

## Re: Python's simplicity philosophy

**Source:** <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2003-11/4734.html>

---

**From:** Douglas Alan (*nessus\_at\_mit.edu*)

**Date:** 11/26/03

Date: Wed, 26 Nov 2003 17:55:29 -0500

Alex Martelli <aleax@aleax.it> writes:

>> *That would have a larger big O time growth value -- most likely*  
>> *O(n \* log n) vs. O(n), for reasonable implemenations. And while I*

> *If the sequence is carefully randomized, yes. If the sequence has*  
> *any semblance of pre-existing order, the timsort is amazingly good*  
> *at exploiting it, so that in many real-world cases it DOES run as*  
> *O(N).*

C'mon -- to make robust programs you have to assume the worst-case scenario for your data, not the best case. I certainly don't want to write a program that runs quickly most of the time and then for opaque reasons slows to a crawl occasionally. I want it to either run quickly all of the time or run really slowly all of the time (so that I can then figure out what is wrong and fix it).

>> *wouldn't sweat a factor of 2 for a feature of a RAD or scripting*  
>> *language, I would be more concerned about moving to a larger big O*  
>> *value.*

> *Me too! That's why I'd like to make SURE that some benighted soul*  
> *cannot code:*

> *onebigstring = reduce(str.\_\_add\_\_, lotsofstrings)*

The idea of aiming a language at trying to prevent people from doing stupid things is just innane, if you ask me. It's not just inane, it's offensive to my concept of human ability and creativity. Let people make their mistakes, and then let them learn from them. Make a programming language to be a fluid and natural medium for expressing their concepts, not a straight-jacket for citing canon in the orthodox manner.

Furthermore, by your argument, we have to get rid of loops, since an obvious way of appending strings is:

```
result = ""  
for s in strings: result += s
```

>> *Your proposed extension to max() and min() has all the same problems.*

> *Not at all. Maybe you have totally misunderstood "my proposed extension"?*

You are correct, sorry -- I misunderstood your proposed extension. But max() and min() still have all the same problems as reduce(), and so does sum(), since the programmer can provide his own comparison and addition operations in a user-defined class, and therefore, he can make precisely the same mistakes and abuses with sum(), min(), and max() that he can with reduce().

>> *But reasonable programmers don't abuse this generality, and so there*

> *So, you're claiming that ALL people who were defending 'reduce' by posting use cases which DID "abuse this generality" are unreasonable?*

In this small regard, at least, yes. So, here reduce() has granted them the opportunity to learn from their mistakes and become better programmers.

>> *this urge to be stifled. Don't take my word for it -- ask Paul Graham. I believe he was even invited to give the Keynote Address at a recent PyCon.*

> *However, if you agree with Paul Graham's theories on language design, you should be consistent, and use Lisp.*

I don't agree with *\*everything\** that anyone says. But if there were a version of Lisp that were as tuned for scripting as Python is, as portable as Python is, came with as many batteries installed, and had anywhere near as large a user-base, I probably *\*would\** be using Lisp. But there isn't, so I don't.

And Python suits me fine. But if it continues to be bloated with a large number special-purpose features, rather than a small number of general and expressive features, there may come a time when Python will no longer suit me.

> *If you consider Python to be preferable, then there must be some point on which you disagree with him. In my case, I would put "simplicity vs generality" issues as the crux of my own disagreements with Dr. Graham.*

Bloating the language with lots of special-purpose features does not match my idea of simplicity. To the extent that I have succeeded in this world, it has always been by understanding how to simplify things

by moving to a more general model, meaning that I have to memorize and understand less. Memorizing lots of detail is not something I am particularly good at, and is one of the reasons why I dislike Perl so much. Who can remember all that stuff in Perl? Certainly not I. I suppose some people can, but this is why \*I\* prefer Python --- there is much less to remember.

Apparently you would have it so that for every common task you might want to do there is one "right" way to do it that you have to remember, and the language is packed to the brim with special features to support each of these common tasks. That's not simple! That's a nightmare of memorizing special cases, and if that were the future of Python, then that future would be little better than Perl.

>> *Just what is it that I don't grasp again? I think my position is clear: I have no intention to abuse reduce(), so I don't worry myself with ways in which I might be tempted to.*

> *Yet you want reduce to keep accepting ANY callable that takes two arguments as its first argument, differently from APL's / (which does NOT accept arbitrary functions on its left);*

That's because I believe that there should be little distinction between features built into the language and the features that users can add to it themselves. This is one of the primary benefits of object-oriented languages --- they allow the user to add new data types that are as facile to use as the built-in data types.

> *and you claimed that reduce could be removed if add, mul, etc, would accept arbitrary numbers of arguments. This set of stances is not self-consistent.*

Either solution is fine with me. I just don't think that addition should be placed on a pedestal above other operations. This means that you have to remember that addition is different from all the other operations, and then when you want to multiply a bunch of numbers together, or xor them together, for example, you use a different idiom, and if you haven't remembered that addition has been placed on this pedestal, you become frustrated when you can't find the equivalent of sum() for multiplication or xor in the manual.

>> *So, now you \*do\* want multiple obviously right ways to do the same thing?*

> *sum(sequence) is the obviously right way to sum the numbers that are the items of sequence. If that maps to add.reduce(sequence), no problem; nobody in their right mind would claim the latter as "the one obvious way", exactly because it IS quite un-obvious.*

It's quite obvious to me. As is a loop.

comp.lang.python: Re: Python's simplicity philosophy

- > *The point is that the primary meaning of "reduce" is "diminish", and*
- > *when you're summing (positive:–) numbers you are not diminishing*
- > *anything whatsoever*

Of course you are: You are reducing a bunch of numbers down to one number.

- >> *"summary" or "gist" in addition to addition. It also can be confusing*
- >> *by appearing to be just a synonym for "add". Now people might have*
- >> *trouble remember what the difference between sum() and add() is.*

- > *Got any relevant experience teaching Python? I have plenty and I*
- > *have never met ANY case of the "trouble" you mention.*

Yes, I taught a seminar on Python, and I didn't feel it necessary to teach either sum() or reduce(). I taught loops, and I feel confident that by the time a student is ready for sum(), they are ready for reduce().

- >> *In Computer Science, however, "reduce" typically only has one meaning*
- >> *when provided as a function in a language, and programmers might as*
- >> *well learn that sooner than later.*

>

- > *I think you're wrong. "reduce dimensionality of a multi–dimensional*
- > *array by 1 by operating along one axis" is one such meaning, but there*
- > *are many others. For example, the second Google hit for "reduce*
- > *function" gives me:*

- > <http://www.gits.nl/dg/node65.html>

That's a specialized meaning of "reduce" in a specific application domain, not a function in a general–purpose programming.

- > *where 'reduce' applies to rewriting for multi–dot grammars, and*
- > *the 5th hit is*

- > <http://www.dcs.ed.ac.uk/home/stg/NOTES/node31.html>

- > *which uses a much more complicated generalization:*

It still means the same thing that reduce() typically means. They've just generalized it further. Some language might generalize sum() further than you have in Python. That wouldn't mean that it still didn't mean the same thing.

- > *while <http://csdl.computer.org/comp/trans/tp/1993/04/i0364abs.htm>*
- > *deals with "the derivation of general methods for the  $L^2$ /*
- > *approximation of signals by polynomial splines" and defines REDUCE*
- > *as "prefilter and down–sampler" (which is exactly as I might expect*
- > *it to be defined in any language dealing mostly with signal*
- > *processing, of course).*

Again a specialized domain.

- > *Designing an over-general approach, and "fixing it in the docs" by*
- > *telling people not to use 90% of the generality they so obviously*
- > *get, is not a fully satisfactory solution. Add in the caveats about*
- > *not using reduce(str.\_\_add\_\_, manystrings), etc, and any reasonable*
- > *observer would agree that reduce had better be redesigned.*

You have to educate people not to do stupid things with loop and sum too. I can't see this as much of an argument.

- > *Again, I commend APL's approach, also seen with more generality in*
- > *Numeric (in APL you're stuck with the existing operator on the left*
- > *of + -- in Numeric you can, in theory, write your own ufuncs), as*
- > *saner. While not quite as advisable, allowing callables such as*
- > *operator.add to take multiple arguments would afford a similarly*
- > *\_correctly-limited generality\_ effect. reduce + a zillion warnings*
- > *about not using most of its potential is just an unsatisfactory*
- > *combination.*

You hardly need a zillion warnings. A couple examples will suffice.

|>oug