Sponsored Link •

All Things Pythonic

Language Design Is Not Just Solving Puzzles

by Guido van van Rossum

February 10, 2006

Summary

An incident on python-dev today made me appreciate (again) that there's more to language design than puzzle-solving. A ramble on the nature of Pythonicity, culminating in a comparison of language design to user interface design.

Some people seem to think that language design is just like solving a puzzle. Given a set of requirements they systematically search the solution space for a match, and when they find one, they claim to have the perfect language feature, as if they've solved a Sudoku puzzle. For example, today someone claimed to have solved the problem of the multi-statement lambda.

But such solutions often lack "Pythonicity" -- that elusive trait of a good Python feature. It's impossible to express Pythonicity as a hard constraint. Even the Zen of Python doesn't translate into a simple test of Pythonicity.

In the example above, it's easy to find the Achilles heel of the proposed solution: the double colon, while indeed syntactically unambiguous (one of the "puzzle constraints"), is completely arbitrary and doesn't resemble anything else in Python. A double colon occurs in one other place, but there it's part of the slice syntax, where a[::] is simply a degenerate case of the extended slice notation a[start:stop:step] with start, stop and step all omitted. But that's not analogous at all to the proposal's lambda <args>::<suite>. There's also no analogy to the use of :: in other languages -- in C++ (and Perl) it's a scoping operator.

And still that's not why I rejected this proposal. If the double colon is unpythonic, perhaps a solution could be found that uses a single colon and is still backwards compatible (the other big constraint looming big for Pythonic Puzzle solvers). I actually have one in mind: if there's text after the colon, it's a backwards-compatible expression lambda; if there's a newline, it's a multi-line lambda; the rest of the proposal can remain unchanged. Presto, QED, voila, etcetera.

But I'm rejecting that too, because in the end (and this is where I admit to unintentionally misleading the submitter) I find *any* solution unacceptable that embeds an indentation-based block in the middle of an expression. Since I find alternative syntax for statement grouping (e.g. braces or begin/end keywords) equally unacceptable, this pretty much makes a multi-line lambda an unsolvable puzzle.

And I like it that way! In a sense, the reason I went to considerable length describing the problems of embedding an indented block in an expression (thereby accidentally laying the bait) was that I wanted to convey the sense that the problem was unsolvable. I should have known my geek audience better and expected someone to solve it. :-)

The unspoken, right brain constraint here is that the complexity introduced by a solution to a design problem must be somehow proportional to the problem's importance. In my mind, the inability of lambda to contain a print statement or a while-loop etc. is only a minor flaw; after all instead of a lambda you can just use a named function nested in the current scope.

But the complexity of any proposed solution for this puzzle is immense, to me: it requires the parser (or more precisely, the lexer) to be able to switch back and forth between indent-sensitive and indent-insensitive modes, keeping a stack of previous modes and indentation level. Technically that can all be solved (there's already a stack of indentation levels that could be generalized). But none of that takes away my gut feeling that it is all an elaborate Rube Goldberg contraption.

Mathematicians don't mind these -- a proof is a proof is a proof, no matter whether it contains 2 or 2000 steps, or requires an infinite-dimensional space to prove something about integers. Sometimes, the software equivalent is acceptable as well, based on the theory that the end justifies the means. Some of Google's amazing accomplishments have this nature inside, even though we do our very best to make it appear simple.

And there's the rub: there's no way to make a Rube Goldberg language feature appear simple. Features of a programming language, whether syntactic or semantic, are all part of the language's *user interface*. And a user interface can handle only so much complexity or it becomes unusable. This is also the reason why Python will never have continuations, and even why I'm uninterested in optimizing tail recursion. But that's for another installment.

Talk Back!

Have an opinion? Readers have already posted 27 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!).

This weblog entry is Copyright © 2006 Guido van van Rossum. All rights reserved.

Sponsored Links

Google Search

Web Artima.com

Copyright © 1996-2019 Artima, Inc. All Rights Reserved. - Privacy Policy - Terms of Use