

tour-de-babel

Tour de Babel

[Stevey's Drunken Blog Rants™](#)

This is my whirlwind languages tour — the one I was going to write for the Amazon Developers Journal this month, but couldn't find a way to do it that was... presentable.



For one thing, I lapse occasionally into coarseness and profanity here, so it wasn't appropriate for an official-ish Amazon publication. Instead, I'm stuffing it into my blog, which nobody reads. Except for you. Yep, just you. Hiya.

For another, it's really a work in progress, just a few snippets here and there, not polished at all. Another great reason to make it a blog entry. Doesn't have to be good, or complete. It's just what I've got today. Enjoy!

My whirlwind tour will cover C, C++, Lisp, Java, Perl, (all languages we use at Amazon), Ruby (which I just plain *like*), and Python, which is in there because — well, no sense getting ahead of ourselves, now.

C



 You just have to know C. Why? Because for all practical purposes, every computer in the world you'll ever use is a von Neumann machine, and C is a lightweight, expressive syntax for the von Neumann machine's capabilities. 

The von Neumann architecture is the standard computer architecture you use every day: a CPU, RAM, a disk, a bus. Multi-CPU doesn't really change it that much. The von Neumann machine is a convenient, cost-effective, 1950s realization of a Turing Machine, which is a famous abstract model for performing computations.

There are other kinds of machines too. For instance, there are Lisp Machines, which are convenient 1950s realizations of Lisp, a programming language notation based on the lambda calculus, which is another model for performing computations. Unlike Turing machines, the lambda calculus can be read and written by humans. But the two models are equivalent in power. They both model precisely what computers are capable of.

Lisp Machines aren't very common though, except at garage sales. von Neumann machines won the popularity race. There are various other kinds of computers, such as convenient realizations of neural networks or cellular automata, but they're nowhere as popular either, at least not yet.

So you have to know C.

You also have to know C because it's the language that Unix is written in, and happens also to be the language that Windows and virtually all other operating systems are written in, because they're OSes for von Neumann machines, so what else would you use? Anything significantly different from C is going to be too far removed from the actual capabilities of the hardware to perform well enough, at least for an OS — at least in the last century, which is when they were all written.

You should also know Lisp. You don't have to use it for real work, although it comes in quite handy for lots of GNU applications. In particular, you should learn Scheme, which is a small, pure dialect of Lisp. The GNU version is called [Guile](#).

They teach Scheme at MIT and Berkeley to new students for a semester or two, and the students have absolutely no clue as to why they're learning this weird language. It's a lousy first language, to be honest, and probably a lousy second one too. You should learn it, eventually, and not as your first or second language.



It's hard, though. It's a big jump. It's not sufficient to learn how to write C-like programs in Lisp. That's pointless. C and Lisp stand at opposite ends of the spectrum; they're each great at what the other one sucks at.

If C is the closest language to modeling how computers work, Lisp is the closest to modeling how *computation* works. You don't need to know a lot of Lisp, really. Stick with Scheme, since it's the simplest and cleanest. Other Lisps have grown into big, complex programming environments, just like C++ and Java have, with libraries and tools and stuff. *That*, you don't need to know. But you should be able to write programs in Scheme. If you can make your way through all the exercises in [The Little Schemer](#) and [The Seasoned Schemer](#), you'll know enough, I think.

But you choose a language for day-to-day programming based on its libraries, documentation, tools support, OS integration, resources, and a host of other things that have very little to do with how computers work, and a whole lot to do with how *people* work.

People still write stuff in straight C. Lots of stuff. You should know it!

C++


C++ is the dumbest language on earth, in the very real sense of being the least sentient. It doesn't know about itself. It is not *introspective*. Neither is C, but C isn't "Object-Oriented", and object orientation is in no small measure about making your programs know about themselves. Objects are actors. So OO languages need to have runtime reflection and typing. C++ doesn't, not really, not that you'd ever use.

As for C: it's so easy to write a C compiler that you can build tools on top of C that act like introspection. C++, on the other hand, is essentially un-parseable, so if you want to write smart tools that can, for example, tell you the signatures of your virtual functions, or refactor your code for you, you're stuck using someone else's toolset, since *you* sure as heck aren't gonna parse it. And all the toolsets for parsing C++ out there just plain suck.

C++ is dumb, and you can't write smart systems in a dumb language. Languages shape the world. Dumb languages make for dumb worlds.

All of computing is based on *abstractions*. You build higher-level things on lower-level ones. You don't try to build a city out of molecules. Trying to use too low-level an abstraction gets you into trouble.

We are in trouble.



The biggest thing you can reasonably write in C is an operating system, and they're not very big, not really. They

look big because of all their apps, but kernels are small.

≡ steve yegge

The biggest thing you can write in C++ is... also an operating system. Well, maybe a little bigger. Let's say three times bigger. Or even ten times. But operating system kernels are at most, what, maybe a million lines of code? So I'd argue the biggest system you can reasonably write in C++ is maybe 10 million lines, and then it starts to break down and become this emergent thing that you have no hope of controlling, like the plant in Little Shop of Horrors. Feeeeeeed meeeeeeee...

If you can get it to compile by then, that is.

We have 50 million lines of C++ code. No, it's more than that now. I don't know what it is anymore. It was 50 million last Christmas, nine months ago, and was expanding at 8 million lines a quarter. The expansion rate was increasing as well. Ouch.

Stuff takes forever to do around here. An Amazon engineer once described our code base as "a huge mountain of poop, the biggest mountain you've ever seen, and your job is to crawl into the very center of it, every time you need to fix something."

That was four years ago, folks. That engineer has moved on to greener pastures. Too bad; he was really good.

It's all C++'s fault. Don't argue. It is. We're using the dumbest language in the world. That's kind of meta-dumb, don't you think?

With that said, it is obviously possible to write nice C++ code, by which I mean, code that's mostly C, with some C++ features mixed in tastefully and minimally. But it almost never happens. C++ is a vast playground, and makes you feel smart when you know all of it, so you're always tempted to use all of it. But that's really, really hard to do *well*, because it's such a crap language to begin with. In the end, you just make a mess, even if you're good.

I know, this is Heresy, with a capital-'H'. Whatever. I loved C++ in college, because it's all I knew. When I heard that my languages prof, Craig Chambers, absolutely detested C++, I thought: "Why? I like it just fine." And when I heard that the inventor of STL was on record as saying he hated OOP, I thought he was cracked. How could anyone hate OOP, especially the inventor of STL?

Familiarity breeds contempt in most cases, but not with computer languages. You have to become an expert with a *better* language before you can start to have contempt for the one you're most familiar with.

So if you don't like what I'm saying about about C++, go become an expert at a better language (I recommend Lisp), and then you'll be armed to disagree with me. You won't, though. I'll have tricked you. You won't like C++ anymore, and you might be irked that I tricked you into disliking your ex-favorite language. So maybe you'd better just forget about all this. C++ is great. Really. It's just ducky. Forget what I said about it. It's fine.

Lisp

(I'm betting this next section will astonish you, even if you've been here a while.)


¶ ⓘ Amazon got its start, we had brilliant engineers. I didn't know all of them, but I knew some of them.

Examples? Shai Kaphan. Brilliant. Greg Linden. Brilliant. Eric Benson. Independently famous in his own right, before he ever even came to Amazon. Also brilliant.



They wrote the Obidos webserver. Obidos made Amazon successful. It was only later that poop-making engineers and web devs, frontend folks mostly — schedule-driven people who could make their managers happy by delivering crap fast — it was only later that these people made Obidos bad. Clogged the river, so to speak. But Obidos was a key cornerstone of Amazon's initial success.



The original brilliant guys and gals here only allowed two languages in Amazon's hallowed source repository: C and Lisp. 

Go figure.

They all used Emacs, of course. Hell, Eric Benson was one of the *authors* of XEmacs¹. All of the greatest engineers in the world use Emacs. The world-changer types. Not the great gal in the cube next to you. Not Fred, the amazing guy down the hall. I'm talking about the greatest software developers of our profession, the ones who changed the face of the industry. The James Goslings, the Donald Knuths, the Paul Grahams², the Jamie Zawinskis, the Eric Bensons. Real engineers use Emacs. You have to be way smart to use it well, and it makes you incredibly powerful if you can master it. Go look over Paul Nordstrom's shoulder while he works sometime, if you don't believe me. It's a real eye-opener for someone who's used Visual Blub .NET-like IDEs their whole career.

Emacs is the 100-year editor.

Shel, Eric, Greg, and others like them that I wasn't fortunate enough to work with directly: they didn't allow C++ here, and they didn't allow Perl. (Or Java, for that matter). They knew better.

Now C++, Java and Perl are all we write in. The elders have moved on to greener pastures too.

Shel wrote Mailman in C, and Customer Service wrapped it in Lisp. Emacs-Lisp. You don't know what Mailman is. Not unless you're a longtime Amazon employee, probably non-technical, and you've had to make our customers happy. Not indirectly, because some bullshit feature you wrote broke (because it was in C++) and pissed off our customers, so you had to go and fix it to restore happiness. No, I mean directly; i.e., you had to *talk* to them. Our lovely, illiterate, eloquent, well-meaning, hopeful, confused, helpful, angry, happy customers, the real ones, the ones buying stuff from us, our *customers*. Then you know Mailman.

Mailman was the Customer Service customer-email processing application for ... four, five years? A long time, anyway. It was written in Emacs. Everyone loved it.



People *still* love it. To this very day, I still have to listen to long stories from our non-technical folks about how much they miss Mailman. I'm not shitting you. Last Christmas I was at an Amazon party, some party I have no idea how I got invited to, filled with business people, all of them much prettier and more charming than me and the folks I work with here in the Furnace, the Boiler Room of Amazon. Four young women found out I was in Customer Service, cornered me, and talked for *fifteen minutes* about how much they missed Mailman and Emacs, and how Arizona (the JSP replacement we'd spent years developing) still just wasn't doing it for them.

It was truly surreal. I think they may have spiked the eggnog.

Shel's a genius. Emacs is a genius. Even non-technical people love Emacs. I'm typing in Emacs right now. I'd never voluntarily type anywhere else. It's more than just a productivity boost from having great typing shortcuts and text-editing features found nowhere else on the planet. I type 130 to 140 WPM, error-free, in Emacs, when I'm doing free-form text. I've timed it, with a typing-test Emacs application I wrote. But it's *more* than that.

 Emacs has the Quality Without a Name.



  We retired Mailman. That's because we have the Quality *With* a Name — namely, Suckiness. We suck. We couldn't find anyone who was good enough at Emacs-Lisp to make it work. Nowadays it would be easy; Amazon's filled up with Emacs Lisp hackers, but back then, CS Apps couldn't get the time of day from anyone, so they did what they could with what they had, and there weren't enough Emacs-Lisp folks. For a while, they even had Bob Glickstein on contract, the guy who wrote the O'Reilly "giraffe" book Writing Gnu Emacs Extensions, sitting there writing Gnu Emacs Extensions for Mailman in this little office in the Securities building.

CS Apps was Amazon's first 2-pizza team, you know. They're completely autonomous — then and now. Nobody talks to them, nobody helps them, they build everything themselves. They don't have web devs, they don't have support engineers, they don't have squat, except for rock-solid engineers and a mentoring culture. And that's all they've ever needed.

But they had to retire Mailman. Alas. Alackaday. And I still get to hear about how much people miss it. At parties, even.

I think there may still be more Lisp hackers, per capita, in CS Apps than in any other group at Amazon. Not that they get to use it much, but as Eric Raymond said, even if you don't program in it much, learning Lisp will be a profound experience that will make you a better engineer for the rest of your life.

Religion isn't the opiate of the masses anymore, Karl. IDEs are.

Java



Java is simultaneously the best and the worst thing that has happened to computing in the past 10 years.

On the one hand, Java frees you up from many mundane and error-prone details of C++ coding. No more bounds errors, no more core dumps. Exceptions thrown point you to the exact line of code that erred, and are right 99% of the time. Objects print themselves intelligently on demand. Etc., etc.

On the other hand, in addition to being a language, a virtual machine, a huge set of class libraries, a security model, and a portable bytecode format, Java is a religion. So you can't trust anyone who loves it too much. It's a tricky business to hire good Java programmers.

But Java really has been a big step forward for software engineering in general.

Going from C++ to Java isn't just changing syntax. It's a shocking paradigm shift that takes a while to sink in. It's like suddenly getting your own Executive Assistant. You know how VPs always seem to have all this time to be in meetings, and know how the company's running, and write cool documents, and stuff like that? VPs tend to forget that they're actually TWO full-time people: their self and their EA. Having an EA frees you up to think about the *problems* you need to solve; not having one forces you to spend half your time on mundane tasks. Switching to Java turns you into two programmers — one taking care of all this *stuff* that you no longer have to think much about, and another one focused on the problem domain. It's a staggering difference, and one you can get used to in a real hurry.

As Jamie Zawinski said in his famous "java sucks" article: "First the good stuff: Java doesn't have free(). I have to admit right off that, after that, all else is gravy. That one point makes me able to forgive just about anything else, no matter how egregious. Given this one point, everything else in this document fades nearly into insignificance."

Jamie's article was written in 1997, which in Java years is a *long* time ago, and Java has improved a lot since he wrote it; some of the things he complains about are even fixed now.

Most of them aren't. Java does still kind of suck, as a language. But as Jamie points out, it's "the best language going today, which is to say, it's the marginally acceptable one among the set of complete bagbiting loser languages that we have to work with out here in the real world."

Really, you should [read it](#).

Java is truly wonderful along almost every dimension except for the language itself, which is mostly what JWZ was griping about. But that's a *lot* to gripe about. Libraries can only get you so far if your language sucks. Trust me: you may know many, many things better than I do, but I know that libraries can't really save a sucky language. Five years of assembly-language hell at Geoworks taught me that.

Compared to C++, Java as a language is about even. Well, scratch that, it's a lot better, because it has *strings*, oh man, how can you use a language with lousy string support.




But Java's missing some nice features from C++, such as pass-by-reference(-to-stack-object), typedefs, macros, and operator overloading. Stuff that comes in handy now and again.

Oh, and multiple inheritance, which now I've come to appreciate in my old age. If you think my [Opinionated Elf](#) was a good counterpoint to polymorphism dogma, I've got several brilliant examples of why you *need* multiple inheritance, or at least Ruby-style mixins or automatic delegation. Ask me about the Glowing Sword or Cloak of Thieving sometime. Interfaces suck.

Gosling even said, a few years ago, that if he had to do it all over again, he wouldn't have used interfaces.

But that's *just exactly* what the problem with Java is. When James said that, people were shocked. I could feel the shock waves, could feel the marketing and legal folks at Sun maneuvering to hush him up, brush it off, say it wasn't so.

The problem with Java is that people are blinded by the marketing hype. That's the problem with C++, with Perl, with any language that's popular, and it's a serious one, because languages can't become popular *without* hype. So if the language designer suggests innocently that the language might not have been designed perfectly, it's time to ot the language designer full of horse tranquilizers and shut down the conference.

Languages need hype to survive; I just wish people didn't have to be *blinded* by it.

   steve yegge

I drank the OOP Kool-Aid, I regurgitated the hype myself. When I started at Amazon, I could recite for you all the incantations, psalms, and voodoo chants that I'd learned, all in lieu of intelligence or experience, the ones that told me Multiple Inheritance is Evil 'cuz Everyone Says So, and Operator Overloading Is Evil, and so on. I even vaguely sort of knew why, but not really. Since then I've come to realize that it's not MI that sucks, it's developers who suck. *I* sucked, and I still do, although hopefully less every year.

I had an interview candidate last week tell me that MI is Evil because, for instance, you could make a Human class that multiply-inherits from Head, Arm, Leg, and Torso. He was both right and wrong. That MI situation was evil, sure, but it was all him. Stupid from a distance, evil if he'd made it in through the front door.

Bad developers, who constitute the majority of all developers worldwide, can write bad code in any language you throw at them.

That said, though, MI is no picnic; mixins seem to be a better solution, but *nobody* has solved the problem perfectly yet. But I'll still take Java over C++, even without MI, because I know that no matter how good my intentions are, I will at some point be surrounded by people who don't know how to code, and they will do far less damage with Java than with C++.

Besides, there's way more to Java than the core language. And even the language is evolving, albeit glacially, so there's hope. It's what we should be using at Amazon.

You just have to be careful, because as with any other language, you can easily find people who know a lot about the language environment, and very little about taste, computing, or anything else that's important.

When in doubt, hire Java programmers who are polyglots, who detest large spongy frameworks like J2EE and EJB, and who use Emacs. All good rules of thumb.

Perl

Perl. Where to start?

Perl is an old friend. Perl and I go way back. I started writing Perl stuff in maybe 1995, and it's served me well for nearly a decade.

It's like that old bicycle you've put 100k or 200k miles on, and you'll always have a warm fuzzy spot for it, even though you've since moved on to a more modern bike that weighs 5 lbs and doesn't make your ass hurt so much.

Perl is popular for three reasons:

1. You can *get stuff done* really fast in Perl, which is what really matters, in the end.
2. Perl has the best marketing in the world. You could write a book about how brilliant their marketing is. Sun has marketed Java with money, and Perl is almost keeping up, popularity-wise, purely on the sheer marketing brilliance of Larry Wall and his buddies. Folks at Harvard Business School should study Perl's marketing. It's astonishing.
3. Until roughly, oh, *now*, it had no real competitors.

There are "better" languages than Perl — hell, there are lots of them, if you define "better" as "not being insane". Lisp, Smalltalk, Python, gosh, I could probably name 20 or 30 languages that are "better" than Perl, inasmuch as

they don't look like that Sperm Whale that exploded in the streets of Taiwan over the summer. Whale guts everywhere, covering cars, motorcycles, pedestrians. That's Perl. It's charming, really.



But Perl has many, many things going for it that, until recently, no other language had, and they compensated for its exo-intestinal qualities. You can make all sorts of useful things out of exploded whale, including perfume. It's quite useful. And so is Perl.

While all those other languages (Lisp and Smalltalk being particularly noteworthy offenders) tried to pretend that operating systems don't exist, and that lists (for Lisp) or objects (for Smalltalk) are the be-all, end-all of getting shit done, Perl did exactly the opposite. Larry said: Unix and string processing are the be-all, end-all of getting shit done.

And for many tasks, he was absolutely right. So Perl is better at Unix integration and string processing than any language on the planet, save one, and that one only arrived on the scene recently, from the land of Godzilla. I'll get to that one later.

Sadly, Larry focused soooooo hard on Unix integration and string processing that he totally forgot about lists and objects until it was far too late to implement them properly. In fact, a few key mistakes he made early on in Perl's... well, I hesitate to use the word "design" for whale guts, but let's call it Perl's "lifecycle" — those mistakes made it so hard to do lists and objects correctly that Perl has evolved into a genuine Rube Goldberg machine, at least if you want to use lists or objects.

Lists and objects are pretty farging important too, Larry!

Perl can't do lists because Larry made the tragically stupid decision early on to flatten them automatically. So (1, 2, (3, 4)) magically becomes (1, 2, 3, 4). Not that you ever want it to work this way. But Larry happened to be working on some problem for which it was convenient on that particular day, and Perl's data structures have been pure exploded whale ever since.

Now you can't read a book or tutorial or PowerPoint on Perl without spending at least a third of your time learning about "references", which are Larry's pathetic, broken, Goldbergian fix for his list-flattening insanity. But Perl's marketing is so incredibly good that it makes you feel as if references are the best thing that ever happened to you. You can take a reference to anything! It's fun! Smells good, too!

Perl can't do objects because Larry never reeeeeeally believed in them. Maybe that's OK; I'm still not quite sure if I believe in them either. But then why did he try adding them? Perl's OO is a halfhearted add-on that never caught on with the Perl community. It's just not as inspired as the string-processing or Unix integration stuff.

And of course, Perl has plenty of other crackpot design features. Take its "contexts", for instance, which are a horrid outgrowth of Larry's comical decision to have N variable namespaces, dereferenced by sigils, which he sort of copied from shell-script. In Perl, every operator, every function, every *operation* in the language behaves randomly in one of six different ways, depending on the current "context". There are no rules or heuristics governing how a particular operation will behave in a given context. You just have to commit it all to memory.



Need an example? Here's one: accessing a hash in a scalar context gives you a string containing a fraction whose

numerator is the number of allocated keys, and the denominator is the number of buckets. Whale guts, I'm telling

~~you~~. steve yegge



Like I said, though — until recently, nothing could get the job done like Perl could.

Ruby



Every 15 years or so, languages are replaced with better ones. C was replaced by C++, at least for large-scale application development by people who needed performance but desperately wanted data types too. C++ is being replaced by Java, and Java will doubtless be replaced with something better in seven years — well, seven years after it finishes replacing C++, which evidently hasn't fully happened yet, mostly because Microsoft was able to stall it before it became ubiquitous on the desktop. But for server-side applications, C++ is basically on its way out.

Perl will be gone soon, too. That's because a new language called Ruby has finally been translated into English. Yep, it was invented in Japan, of all places — everyone else was as surprised as you are, since Japan's known for its hardware and manufacturing, but not for its software development. Why, is anyone's guess, but I'm thinking it's the whole typing thing; I just can't imagine they were able to type fast enough before, what with having an alphabet with ten thousand characters in it. But Emacs got multibyte support a few years ago, so I can imagine they're pretty dang fast with it now. (And yes, they use Emacs — in fact Japanese folks did the majority of the Mule [multibyte] support for Emacs, and it's rock-solid.)

Anyway, Ruby stole *everything* good from Perl; in fact, Matz, Ruby's author (Yukihiro Matsumoto, if I recall correctly, but he goes by "Matz"), feels he may have stolen a little too much from Perl, and got some whale guts on his shoes. But only a little.

  steve yegge

For the most part, Ruby took Perl's string processing and Unix integration *as-is*, meaning the syntax is identical, and so right there, before anything else happens, you already have the Best of Perl. And that's a great start, especially if you don't take the Rest of Perl.

But then Matz took the best of list processing from Lisp, and the best of OO from Smalltalk and other languages, and the best of iterators from CLU, and pretty much the best of everything from everyone.

And he somehow made it all work together so well that you don't even notice that it *has* all that stuff. I learned Ruby faster than any other language, out of maybe 30 or 40 total; it took me about 3 days before I was more comfortable using Ruby than I was in Perl, after eight years of Perl hacking. It's so consistent that you start being able to guess how things will work, and you're right most of the time. It's beautiful. And fun. And practical.

If languages are bicycles, then Awk is a pink kiddie bike with a white basket and streamers coming off the handlebars, Perl is a beach cruiser (remember how cool they were? Gosh.) and Ruby is a \$7,500 titanium mountain bike. The leap from Perl to Ruby is as significant as the leap from C++ to Java, but without any of the downsides, because Ruby's essentially a proper superset of Perl's functionality, whereas Java took some things away that people missed, and didn't offer real replacements for them.

I'll write more about Ruby sometime. I need to be inspired first. Read [Why the Lucky Stiff's \(poignant\) guide to Ruby](#). *That* is an inspired piece of work. Really. Read it. It's amazing. I don't understand the kind of mind that could produce it, but it's funny, and poignant, and all about Ruby. Sort of. You'll see.

Python



Well gosh, what about Python, a nice language that has patiently been waiting in the wings for all these years? The Python community has long been the refuge for folks who finally took the red pill and woke up from the Perl Matrix.

Well, they're just like the Smalltalk folks, who waited forever to replace C++, and then Java came along and screwed them royally, and permanently. Oops. Ruby's doing exactly that to Python, right now, today. Practically overnight.

Python *would* have taken over the world, but it has two fatal flaws: the whitespace thing, and the permafrost thing.

The whitespace thing is simply that Python uses indentation to determine block nesting. It forces you to indent everything a certain way, and they do this so that everyone's code will look the same. A surprising number of programmers *hate* this, because it feels to them like their freedom is being taken away; it feels as if Python is trampling their constitutional right to use shotgun formatting and obfuscated one-liners.³

Python's author, Guido Van Rossum, also made some boneheaded technical blunders early on — none quite as e: ⓘ agant as Larry's blunders, but a few were real doozies nonetheless. For instance, Python originally had no lexical scoping. But it didn't have dynamic scoping either, and dynamic scoping may have its share of problems,

but it at least sort of works. Python had NOTHING except for global and local (function) scope, so even though it had a "real" OO system, classes couldn't even access their own damned instance variables. You have to pass a "self" parameter to EVERY instance method and then get to your instance data by accessing it through self. So everything in Python is self, selfself, selfselfself, selfSELFselfSELF_SELF__, and it drives you frigging nuts, even if you don't mind the whitespace thing.

Etc.

But in my opinion, it's really the frost thing that killed Python, and has prevented it from ever achieving its wish to be the premier scripting language, or the premier anything language, for that matter. Heck, people still use Tcl as an embedded interpreter, even though Python is far superior to Tcl in every conceivable way — except, that is, for the frost thing.

What's the frost thing, you ask? Well, I used to have a lot of exceptionally mean stuff written here, but since Python's actually quite pleasant to work with (if you can overlook its warts), I no longer think it's such a great idea to bash on Pythonistas. The "frost thing" is just that they used to have a tendency to be a bit, well, frosty. Why?

Because they were so tired of hearing about the whitespace thing!

I think that's why Python never reached Perl's level of popularity, but maybe I'm just imagining things.

Coda

That was the ADJ article I really wanted to write. Or at least something like it. For some reason, though, my true feelings only seem to come out during insomniac attacks between 3am and 6am. Time for bed! 2 hours 'til my next meeting.

(Published September 2004. Minor updates on 3/28/2006)

Notes

¹ Eric tells me it was actually almost all Jamie Zawinski, when they worked at Lucid together.

² It's been pointed out many times since I wrote this that Paul Graham actually uses vi. Go figure!

³ For the record, I personally don't mind the whitespace thing at all. I think it's silly to dislike Python for that reason. What I'm saying is that a surprising percentage of *other* programmers hate it.

[Back to Stevey's Drunken Blog Rants™](#)



