

Re: Python's simplicity philosophy

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2003-12/0627.html>

From: Douglas Alan (*nessus_at_mit.edu*)

Date: 12/05/03

Date: Fri, 05 Dec 2003 02:48:40 -0500

BW Glitch <bwglitch@hotpop.com> writes:

> *Douglas Alan wrote:*

>> *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).*

> *In theory, I'd agree with you Douglas. But IRL, I agree with
> Alex. If I have to choose between two algorithms that do almost the
> same, but one works on an special case (that's very common to my
> range) and the other works in general, I'd go with the special
> case. There is no compelling reason for getting into the trouble of
> a general approach if I can do it correctly with a simpler, special
> case.*

When people assert that

```
reduce(add, seq)
```

is so much harder to use, read, or understand than

```
sum(seq)
```

I find myself incredulous. People are making such claims either because they are sniffing the fumes of their own righteous argument, or because they are living on a different planet from me. On my planet, `reduce()` is trivial to understand and it often comes in handy. I find it worrisome that a number of vocal people seem to be living on another planet (or could use a bit of fresh air), since if they end up having any significant influence on the future of Python, then, from where I am standing, Python will be aimed at aliens. While this may be fine and good for aliens, I really wish to use a language designed for natives of my world.

Reasonable people on my world typically seem to realize that `sum()` is just not useful enough that it belongs being a built-in in a general-purpose programming language that aims for simplicity. This is why `sum()` occurs rather infrequently as a built-in in general purpose programming languages. `sum()`, however, should be in the dictionary as a quintessential example of the word "bloat". If you agree that `sum()` should have been added to the language as a built-in, then you want Python to be a bloated language, whether you think you do, or not. It is arguable whether `reduce()` is useful enough that it belongs as a built-in, but it has many more uses than `sum()`, and therefore, there's a case for `reduce()` being a built-in. `reduce()` may carry the weight of its inclusion, but `sum()` certainly does not.

> *One example. I once wrote a small program for my graphic calculator to
> analyze (rather) simple electrical networks. To make the long story
> short, I had two approaches: implement Gaussian Elimination "purely" (no
> modifications to take into account some nasty problems) or implement it
> with scaled partial pivoting. Sure, the partial pivoting is _much_
> better than no pivoting at all, but for the type of electrical networks
> I was going to analyze, there was no need to. All the fuzz about
> pivoting is to prevent a small value to wreck havoc in the calculations.
> In this *specific* case (electric networks with no dependent sources),
> it was impossible for this situation to ever happen.*

> *Results? Reliable results in the specific range of working, which is
> what I wanted. :D*

You weren't designing a general purpose programming language -- you were designing a tool to meet your own idiosyncratic needs, in which case, you could have it tap dance and read Esperanto, but only on second Tuesdays, if you wanted it to, and no one need second guess you. But that's not the situation that we're talking about.

>>> *[Alex Martelli:] 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.*

> *It's not about restraining someone from doing something. It's about
> making it possible to *read* the "f(.)+" code.*

A language should make it *easy* to write readable code; it should not strive to make it *impossible* to write unreadable code. There's no

way a language could succeed at that second goal anyway, and any serious attempts to accomplish that impossible goal would make a language less expressive, more bloated, and would probably even end up making the typical program less readable as a consequence of the language being less expressive and more complex.

> *Human ability and creativity are not compromised when restrictions are made.*

That would depend on the restrictions that you make. Apparently you would have me not use `reduce()`, which would compromise my ability to code in the clear and readable way that I enjoy.

> *In any case, try to program an MCU (micro-controller unit). Python's restrictions are nothing compared with what you have to deal in a MCU.*

I'm not sure what point you have in mind. I have programmed MCU's, but I certainly didn't do so because I felt that it was a particularly good way to express the software I wished to compose.

```
>> 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
```

> *By your logic, fly swatters should be banned because shotguns are more general.*

That's not my logic at all. In the core of the language there should be neither shotguns nor fly-swatters, but rather elegant, orthogonal tools (like the ability to manufacture interchangeable parts according to specs) that allow one to easily build either shotguns or fly-swatters. Perhaps you will notice, that neither "shotgun" nor "fly-swatter" is a non-compound (i.e., built-in) word in the English language.

> *:S It's a matter of design decisions. Whatever the designer thinks is better, so be it (in this case, GvR).*

Of course it is a matter of design decisions, but that doesn't imply that all the of design decisions are being made correctly.

> *At least, in my CS introductory class, one of the things we learned was that programming languages could be extremely easy to read, but very hard to write and vice versa*

Or they can be made both easy to read *and* easy to write.

```
>> 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().*

> *It might still be abused, but not as much as reduce(). But
> considering the alternative (reduce(), namely), it's *much* better
> because it shifts the problem to someone else. We are consenting
> adults, y'know.*

Yes, we're consenting adults, and we can all learn to use reduce() properly, just as we can all learn to use class signatures properly.

>>> *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.*

> *Then reduce() shouldn't be as general as it is in the first place.*

By that reasoning, you would have to remove all the features from the language that people often use incorrectly. And that would be, ummm, *all* of them.

>> *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.*

> *reduce() ... expressive? LOL. I'll grant you it's (over)general, but
> expressive? Hardly.*

You clearly have something different in mind by the word "expressive", but I have no idea what. I can and do readily express things that I wish to do with reduce().

> *It's not a bad thing, but, as Martelli noted, it is
> overgeneralized. Not everyone understand the concept off the bat (as
> you _love_ to claim) and not everyone find it useful.*

If they don't find it useful, they don't have to use it. Many people clearly like it, which is why it finds itself in many programming languages.

> *What I understand for simplicity is that I should not memorize anything
> at all, if I read the code.*

There's no way that you could make a programming language where you don't have to memorize anything at all. That's ludicrous. To use any programming language you have to spend time and effort to develop some amount of expertise in it -- it's just a question of how much bang you get for your learning and memorization buck.

> *That's one of the things I absolutely hate*
> *about LISP/Scheme.*

You have to remember a hell of a lot more to understand Python code than you do to understand Scheme code. (This is okay, because Python does more than Scheme.) You just haven't put in the same kind of effort into Scheme that you put into Python.

For instance, no one could argue with a straight face that

```
seq[1:]
```

is easier to learn, remember, or read than

```
(tail seq)
```

> *Now, what's so hard about sum()?*

There's nothing hard about sum() -- it's just unnecessary bloat that doesn't do enough to deserve being put into the language core. If we put in sum() in the language core, why not quadratic_formula() and newtons_method(), and play_chess()? I'd use all of those more frequently than I would use sum(). The language core should only have stuff that gives you a lot of bang for the buck. sum() doesn't.

>> *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.*

> *_Then_ let them *build* new classes to use sum(), min(), max(),*
> *etc. This functionality is better suited for a class/object in an*
> *OO approach anyway, *not* a function.*

No, making new classes is a heavy-weight kind of thing, and every class you define in a program also should pay for its own weight. Requiring the programmer to define new classes to do very simple things is a bad idea.

>>> *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.*

> *Have you ever programmed in assembly? It's worth a look...*

Yes, I have. And I've written MCU code for microcontrollers that I designed and built myself out of adders and other TTL chips and lots of little wires. I've even disassembled programs and patched the binary to fix bugs for which I didn't have the source code.

> *(In case someone's wondering, addition is the only operation available
> in many MPU/MCUs. Multiplication is heavily expensive in those that
> support it.)*

If your point is that high-level languages should be like extremely low level languages, then I don't think that you will find many people to agree with that.

>>> *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.*

> *That make sense if you are in a math related area. But for a layperson,
> that is nonsense.*

It wasn't nonsense to me when I learned this in tenth grade -- it made perfect sense. Was I some sort of math expert? Not that I recall, unless you consider understanding algebra and geometry to be out of the realm of the layperson.

>> *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().*

> *<sarcasm>But why didn't you teach reduce()? If it were so simple, it
> was a must in the seminar.</sarcasm>*

I did teach lambda in the seminar and no one seemed to have any difficulty with it. I really couldn't fit in the entire language in one three-hour seminar. There are lots of other things in the language more important than reduce(), but *all* of them are more

important than sum().

- > *Now in a more serious note, reduce() is not an easy concept to*
- > *grasp. That's why many people don't want it in the language. The*
- > *middle land obviously is to reduce the functionality of reduce().*

That's silly. Reducing the functionality of reduce() would make it *harder* to explain, learn, and remember, because it would have to be described by a bunch of special cases. As it is, reduce() is very orthogonal and regular, which translates into conceptual simplicity.

- > *You mention here "general-purpose programming". The other languages*
- > *that I have done something more than a small code snippet (C/C++, Java*
- > *and PHP) lack a reduce()-like function.*

Lots of languages don't provide reduce() and lots of languages do. Few provide sum(). Higher-order functions such as reduce() are problematic in statically typed languages such as C, C++, or Java, which may go a long way towards explaining why none of them include it. Neither C, C++, or Java provide sum() either, though PHP provides array_sum(). But PHP has a huge number of built-in functions, and I don't think that Python wishes to go in that direction.

- >> *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.*

- > *After reading the whole message, how do you plan to do *that*?*

You put a line in the manual saying "As a rule of thumb, reduce() should only be passed functions that do not have side-effects."

- > *The only way to effectively do this is by warning the user*
- > *explicitly *and* limiting the power the functionality has.*

Is this the only way to educate people not to abuse loops?

I didn't think so.

- > *As Martelli said, APL and Numeric does have an equivalent to*
- > *reduce(), but it's limited to a range of functions. Doing so ensures*
- > *that abuse can be contained.*

Doing so would make the language harder to document, remember, and implement.

- >> *You hardly need a zillion warnings. A couple examples will suffice.*
- > *I'd rather have the warnings. It's much better than me saying "How*
- > *funny, this shouldn't do this..." later. Why? Because you can't*
- > *predict what people will actually do. Pretending that most people*
- > *will act like you is insane.*

If the day comes where Python starts telling me that I used reduce() incorrectly (for code that I have that currently works fine), then that is the last day that I would ever use Python.

> *Two last things:*

> *1) Do you have any extensive experience with C/C++? (By extensive, I mean a small-medium to medium project)*

Yes, quite extensive. On large projects, in fact.

> *These languages taught me the value of -Wall. There are way too many bugs lurking in the warnings to just ignore them.*

I don't use C when I can avoid it because I much prefer C++. C++'s strong static type-checking is very helpful in eliminating many bugs before the code will even compile. I find -Wall to be useless in the C++ compiler I typically use because it will complain about all sorts of perfectly okay things. But that's okay, because usually once I get a program I write in C++ to compile, it typically has very few bugs.

> *2) Do you have any experience in the design process?*

Yes, I design and implement software for a living.

|>oug