have sum(); I'd happily trade reduce() for product(), so that takes care of the two most common uses. The bitwise operators are rather specialized and we can put fast versions in a library module if there's demand; for shortcut booleans I have the following proposal.

Let's add any() and all() to the standard builtins, defined as follows (but implemented more efficiently):

```
def any(S):
    for x in S:
        if x:
           return True
    return False

def all(S):
    for x in S:
        if not x:
           return False
    return True
```

Combine these with generator expressions, and you can write things like these::

```
any(x > 42 for x in S)     # True if any elements of S are > 42
all(x != 0 for x in S)     # True if all elements if S are nonzero
```

This will mostly give us back the quantifiers from ABC: ABC's `IF EACH x IN s HAS p(x):` becomes `if all(p(x) for x in s):`, its `IF SOME x IN s HAS p(x):` becomes `if any(p(x) for x in s):`, and `IF NO x IN s HAS p(x):` becomes `if not any(p(x) for x in s):`. (Except that in ABC, the variables would be bound and usable inside the IF block; if you need that in Python you'll have to write an explicit for loop and use a break statement.)

I expect that even if you disagree with dropping reduce(), you will agree that adding any() and all() is a good idea -- maybe even for Python 2.5!

Talk Back!

Have an opinion? Readers have already posted 119 comments about this weblog entry. Why not add yours?

RSS Feed

If you'd like to be notified whenever Guido van van Rossum adds a new entry to his weblog, subscribe to his RSS feed.

Digg | del.icio.us | Reddit

About the Blogger

Guido van Rossum is the creator of Python, one of the major programming languages on and off the web. The Python community refers to him as the BDFL (Benevolent Dictator For Life), a title straight from a Monty Python skit. He moved from the Netherlands to the USA in 1995, where he met his

wife. Until July 2003 they lived in the northern Virginia suburbs of Washington, DC with their son Orlijn, who was born in 2001. They then moved to Silicon Valley where Guido now works for Google (spending 50% of his time on Python!).

Sponsored Links

Google Search

( ) Web [✓ Artima.com]