What are five things you hate about your favorite language? [closed]

Asked 16 years, 1 month ago Modified 9 years, 8 months ago

Viewed 82k times

402

votes

There's been a cluster of Perl-hate on Stack Overflow lately, so I thought I'd bring my "Five things you hate about your favorite language" question to Stack Overflow. Take your favorite language and tell me five things you hate about it. Those might be things that just annoy you, admitted design flaws, recognized performance problems, or any other category. You just have to hate it, and it has to be your favorite language.

Don't compare it to another language, and don't talk about languages that you already hate. Don't talk about the things you like in your favorite language. I just want to hear the things that you hate but tolerate so you can use all of the other stuff, and I want to hear it about the language you wished other people would use.

I ask this whenever someone tries to push their favorite language on me, and sometimes as an interview question. If someone can't find five things to hate about his favorite tool, he doesn't know it well enough to either advocate it or pull in the big dollars using it. He hasn't used it in enough different situations to fully explore it. He's advocating it as a culture or religion, which means that if I don't choose his favorite technology, I'm wrong.

I don't care that much which language you use. Don't want to use a particular language? Then don't. You go through due diligence to make an informed choice and still don't use it? Fine. Sometimes the right answer is "You have a strong programming team with good practices and a lot of experience in Bar. Changing to Foo would be stupid."

This is a good question for code reviews too. People who really know a codebase will have all sorts of suggestions for it, and those who don't know it so well have non-specific complaints. I ask things like "If you could start over on this project, what would you do differently?" In this fantasy land, users and programmers get to complain about anything and everything they don't like. "I want a better interface", "I want to separate the model from the view", "I'd use this module instead of this other one", "I'd rename this set of methods", or whatever they really don't like about the current situation. That's how I get a handle on how much a particular developer knows about the codebase. It's also a clue about how much of the programmer's ego is tied up in what he's telling me.

Hate isn't the only dimension of figuring out how much people know, but I've found it to be a pretty good one. The things that they hate also give me a clue how well they are thinking about the subject.

programming-languages language-agnostic

Share

edited Apr 27, 2015 at 7:21

community wiki
14 revs, 6 users 40%
brian d foy

11 This is a really nice spin on the old "your favorite language" question. Good justification. – Tom Leys Nov 11, 2008 at 23:03

14 I find it interesting that despite SO having a large .NET audience, at the time of this writing there are 24 answers, only one of which (mine) is about .NET or a .NET language. I have no idea what this says about SO or .NET, but it's interesting... – Jon Skeet Nov 11, 2008 at 23:40

22 The first 15 years of programming with C/C++, I hated (in alphabetical order): 1. Pointers 2. Pointers 3. Pointers 4. Pointers 5. Pointers – ileon Mar 10, 2010 at 12:01

4 I wonder how many comments people made about hating their language of choice because they didn't understand how to program in their language of choice.... – Kris.Mitchell May 25, 2010 at 19:12

3 This is a fantastic question. If you're wondering what some language is like, reading 3 different replies about it on this page would be *easily* the best useful-information-for-time-spent you could find. Also a great way to gauge a programmer's experience (and humility) levels if you already know the language. – j_random_hacker Jun 1, 2010 at 13:11

Comments disabled on deleted / locked posts / reviews |

182 Answers

Sorted by: Highest score (default)

1 2 3 4 5 … 7 Next

215 votes

Wow, I'm surprised that **SQL** hasn't made it up here yet. Guess that means nobody loves it :)

- Inconsistent syntax across implementations
- Subtle code differences can have massive performance ramifications for seemingly obscure reasons
- Poor support for text manipulation
- Easy cost of entry but steep learning curve towards mastering the language
- Minimal standardization across the community for best practices, this includes syntax style.

...And a few bonus reasons to hate it, at no extra charge

- the WHERE clause goes last, making it easy to prematurely execute an UPDATE or DELETE, destroying the whole table. Instead, the WHERE should go somewhere up front.
- It's difficult to implement relational division.
- I can set a value to NULL, but I can't test it for equality with NULL. I can check IS NULL, but that just complicates code -- needlessly so, in my opinion.
- Why do we need to completely respecify the formula for a GROUPed column, rather than setting an alias on the column and then GROUP BY the alias (or column index as with SORT)?

Share

edited May 25, 2010 at 22:10

community wiki
3 revs, 3 users 57%
Jeremiah Peschka

---

7 Maybe nobody can learn to love it until they stop thinking of it as a language. :) – Alan Moore Nov 24, 2008 at 4:36

---

4 +1 for everything. And yet people wonder why I'll put up with the headaches of ORM... – James Schek Nov 24, 2008 at 15:27

---

2 @Alan M...isn't that what the L stands for? :) – Kev Dec 9, 2008 at 19:00

29 I can't understand why the syntax for INSERT is so different from UPDATE. And MERGE is incomprehensible. – LaJmOn Jan 14, 2009 at 15:39

3 The necessity of IS NULL should be clear, if you consider that NULL is a third possible result, right after TRUE and FALSE. Since it's meaning is "unknown" you can not tell if something which is unknown matches another thing which is unknown as well. Another example: if NULL equals NULL this would mean that the whole concept of making JOINs would be impossible, since any NULL value could be matched to another NULL value. If you understand this (what also is called ternary logic) than you might understand the reason for introducing the "IS" operator for testing against NULL. – Alex Aug 26, 2010 at 16:58

181 votes

Five things I hate about Java:

- No first-class functions.
- No type inference.
- Lack of sane defaults in eg graphics.
- NullPointerException not containing more information about what is null.
- The proliferation of pointlessly "configurable" frameworks/service provider interfaces/factory classes/dependency injection systems. The configurability is almost never used, DRY is violated egregiously, and code quadruples in size and halves in legibility.

I know, I should check out Scala.

Share

answered Nov 24, 2008 at 16:04

community wiki
Zarkonnen

7 @both: The NPE is shown in the first line of the stack trance. It contains (most of the time) class,java file name, and line number like: "at your.faulty.code.Instance(Intance.java:1234)" Then you just open that file, go to that line and there it is, a variable which has nothing assigned to it. – OscarRyz Jun 25, 2009 at 20:24

35 @Oscar Reyes - Er, we know that. But there may be multiple variables on that line, and the exception message doesn't tell me which one is null. – Zarkonnen Oct 24, 2009 at 10:09

10 Scala has its warts too. However, it is magnificently better than Java. – wheaties Dec 19, 2009 at 21:16

10 +1 for the proliferation of frameworks etc. – Erich Kitzmueller Feb 16, 2010 at 12:52

6 @Valentin, just imagine the fun of the NullPointerException being in a gigantic logfile from a nightly run and you need to figure out what happened... Debugging is not an option. – Thorbjørn Ravn Andersen Feb 21, 2010 at 20:26

158 votes

JavaScript:

1. All the coolest things are insanely complex, but then, all the coolness is also wrapped up in such a small amount of code that you feel stupid for struggling to follow it

2. '+' is an absurd choice of operator for concatenation in a weakly-typed language. Were they *trying* to scare off the noobs?
3. It's a cross-browser compatibility minefield (never mind if it's even turned on or not)
4. It's generally untrusted - associated with scummery such as blocking the back button, pop-ups that never die, etc.
5. It's nearly impossible to debug because there are only a few different error messages and a few different types (Number, String, Object, etc.)

If it wasn't for jQuery, I'd probably still hate it as much as I used to :)

Share

edited Apr 26, 2015 at 18:44

community wiki
4 revs, 4 users 81%
jTresidder

---

15 I agree with mausch. ECMAscript in and of itself is a beautiful and powerful language. It's the pesky browsers (:cough: IE) that muddle its name. – TJ L Feb 6, 2009 at 20:54

32 @Mausch: where does javascript live in the *vast* majority of cases? You're saying the equivalent of "cars don't contribute to global warming, it's driving cars that does it" - true, of course, but missing the point - what else do you do with a car? – jTresidder Feb 7, 2009 at 14:47

20 @Chris: Yes, "+" is a good operator for concatenation in a strongly typed language (like Python). In a weakly typed language (like Javascript or C) it is terrible; it decides (silently!) that 'sum: '+2+3 is not 'sum: 5' but 'sum: 23'. Someone with more Javascript experience can give better examples. – ShreevatsaR Feb 20, 2009 at 0:01

5 Yes, C is weakly typed, compared to, say, Python (e.g. you can assign integers to `char` s, cast anything to anything via void* pointers, etc.) It is *statically* typed instead of *dynamically* typed, and also requires *explicit* typing instead of type inference, but those are unrelated to strong v/s weak typing. [Random examples: Python has implicit dynamic strong typing, Haskell has (optionally explicit) static strong typing, Java has explicit (mostly static) strong typing, C has explicit static (relatively weak) typing.] "Strongly typed" and "weakly typed" are actually not well-defined. – ShreevatsaR Apr 27, 2010 at 21:16

5 @ShreevatsaR The clasical example is: `'3'+'2'='32'` , `'3'-'2'=1` . – Thomas Ahle May 25, 2010 at 18:19

148 votes

PHP:

1) Forces me to make unnecessary variables:

```
$parts = explode('|', $string);
$first = $parts[0];
```

2) An implementation of lambdas so lame it is roughly equivalent to using `eval()` and so hideously wrong I have never used it (see http://www.php.net/create_function).

3) A try/catch system which can only catch about 80% of errors that might occur.

4) Regex support just as lame as lambda support because it has to be written inside regular strings, making one of the most hard-to-learn programming tools about three times as difficult. And PHP is supposed to be an "easy" language?!?!?

5) No way to safely pull stuff out of $_POST without writing it twice or building your own function, or using the '@' operator:

```
$x = isset($_POST['foo']['bar']) ? $_POST['foo']
['bar'] : null;
```

6) Bonus answer: '@'. If you can't be bothered writing your code correctly, just add '@', and too bad for anyone who has to debug your code later.

Share

edited Oct 10, 2012 at 19:04

community wiki
3 revs, 3 users 94%
too much php

44 what about list($first) = explode('|', $string); ? – mlarsen Nov 12, 2008 at 6:56

44 Ideally, I would like to use some_function(explode('|', $string)[0]); – too much php Nov 16, 2008 at 21:10

8 What weird variable scoping? Having everything local and forcing you to declare when you want to use a global is a good idea, it prevent noobs from making functions that just use globals, rather than using arguments and return values like they should do. – scragar Sep 10, 2009 at 9:12

24 you forgot about functions with parameter order changing randomly – dusoft Oct 9, 2009 at 20:43

39 You forgot about verbNoun, verb_noun, noun_verb, nounverb, verbnoun, nounVerb, etc >_> – Warty May 23, 2010 at 18:10

135 votes

C++

- Far too easy to randomly corrupt memory and create almost impossible-to-find bugs (although, Valgrind goes a long way towards fixing this).
- Template error messages.
- When using templates it's easy to end up having to include everything in one file, and then get stupid compile times.
- The standard library is a joke in the modern age (still no threads or network by default?)
- Lots of nasty little bits of C poking through (in particular, all the conversions between short/int/unsigned/etc..)

Share edited Aug 7, 2010 at 10:15

community wiki
3 revs, 3 users 57%
Chris Jefferson

13 I agree with the STL, but I will say what *is* there is pretty good. – Bernard Nov 26, 2008 at 19:20

22 unicode. i do respect ascii's simplicity, but for goodness' sake, we're well into the 21st century now. – wilhelmtell Jan 9, 2009 at 7:57

29 @Kieveli const correctness is actually one of the things i miss most when programming in other languages. particularly the dynamically-typed ones. raii is a big feature i often miss too. – wilhelmtell Jul 13, 2009 at 6:36

6 Most of C++ problems come from being an ISO standard and locked for 10 years. – graham.reeds May 4, 2010 at 9:13

7 +1 "Template error messages." – João Portela May 23, 2010 at 13:10

129 votes

C# / .NET:

- Classes should be sealed by default
- There should be no `lock` statement - instead, you should have specific locking objects, and there should be methods such as `Acquire` which return disposable lock tokens. Corollary: there shouldn't be a monitor for every object.
- `GetHashCode()` and `Equals()` shouldn't be in `System.Object` - not everything's suitable for hashing. Instead, have an `IdentityComparer` which does the

same thing, and keep the `IComparer<T>`, `IComparable<T>`, `IEqualityComparer<T>` and `IEquatable<T>` interfaces for custom comparisons.

- Poor support for immutability
- Poor way of discovering extension methods - it should be a much more conscious decision than just the fact that I'm using a namespace.

Those were off the top of my head - ask me tomorrow and I'll come up with a different 5 :)

Share

answered Nov 11, 2008 at 22:21

community wiki
Jon Skeet

---

22 Sealed by default: inheritance should either be designed into a class (which takes time and limits future options) or prohibited. hashCode/equals: it sucks in Java too. One day I'll write a long blog post about it. Read Effective Java for details of why equals is hard in inheritance chains. – Jon Skeet Nov 11, 2008 at 23:37

---

88 Sealing by default means that you have thought of every possible reason that someone may want to inherit from your class and you don't think any of them make sense. Sorry, but none of us are that smart. – Ed Swangren Nov 12, 2008 at 1:08

---

69 In that case I'm not smart enough for you to derive from my code: because I can't predict what future changes I might make which could break your code. That's a very significant problem, IMO. Sealing the code is more restrictive, but leads

to more implementation freedom and robustness. – Jon Skeet Nov 12, 2008 at 2:38

11 I can't believe no-one mentioned the "goto case" syntax, I hate that one! – Aistina Dec 9, 2008 at 19:07

20 It's a good thing Jon Skeet didn't design C#, or my list would look like "1. classes are sealed by default; 2. locking is too complicated; 3. most objects aren't hashable"! – Gabe May 23, 2010 at 21:27

113 votes

C

- string manipulation.

Having to deal manually with the string buffers is an error-prone pain. Since so much computing is really moving and modifying strings (computers aren't used quite as much for big number-crunching stuff as people thought they'd be way back when), it's really nice to be able to use managed languages or C++'s string objects to deal with these. When I have to do it in straight C, it feels like swimming in quicksand.

Share

answered Nov 11, 2008 at 22:27

community wiki
Michael Burr

50 Agreed. String manipulation is item 1 through 5 of things I hate about C. – BoltBait Nov 11, 2008 at 23:11

1 Just use DJB's safe string library or something. XML manipulation is difficult in most languages, and a lot of programs do XML manipulation, but you don't see many posts saying "Perl is totally broken because it doesn't support DOM nodes as a primitive data type". They use a library.
– Steve Jessop Nov 15, 2008 at 1:19

5 C string manipulation does suck, but as far as language issues go, it's not the worst. – Chris Lutz Feb 19, 2009 at 7:15

3 strcat to concatenate, but wait... does the destination have enough space... ok, must insert if statement to check... but wait, what if my string is on the heap? Ok, must keep a variable around to keep track of size... And this can go on and on and on... – blwy10 Nov 12, 2009 at 14:32

4 We need a thread for five things we *don't* hate about C...
– Longpoke May 23, 2010 at 16:35

94 votes

How about five things I hate about "Things I hate about some language" lists? :D

5- Painting an orange red doesn't make it an apple.

When a language is designed, the designers typically have in mind what it's useful for. Using it for something completely different *can* work, but complaining when it doesn't is just dumb. Take Python. I'm sure either someone has or someone will some day make a utility to create exe's from Python code. Why on God's earth would you *want* to do that? It would be neat—don't get me wrong—but it has no use. So stop complaining about it!

A well-designed project would likely contain code from multiple languages. That's not to say you cannot complete a project with only one language. Some projects may be well within the abilities of whatever language you are using.

4- Are you standing on wooden legs?

The platform can be a large influence of what the language can do. With nowadays garbage collectors, or well even pascals early attempt at "garbage collection", can aid in memory fade (maybe malloc more ram??). Computers are faster and so of course, we expect more out of our languages. And quite frankly, we probably should. However, there is a huge price to pay for the convenience of the compiler to create hash tables or strings or a variety of other concepts. These things may not be inherit to the platform of which they are used. To say they are easy to include to a language just tells me you may not have a leg to stand on.

3- Who's fault is it really?

Bugs. You know. I love bugs. Why do I love bugs. Because it means I get to keep my job. Without bugs, there would be many closed pizza shops. However, users hate bugs. But here is a little splash of cold water. Every bug *is* the programmers fault. Not the language's. A language with such a strict syntax that would significantly reduce how many bugs were possible to generated would be a completely useless language. It's abilities could probably be counted on one hand. You want flexibility or power?

You've got bugs. Why? Because you're not perfect, and you make mistakes. Take a really identifiable example in C:

```
int a[10];
for (int idx = 0; idx < 15; idx++) a[idx] = 10;
```

We all know what that's going to do. However, what maybe some of us don't realize is.. that functionality can be very beneficial. Depending on what you are doing. Buffer overruns are the cost of that functionality. That code above. If I actually released that to the public. That's again.. say it with me.. "My fault". Not C's for allowing me to do it.

2- Shouldn't we put that in the recycle bin?

It's very easy to point at a feature in a language we don't understand because we don't use it often and call it stupid. Complain that it's there etc. Goto's always entertain me. People always complain about goto's being in a language. Yet I bet your last program included a type of goto. If you have ever used a break or a continue, you've used a goto. That's what it is. Granted, it's a "safe" goto, but it is what it is. Goto's have their uses. Whether "implicit" gotos like continue or break are used or explicit gotos (using the actual keyword "goto" for whatever language). Not that language developers are flawless, but typically... if functionality has existed since the dawn of time (for that language). Likely that aspect is a defining quality of that language. Meaning.. it's being used and likely is not hanging around because of backwards compatibility. It's being used today. As in 5 minutes ago. And used properly.

Well.. arguably someone is using it improperly as well, but that relates to #3 on my list.

1. - Everything is an object.

Ok.. this one is really a subset of #2. But this is by far the most annoying complaint I see in hate lists. Not everything is an object. There are a great many of concepts that do not belong or need to be objects. Putting things where they don't belong is just ugly and can decrease efficiency of a program. Sure. Maybe not much depending on the language. This also relates to #5. This means... yes. Global are ok. Functions as apposed to static methods are ok. Combining OO programming with global functions is ok. Now.. that doesn't mean we should all go out and "free" our code from it's object models either. When designing a section of code or a whole project, what happens behind the scenes *should* be considered when putting it together. Not only where that concept lives and many other factors. Why wrap global functions within classes or name space concepts if it serves no purpose? Take static member variables. That greatly amuses me because.. well..Depending on the language and implementation of course, but generally speaking, you just declared a global. Yes, there are some reasons to wrap these non-OO concepts in OO wrappers. One of course being self documenting code. That can make sense. So.. like I say. Don't go out and "free" your code. But any good modern language will have a global concept outside of it's OO modeling. Yes I'm specifically meaning to point out that an OO programming language without a global concept most likely has a serious design flaw. Again though.. depends on the intention and design of the language so I'm not attempting to pick on any specific language and there are far too many to analyze right here. Anywho, Consider where the code should live and be the most effective. Adding a bunch of flare to something which doesn't add functionality or support just wears down the keyboard faster. It doesn't do anybody any good. Well.. unless you like brownie points from the person who probably incorrectly taught you that everything is an object.

In short, programming isn't just mindlessly tapping on the keyboard. There are a lot of design considerations to any project. I know it's cliche, but you have to look at it from every angle. Even with nowadays type-safe languages. You don't just chuck code out and expect it to work well. Sure.. it may work, but it may not be the right way to go about it. Overall, pick the language and format that is best suited for the specific job AND the environment. But *no* language takes away the thought behind it. If you're not thinking.. you're just typing.

Share

edited Aug 23, 2010 at 3:39

community wiki
5 revs, 4 users 64%
unknown

19 Languages aren't perfect, and if you make a list of things you hate about a language, you may get some interesting

comments and ideas. First, it allows others to give you solutions which you didn't know existed (look through the posts, you will see a few things were learned). Second, it constitutes user feedback for the language developers (wouldn't you be interested if your users came up with a list of the 5 things they hate the most about your software?), and third, it's kinda interesting to ponder on the flaws of your tools. – Sylver Sep 10, 2009 at 9:20

4 If you view it at that level not only break and continue are gotos, but loops are gotos (jump the begin of the loop if the condition is met), if is goto (if the condition is not met jump over the block, function calls are goto (jump to the begin of the function and later jump back), ... – helium Jan 3, 2010 at 17:04

17 Creating executable files from source code "has no use"? What? – detly May 12, 2010 at 1:41

4 Perl could create an executable from a Perl file since the late '80's. One thing to distribute is useful. No need to a) install Perl, b) install components of program, c) maybe write a script to set paths and execute it all... Yeah, really useless. – xcramps May 25, 2010 at 19:27

1 But, if you can't create .exe files from the source, windows users won't be able to run it. ;) – Evan Plaice Jun 8, 2010 at 3:40

88 votes

Five things I hate about **Java** (which, presently, is my favorite language) in no particular order.

1. As much as I am a fan of Java Generics, there are a lot of oddities that arise from the way it was designed. As such there a myriad of annoying limitations with generics (some of which are the result of type-erasure).

2. The way Object.clone() and the Cloneable interfaces work is totally broken.
3. Instead of taking the high-road and making everything an object (a.la. SmallTalk), Sun wimped out created two distinct categories of data-types: Objects and primitives. As a result there are now *two* representations for fundamental data types and wierd curiosities such as boxing/unboxing and not being able to put primitives in a Collection.
4. Swing is too complex. Don't get me wrong: there's a lot of cool stuff one can do with Swing but it is a great example of over-engineering.
5. This final complaint is equally the fault of Sun and those whom have written XML libraries for Java. Java XML libraries are way too complicated. In order to simply read in an XML file, I often have to worry about what parser I am using: DOM or SAX? The APIs for each is equally confusing. Native support in the language for **easily** parsing/writing XML would be very nice.
6. java.util.Date sucks. Not only is it unnecessarily complicated but all the **useful** methods have been deprecated (and replaced with others that increase complexity).

Share

edited Aug 7, 2010 at 9:41

community wiki

4 revs, 3 users 76%
Ryan Delucchi

32 You forgot about java.util.Date! – TM. Dec 9, 2008 at 18:59

3 Also: The "Cloneable" interface doesn't have a "clone()" method. This makes The Cloneable interface an Oxymoron. And since clone() returns an Object, type safety is out the window (there does not appear any attempt made to rectify this even after Generics have been introduced in J2SE 5.0). – Ryan Delucchi Jan 7, 2009 at 20:18

2 As long as we're bashing cloneable, might as well include the so called Serializable "interface". Whenever using it I always want to stab myself. – wds Jan 8, 2009 at 15:18

12 Hard to do simple things like open a file and read from it. – Eric Johnson Mar 13, 2009 at 18:31

3 @Ryan clone() does not necessarily need to return "Object". With J2SE 5.0 Java introduced covariant return types, which means you can return any subtype of a base class. So public MyType clone() IS possible! – helpermethod May 4, 2010 at 11:15

73 votes

Ruby has many flaws related to its speed, but I don't hate those. It also has flaws with the community evangelism going overboard, but that doesn't really bother me. These are what I hate:

- Closures (blocks) have 4 different creation syntaxes, and none of them are optimal. The elegant syntax is incomplete and ambiguous with hashes, and the full syntax is ugly.

- The community tends to be against real documentation, favoring ‘read the code’. I find this childish and lazy.
- Metaprogramming abuse, particularly in libraries, makes bugs a nightmare to track down.
- On a related note, pervasive metaprogramming makes a comprehensive IDE difficult, if not impossible, to make.
- The way block passing to functions is done is silly. There is no reason blocks should be passed outside the parameter list, or have odd special syntax to access (yield). I am of the opinion that blocks should have been given a less ambiguous syntax (or hashes could have used different delimiters; perhaps <> rather than {}), and passing as parameters to methods should have been just like all other parameters.

  ```
  object.method(1, {|a| a.bar}, "blah")
  ```

  These oddities, like the block must be the last parameter passed and passing more than one block is different with longer syntax, really annoy me.

Share

edited May 23, 2010 at 11:48

community wiki
3 revs, 3 users 77%
Myrddin Emrys

2 sub-optimal m17n & unicode support although it is getting better. 1.9 remains complicated... – Keltia Dec 20, 2008 at 15:21

37 I thought that metaprogramming abuse is called "idiomatic ruby" :) – Slartibartfast Jan 10, 2009 at 15:27

2 akway: The other two syntaxes are *lambda* and *Proc.new*. – Myrddin Emrys Aug 25, 2009 at 20:49

2 Re documentation, I once heard a talk by someone working at the Pragmatic Programmers publishing house, who said that when the company was founded, they wanted a Ruby book because the only one that was available was in Japanese. So they could have had that book translated and published by their company. But what they did instead what to read the source code :-) The Ruby book was apparently one of the books that launched Pragmatic Programmers. – Arthur Reutenauer Oct 9, 2009 at 21:11

13 I find it interesting that 3 of these are to do with people and not the language itself. Ruby remains the language I hate least. – Toby Hede Nov 25, 2009 at 8:54

72 votes

Perl

- Mixed use of sigils

```
my @array = ( 1, 2, 3 );
my $array = [ 4, 5, 6 ];

my $one  = $array[0]; # not @array[0], you
would get the length instead
my $four = $array->[0]; # definitely not
$array[0]

my( $two,  $three ) = @array[1,2];
my( $five, $six   ) = @$array[1,2]; # coerce to
array first

my $length_a = @array;
my $length_s = @$array;

my $ref_a = \@array;
my $ref_s = $array;
```

- For example **none** of these are the same:

  ```
  $array[0]   # First element of @array
  @array[0]   # Slice of only the First
  element of @array
  %array[0]   # Syntax error
  $array->[0] # First element of an array
  referenced by $array
  @array->[0] # Deprecated first element of
  @array
  %array->[0] # Invalid reference
  $array{0}   # Element of %array referenced
  by string '0'
  @array{0}   # Slice of only one element of
  %array referenced by string '0'
  %array{0}   # Syntax error
  $array->{0} # Element of a hash referenced
  by $array
  @array->{0} # Invalid reference
  %array->{0} # Deprecated Element of %array
  referenced by string '0'
  ```

In `Perl6` it is written:

```
my @array = ( 1, 2, 3 );
my $array = [ 4, 5, 6 ];

my $one  = @array[0];
my $four = $array[0]; # $array.[0]

my( $two,  $three ) = @array[1,2];
my( $five, $six   ) = $array[1,2];
```

```
my $length_a = @array.length;
my $length_s = $array.length;

my $ref_a = @array;
my $ref_s = $array;
```

- Lack of true OO

  ```
  package my_object;
  # fake constructor
  sub new{ bless {}, $_[0] }
  # fake properties/attributes
  sub var_a{
    my $self = shift @_;
    $self->{'var_a'} = $_[0] if @_;
    $self->{'var_a'}
  }
  ```

  In `Perl6` it is written:

  ```
  class Dog is Mammal {
      has $.name = "fido";
      has $.tail is rw;
      has @.legs;
      has $!brain;
      method doit ($a, $b, $c) { ... }
      ...
  }
  ```

- Poorly designed regex features

  ```
  /(?=regexp)/;           # look ahead
  /(?<=fixed-regexp)/;    # look behind
  /(?!regexp)/;           # negative look ahead
  /(?<!fixed-regexp)/;    # negative look behind
  /(?>regexp)/;           # independent sub
  expression
  /(capture)/;            # simple capture
  /(?:don't capture)/;    # non-capturing group
  /(?<name>regexp)/;      # named capture
  /[A-Z]/;                # character class
  ```

```
/[^A-Z]/;                # inverted character
class
# '-' would have to be the first or last
element in
# the character class to include it in the
match
# without escaping it
/(?(condition)yes-regexp)/;
/(?(condition)yes-regexp|no-regexp)/;
/\b\s*\b/;               # almost matches
Perl6's <ws>
/(?{ print "hi\n" })/;   # run perl code
```

In `Perl6` it is written:

```
/ <?before pattern>  /;   # lookahead
/ <?after pattern>   /;   # lookbehind
/ regexp :: pattern  /;   # backtracking
control
/ ( capture )        /;   # simple capture
/ $<name>=[ regexp ] /;   # named capture
/ [ don't capture ]  /;   # non-capturing group
/ <[A..Z]>           /;   # character class
/ <-[A..Z]>          /;   # inverted character
class
# you don't generally use '.' in a character
class anyway
/ <ws>               /;   # Smart whitespace
match
/ { say 'hi' }       /;   # run perl code
```

- Lack of multiple dispatch

```
sub f(   int $i ){ ... }  # err
sub f( float $i ){ ... }  # err
sub f($){ ... } # occasionally useful
```

In `Perl6` it is written:

```
multi sub f( int $i ){ ... }
multi sub f( num $i ){ ... }
multi sub f( $i where $i == 0 ){ ... }
multi sub f(     $i ){ ... } # everything else
```

- Poor Operator overloading

  ```
  package my_object;
  use overload
    '+' => \&add,
    ...
  ;
  ```

  In `Perl6` it is written:

  ```
  multi sub infix:<+> (Us $us, Them $them) |
                      (Them $them, Us $us) { ...
  }
  ```

Share

edited Aug 26, 2010 at 16:14

community wiki
11 revs, 3 users 97%
Brad Gilbert

5 I don't see lack of true OO as being as bad as you make it. Sometimes, it's a saviour, especially when the CPAN module you're using didn't think to expose what you need. And lack of multiple dispatch could be worse: perl could have been strongly typed ;-) – Tanktalus Nov 28, 2008 at 18:57

3 I like that Perl isn't strongly typed, but it would be useful to add in some type information. – Brad Gilbert Dec 10, 2008 at 22:01

13 It seems like you chose to criticize a language that's not your favorite (you should have criticized perl6) – Frew Schmidt Feb 23, 2009 at 4:13

5 What's the point about comparing to perl 6? Are you suggesting that perl 6 fixes your issues, or continues them? – Robert P Feb 23, 2009 at 18:14

2 I doubt I need to say more than: ozonehouse.com/mark/periodic – Arafangion Jun 6, 2010 at 1:06

---

57 votes

I'll do **PHP** as I like it at times and Python will be done way too much.

- No namespace; everything is in a kind of very big namespace which is hell in bigger environments
- Lack of standards when it comes to functions: array functions take a needle as a first argument, haystack as second (see array_search). String functions often take the haystack first, needle second (see strpos). Other functions just use different naming schemes: bin2hex, strtolower, cal_to_jd

  Some functions have weird return values, out of what is normal: This forces you to have a third variable declared out of nowhere while PHP could efficiently interpret an empty array as false with its type juggling. There are near no other functions doing the same.

  ```
  $var = preg_match_all('/regexp/', $str, $ret);
  echo $var; //outputs the number of matches
  print_r($ret); //outputs the matches as an
  array
  ```

- The language (until PHP6) does its best to respect a near-retarded backward compatibility, making it carry bad practices and functions around when not needed (see mysql_escape_string vs. mysql_real_escape_string).

- The language evolved from a templating language to a full-backend one. This means anybody can output anything when they want, and it gets abused. You end up with template engines for a templating language...
- It sucks at importing files. You have 4 different ways to do it (include, include_once, require, require_once), they are all slow, very slow. In fact the whole language is slow. At least, pretty slower than python (even with a framework) and RoR from what I gather.

I still like PHP, though. It's the chainsaw of web development: you want a small to medium site done real fast and be sure anybody can host it (although configurations may differ)? PHP is right there, and it's so ubiquitous it takes only 5 minutes to install a full LAMP or WAMP stack. Well, I'm going back to working with Python now...

Share

edited Nov 17, 2023 at 19:24

community wiki
5 revs, 3 users 91%
I GIVE TERRIBLE ADVICE

4 I suppose point 1 is implemented in 5.3 :) While param ordering is getting better naming is still poor. I agree with the backward compatability though. – Ross Mar 9, 2009 at 21:55

4 Gotta love #4. That's one of the things that bothered me the most all the time, too. – Franz Nov 4, 2009 at 11:41

1 I think, the speed argument is pretty subjective. Speed depends much more on how efficient the code is than on the language itself. Poor PHP code is probably slower than high quality python code but good PHP may also perform better than poor Python. – selfawaresoup May 23, 2010 at 12:02

17 no_really_now_mysql_escape_the_string_im_serious() – Salaryman May 26, 2010 at 12:06

2 namespaces schmamespaces. PHP is on the world wide web so everything should be global – Evan Plaice Jun 8, 2010 at 3:47

50 votes

Here are some things I dislike about Java (which is not my favorite language):

- Generics type erasure (i.e. no reified generics)
- Inability to catch multiple exceptions (of different types) in a single catch block
- Lack of destructors (finalize() is a very poor substitute)
- No support for closures or treating functions as data (anonymous inner classes are a very verbose substitute)
- Checked exceptions in general, or more specifically, making unrecoverable exceptions checked (e.g.

SQLException)

- No language-level support for literal collections
- No type-inference when constructors of generic classes are called, i.e. the type parameter(s) must be repeated on both sides of the '='

Share

edited Sep 1, 2010 at 14:35

community wiki
4 revs
Don

1 @Svish - I think is the point is that you would only use this construct when you don't care what type of Exception you're dealing with. In other words, when you want to handle them all identically – Dónal Jan 31, 2009 at 2:10

3 I wouldn't call a lack of destructors a flaw when the language has a GC, and a GC that's gotten better and better with each release. Destructors were missed in java 1.1.8 but not in java 6 because gc is so vastly improved. – Mike Reedell Apr 10, 2009 at 11:49

7 C# fixes all of these except catching multiple exceptions. Generics are reified, destructors are replaced by using/IDisposable, closures are implemented by anon methods and lambdas, exceptions are unchecked, there are collection literals, and there is 'var' to avoid specifying the constructed type twice. – Daniel Earwicker May 9, 2009 at 22:34

1 Java definitely has closures. An anonymous inner class closes over local final variables in its scope. I agree that anonymous inner classes are not a proper substitute for anonymous

functions, but they *are* closures. – Adam Jaskiewicz Jun 5, 2009 at 17:10

2 Anon inner classes are NOT closures: try creating a visitor callback with something like "sum += current.amount()" in it, where "sum" is a non-final variable from the enclosing scope. Close, but no cigar. – Roboprog Jun 12, 2009 at 23:51

40 votes

C++

1. Template Syntax
2. Diamond Inheritance issues
3. The plethora/lack of standard libraries that modern languages have (though boost comes close).
4. IOStreams
5. The syntax used around IOStreams

Python

1. Spaces are meaningful (sometimes)
2. underscored keywords
3. Limited thread support (at least currently)
4. "self" instead of "this"
5. Spaces are meaningful (sometimes)

Share answered Nov 11, 2008 at 22:26

community wiki

grieve

80 You can refer to "self" as "this" is you really want to (although it might be hard for others to follow). "Self" isn't a keyword, and you can name the variable anything you want. – mipadi Nov 11, 2008 at 23:49

36 there you go, I would actually list the meaningfulness of whitespace (especially indentation) in Python as one of its biggest pluses... ;) – Oliver Giesen Nov 12, 2008 at 1:54

22 "spaces are meaningful" is one of the best features of python!! p.s. try to run this in an interpreter "from **future** import braces" – hasen Dec 5, 2008 at 18:05

4 I disagree with pretty much your whole python list, except thread support. Whitespace is not meaningful, indentation is meaningful; there's a big difference. – Christian Oudard Aug 31, 2009 at 20:10

3 Wow. It's like no one invented a text editor that highlights/shows whitespace/tabs as special chars (What, are you coding in notepad?). Also, if you expand tabs to spaces, please go die in a fire. – Fake Name Aug 7, 2010 at 11:06

37 votes

Objective-C

1) No namespaces, just manual naming conventions - I don't mind the that in terms of class separation, but I do miss being able to import all class definitions in a namespace in a single line (like import com.me.somelibrary.*).

2) Libraries still have some holes in important areas like RegEx support.

3) Property syntax is a bit clumsy, requiring three lines (in two separate files) to declare a property.

4) I like the retain/release model, but it is easier than it should be to release a reference and then accidentally make use of it later.

5) Although not really a language feature, Xcode is so intertwined with use of Objective-C I can't help thinking about that aspect... basically the autocompletion, is very iffy. It's more like a system that rewards you for finding something you want exists, and then presents it as a choice afterwards. But then I suppose I never have liked autocomplete engines.

Share

edited May 28, 2010 at 23:02

community wiki
3 revs, 3 users 79%
Kendall Helmstetter Gelner

2 Agree about the namespaces, prefixing classes with letter codes is dumb. And I would add missing support for real class variables, I don’t like faking them with file statics. – zoul Nov 13, 2008 at 18:05

2 Objective-C properties. Seriously, they are shocking, I can't understand the hype especially seeing how well C# does them. – Justicle Jun 19, 2009 at 3:48

6 Actually I really liked that aspect of Lisp and ObjC - you just need an editor with good brace matching, like Emacs or XCode. I usually type braces in pairs before I type anything in

them, so I don't really run into problems with matching... and XCode can also highlight the region enclosed by a brace just by double-clicking on either containing brace.
– Kendall Helmstetter Gelner Dec 28, 2009 at 17:45

1 @Chris S: Are you saying `YES/NO` for booleans is a bad thing? And more importantly, are you saying Named Parameters are a bad thing?? I can understand bools, but named params are possibly one of ObjC's best features (in terms of readability). – jbrennan May 25, 2010 at 20:32

3 Maybe I'm a masochist, but I like prefixed class names. It makes google and documentation searches crystal clear, there's never any confusion about what kind of string your using if the class is called NSString. – kubi Jun 26, 2010 at 16:48

36 votes

C++

- **Strings.**
  They are not interoperable with platform strings, so you end up using std::vector half of the time. The copy policy (copy on write or deep copy) is not defined, so performance guarantees can not be given for straightforward syntax. Sometimes they rely on STL algorithms that are not very intuitive to use. Too many libraries roll their own which are unfortunately much more comfortable to use. Unless you have to combine them.
- **Variety of string representations**
  Now, this is a little bit of a platform problem - but I still hope it would have been better when a less obstinate standard string class would have been available

earlier. The following string representations I use frequently:

    - generic LPCTSTR,
    - LPC(W)STR allocated by CoTaskMemAlloc,
    - BSTR, _bstr _t
    - (w)string,
    - CString,
    - std::vector
    - a roll-my-own class (*sigh*) that adds range checking and basic operations to a (w)char * buffer of known length

- **Build model.**
  I am sick to death of all the time spent muddling around with who-includes-what, forward declarations, optimizing precompiled headers and includes to keep at least incremental build times bearable, etc. It was great in the eighties, but now? There are so many hurdles to packing up a piece of code so it can be reused that even moms dog gets bored listening to me.
- **Hard to parse**
  This makes external tools especially hard to write, and get right. And today, we C++ guys are lacking mostly in the tool chain. I love my C# reflection and delegates but I can live without them. Without great refactoring, I can't.
- **Threading is too hard**
  Language doesn't even recognize it (by now), and the

freedoms of the compiler - while great - are to painful.

- **Static and on-demand initialization** Technically, I cheat here: this is another puzzle piece in the "wrap up code for reuse": It's a nightmare to get something initialized only when it is needed. The best solution to all other redist problems is throwing everything into headers, this problem says "neeener - you cannot".

Granted, a lot of that is beyond strict language scope, but IMO the entire toolchain needs to be judged and needs to evolve.

Share

edited May 25, 2010 at 18:27

community wiki
3 revs, 3 users 93%
peterchen

Looking documentation on the STL is like looking for manuals on how to build a graphics card from scratch. – aviraldg Mar 3, 2010 at 7:28

Frankly, most of these points sound like you never quite bothered to learn C++ properly... this gets rather obvious in #3, since inclusion guards are something that *every* C++ programmer should know. I'm not sure how to understand Point #1 either, are you confused about `std::string`? maybe reading a good documentation and/or tutorial on `std::vector` (and why you're not supposed to use `std::string` in places where it was never designed for) could clear that up for you. – user350814 Feb 8, 2011 at 23:55

@nebukadnezzar: I found Meyers illuminating on the STL, but it doesn't solve the fundamental problems. Frankly, this sounds like you never had to maintain a large project, you never had to hunt down a circular dependency in a dozens-deep include hierarchy. I know include guards, but why do we have to bother with them? BTW. they don't fix every problem. How "standard" is a `std::string` if I can't use it half of the time? (C++0x at least fixes that, but I'm still stuck with dozens of libraries that use different string representations). – peterchen Feb 9, 2011 at 9:55

`but why do we have to bother with them (inclusion guards)` - because C++ doesn't have modules. `How "standard" is a std::string if I can't use it half of the time?` - I think that depends on the way you use `std::string`. The string class allows you to access the string data as `const char*` via `std::string::c_str`, which already makes `std::string` perfectly compatible with every class/function that also takes `const char*` arguments.
– user350814 Feb 9, 2011 at 15:44

---

because C++ doesn't have modules - exactly my complaint: the build model is antique (I'd just accept any other solution than modules, too). ----- *perfectly compatible* - but perfectly incompatible with many other scenarios (I'd argue C++0x fixing this says I do have a point here.) I'd be happy if std::string had been pervasive enough to have been adopted as THE string class 10 years ago, but it wasn't - the other complaint.
– peterchen Feb 9, 2011 at 15:56

35

votes

JavaScript:

- The `Object` prototype can be modified. Every single object in your program gets new properties, and something probably breaks.

- All objects are hash maps, but it's difficult to safely use them as such. In particular, if one of your keys happens to be `__proto__`, you're in trouble.
- No object closure at function reference time. In fact, no object closure at all -- instead, `this` is set whenever a function is called with object notation or the `new` operator. Results in much confusion, particularly when creating event callbacks, because `this` isn't set to what the programmer expects.
  - Corollary: calling a function *without* object notation or the `new` operator results in `this` being set equal to the global object, resulting in much breakage.
- Addition operator overloaded to also perform string concatenation, despite the two operations being fundamentally different. Results in pain when a value you expect to be a number is in fact a string.
- `==` and `!=` operators perform type coercion. Comparisons between different types involve a list of rules that no mortal can remember in full. This is mitigated by the existence of `===` and `!==` operators.
- Both `null` and `undefined` exist, with subtly different, yet redundant meanings. Why?
- Weird syntax for setting up prototype chains.
- `parseInt(s)` expects a C-style number, so treats values with leading zeroes as octal, etc. You can at least `parseInt(s, 10)` but the default behaviour is confusing.

- No block scope.
- Can declare the same variable more than once.
- Can use a variable without declaring it, in which case it's global and probably breaks your program.
- `with { }`.
- **Really** difficult to document with JavaDoc like tools.

Share

edited Dec 7, 2008 at 2:23

community wiki
2 revs
Daniel Cassidy

3 For `null` and `undefined`: sometimes you really want to know if the variable has been assigned a value or not. Since null is a value, undefined is the only way to tell. Granted, the only time I've found this useful was for creating getter/setter functions. – Zach Jul 11, 2009 at 22:38

1 "if one of your keys happens to be **proto**" -- well, it's a reserved word with special meaning. it's like complaining that you can't use `for` as a variable name. – nickf Jan 2, 2010 at 5:13

5 @nickf: The key to a hash is a string. Strings can have any value including reserved words. In particular the value `"for"` is valid as a hash key. `__proto__` is not a reserved word. Special string values that do not work as expected when used as hash keys violate reasonable expectations about how associative arrays work in any language. They also violate the EcmaScript spec. – Daniel Cassidy Jan 3, 2010 at 23:04

2 Thomas: Newline doesn't always end a statement. Therefore sensible coders terminate every statement with a semicolon to

make the code more clear. – Daniel Cassidy May 26, 2010 at 10:30

2 `newline may or may not end a statement depending on context` is one in my top 5 list – reinierpost Aug 26, 2010 at 16:37

34 votes

Python:

- Lack of static typing
- Default argument handling (specifically the fact that you can *change* the default argument for future callers!)
- Too many required underscores (constructors must be called `__init__`)
- Lack of proper private members and functions (convention just says that most things that start with underscore are private, except for all the stuff like `__getattr__` that isn't)
- Funny syntax for `print`ing to a file (but they're fixing that in Python 3)

Share

edited May 23, 2017 at 11:33

community wiki
3 revs
Greg Hewgill

10 What I'd like is an *option* to use static types. – Greg Hewgill Nov 12, 2008 at 10:06

4 BTW: **init** isn't really the constructor, the object was already created, when you enter there (guess what self is...). The constructor is acutally **new** where you get access to the class to be instantiated. – André Nov 26, 2008 at 19:38

90 If you prefer static typing, why is Python your favourite language? – finnw Dec 7, 2008 at 18:09

9 finnw: Static typing is great for some kinds of programs, and not really needed for other types. I usually don't mind the lack of static typing, but when you need it, it's *really* nice to have at least the option. – Greg Hewgill Dec 7, 2008 at 18:26

8 I would say that lack of static typing is a feature, not missing functionality... – arnorhs May 25, 2010 at 17:56

32 votes

C#

- I wish I could `switch()` on any type, and that `case` could be any expression.
- Can't use object initializer syntax with 'readonly' fields / `private set` autoprops. Generally, I want language help with making immutable types.
- Use of `{}` for **namespace** and **class** and **method** and **property/indexer blocks** and **multi-statement blocks** and **array initializers**. Makes it hard to figure out where you are when they're far apart or mismatched.

- I hate writing `(from x in y ... select).Z()`. I don't want to have to fall back to method call syntax because the query syntax is missing something.
- I want a `do` clause on query syntax, which is like `foreach`. But it's not really a query then.

I'm really reaching here. I think C# is fantastic, and it's hard to find much that's broken.

Share

edited Jun 3, 2010 at 18:18

community wiki
2 revs
Jay Bazuzi

14 +1 for switch on any type – ɹəɯɔoɹ Jul 12, 2009 at 20:48

+1 for switch issues, and {} issues, which I hadn't really thought about until now – Maslow Aug 21, 2009 at 21:27

I hate {}. They look too much like (). Mismatching has never been much of an issue to me because I always put them at the same level unless they are basically one-liners.
– Loren Pechtel Jan 2, 2010 at 5:46

2 +1 for the linq query. Especially when you only want one object returned. Instead of (from x in y select).first(), why not a (from x in y select top 1) or something to fit closer to actual sql syntax.
– AdmSteck May 12, 2010 at 19:43

if you wish you could switch() on any type, and that case could be any expression check out F# pattern matching. c-sharpcorner.com/UploadFile/mgold/… – gradbot May 26, 2010 at 3:28

25 votes

PHP

1. No debugging features if you don't control the server, and even then they kinda suck
2. The extreme amount of bad PHP code floating around gives all PHP programmers a bad name
3. Inconsistent function naming
4. Inability to have a static typed variable if I want one (I'm a big fan of dynamic typing 90% of the time)
5. REGISTER_GLOBALS is the devil

Share answered Nov 13, 2008 at 14:37

community wiki
MattBelanger

25 REGISTER_GLOBALS once ate my dog :(– Pim Jager Nov 15, 2008 at 11:08

2 1: I recommend xdebug and a GUI client such as MacGDBp. That really eases some of the pain... I agree on the other points. – Jonas Due Vesterheden Nov 15, 2008 at 11:54

5 #2: Oh god, don't get me started on that. I always have to defend myself as a PHP developer against people who only have seen the mess that many people create with PHP. – selfawaresoup May 23, 2010 at 12:04

1 +1 for #2 I've spent far too much time defending myself as a PHP developer. – UnkwnTech May 25, 2010 at 8:20

+1 for #2 -- results in bad salary too :(– Shiki May 25, 2010 at 18:44

25 votes

C (OK, it's not my favorite, but it hadn't been done yet.)

- Socket library syntax.
- No function overloading.
- C-style strings.
- Buffer overruns.
- Cryptic syntax. I don't know how many times I've looked up stuff like atoi, slapped my forehead, and shouted "Of course!"

EDIT: I could probably come up with more if I resorted to more library code (like I did with sockets, but those are particularly bad), but I already felt like I was cheating for picking on C. So many languages exist only to take the good parts of C and replace the bad that it's kind of like beating a dead horse.

Share

edited Mar 22, 2010 at 17:31

community wiki
4 revs
Bill the Lizard

22 What socket syntax? C has no concept of sockets. – Ferruccio Nov 11, 2008 at 22:47

3 Oh, c'mon! You can come up with five. Doesn't pointer arithmetic just suck? :) – brian d foy Nov 11, 2008 at 23:02

8 +1 I laughed at "C-style strings." And @brain_d_foy: pointer arithmetic only sucks if you don't understand it. – Chris Lutz Feb 19, 2009 at 7:07

1 @Chris Luts: Even back when I was learning plain C (before I knew C++ or any other OO language) I just knew there was something wrong about char arrays. :) – Bill the Lizard Feb 19, 2009 at 13:56

2 pointer arithmetic is a power saw - very efficient, but you risk taking your whole leg – Thorbjørn Ravn Andersen Dec 23, 2009 at 14:33

Common Lisp:

24

votes

1. Keywords are often too wordy.
2. Library support is pitiful.
3. Doesn't work well in OSes that want to handle memory more strictly.
4. Doesn't have good facilities for interacting with the OS.
5. The "loop" facility is not well defined, and sure doesn't look Lispy.

Share

answered Nov 11, 2008 at 22:50

community wiki
David Thornley

2 'loop' might not be lispy, but what's poorly defined about it? – Daniel Cassidy Dec 7, 2008 at 2:08

2 I haven't read the standard myself, I'm mostly going on Paul Graham's "On Lisp". He says the standard is mostly examples, and doesn't define corner cases at all well. – David Thornley Jun 25, 2009 at 20:19

3 don't you mean keywords-are-too-wordy ? – GClaramunt May 25, 2010 at 18:26

I agree that it isn't "lispy," but CLtLv2 spends a lot of time on it. I just think it was designed to do way too much. sunsite.univie.ac.at/textbooks/cltl/clm/… – Hans Van Slooten May 25, 2010 at 18:32

In addition to "loop", "format" is also not very Lisplike. I hate "format" and "loop" both even though Lisp is my favorite language. – Paul Reiners May 25, 2010 at 20:12

24 votes

BrainF*ck

- Your highlight is that you're *Turing complete*?! I can do more in Perl regular expressions!
- Lack of objects. C'mon, people! It's like, *hello*...
- No networking libraries. All I want is to scrape a web page, GOSH.
- No first-class functions. Congratulations — you get to commiserate with your Java friends.
- An infinite tape for storage and nothing else. This is so anally pretentious that we might as well be writing Lisp.

Share

edited Mar 4, 2011 at 9:03

community wiki
3 revs, 2 users 92%
a paid nerd

6 There's no namespace or dynamic module support. How can we be expected to write chemical plant control systems without such basics? – Donal Fellows Jun 1, 2010 at 19:42

No syntactic sugar, such as >10 (move 10 times), 0 (insert zero), +5(add 5). – Squall Apr 5, 2011 at 15:08

23
votes

JavaScript

1. numbers as strings - Math can be frustrating when numbers are intpreted as strings. 5 + 2 = 52? Grrr...
2. permissions - all the best stuff requires permission from the user!
3. screen updates - The browser must be in the steady state to update the screen. There doesn't seem to be a way to force the screen to update in the middle of a script.
4. Slow - although Google's Chrome is nice...
5. Browser differences make using the language a [censored].

Share

edited May 23, 2010 at 11:00

community wiki
2 revs, 2 users 91%
BoltBait

4 The numbers as strings is easily fixed. If you've ever got a string, you need to parseInt(x,10) it. The giant fail is when you leave out the ,10, and it interprets '017' as OCTAL
– Orion Edwards Nov 13, 2008 at 0:12

3 false == 0 == [] == "" but null and NaN are not. NaN != NaN. null == null. – Jimmy Nov 24, 2008 at 19:21

7 typeof "a string" == "string". typeof new String("another string") == "object. new String('a').constructor == "a".constructor. typeof new Array() == 'object' – Jimmy Nov 24, 2008 at 19:24

1 for(x in object) returns functions – Jimmy Nov 24, 2008 at 19:28

14 -1, this list is mostly about browser issues, not the language itself. – Mauricio Scheffer Jan 21, 2009 at 13:48

20 votes

PHP:

- One can never be sure that certain *almost common* extensions are available on all webservers.
- tries to be everything in future (goto, closures, ...)
- many security risks for unexperienced users
- more operator overloading would be nice
- all the poor programmers that don't learn how to make it work properly, and give it a bad name

Nevertheless PHP is *the* (scripting) language. ;-)

Share

edited Dec 5, 2008 at 17:53

community wiki
2 revs, 2 users 93%
okoman

OK, only one more thing to go! – brian d foy Nov 11, 2008 at 23:04

4 Totally agree with point 5 - would be on a Javascript list too. – Steve Claridge Oct 9, 2009 at 20:06

I disagree with "all the poor programmers that don't learn how to make it work properly, and give it a bad name". I'd replace it with "massive tacked on runtime language configuration options". – Longpoke May 23, 2010 at 17:34

18 votes

VB6

1. Windows only.
2. No longer supported.
3. Arrays can start at any number, rather then all being normalized to 0.
4. compiled applications depends on many dll's to run properly.
5. Many complicated controls like a browser control or complicated pieces of code tend to break the IDE

when you run code uncompiled, but work just fine when compiled.

Share

answered Nov 12, 2008 at 3:54

community wiki
Asrrin29

---

13 VB is someone's favourite language? O_o. Why isn't "syntaz that is completely different and incompatible with other languages" and "gives bad habits with regards to other languages" here? – Jonta Nov 24, 2008 at 19:15

3 I actually find #3 a very powerful feature, not a bug - I'd really love VB.NET had this. AWK has it, in a sense, but then in AWK arrays are really hashes in disguise :(– Joe Pineda Nov 26, 2008 at 23:11

3 On 1 and 4, and .NET C# doesn't require A COMPLETE FRAMEWORK and Operating System??? (hey, I heard that you mono bigot... it's still a "complete framework" for you, and I doubt a debian dist ever eats it). Regarding 5, no right-minded VB6 programmer (back in the day) kept the default "Compile On Demand" option ON... – jpinto3912 May 29, 2009 at 22:14

2 Still have to support vb6 occasionally. Pet pieves: can't initialize a variable at declaration, no parametrized constructurs, one class per file, etc... If they would fix these issues, the language could go on for another 10 years easy. – AngryHacker Aug 4, 2009 at 0:18

14 What about "On Error Resume Next"... thats like saying "this code is F**KED, but lets keep running it anyway. =) – StingyJack Oct 13, 2009 at 12:11

18

votes

Ruby is my favourite language, here's what I don't like:

- Green threads + blocking C libraries = giant fail
- SO PAINFULLY SLOW
- The standard library itself is inconsistent with its use of bang! methods
- Module include + extend is messy.
- "Open Classes" can't be scoped - I want to add a String#dostuff, but I don't want that to leak into all the third party libraries
- No binary deployment packaging solution.

Share

edited Nov 27, 2010 at 2:24

community wiki

2 revs, 2 users 88%

Orion Edwards

3 Have you tried Ruby 1.9.1? It offers a vast speed-up compared to Ruby 1.8.6 – Christian Stade-Schuldt Feb 6, 2009 at 20:52

Try jrubyc. JVM JIT FTW! – KitsuneYMG May 25, 2010 at 13:15

+1 for including reasonable problems, as opposed to the 'hates' the from the top-rated Ruby answer. – Phrogz Feb 21, 2011 at 12:38

17
votes

Delphi:

- IDE is a bit unstable.
- Code insight is sometimes confused.
- Debugging is sometimes buggy.
- Updating several project files can be cumbersome.
- If starting up when one or more packages are unavailable, the error message is popped several times.

Share

answered Nov 11, 2008 at 22:44

community wiki
Toon Krijthe

5 All of these seem to be complaints about Delphi the IDE rather than Delphi the language (AKA Object Pascal) – Dónal Nov 11, 2008 at 22:52

11 Presumably that's because Object Pascal is perfect ;-) – Mark Bessey Nov 11, 2008 at 23:42

3 I'm a bit late to the party, but here goes anyway: - having to write down method signatures twice (interface + implementation) - Unit name is REQUIRED to be identical to the file name. WTF?!? – Martijn Mar 27, 2009 at 17:30

1 I find the begin..ends to be superior--they're much clearer than {}. You spend a lot more time reading code than writing it. For a gripe, though--you can't use defined subranges of enumerated types in a case even though it's perfectly legal if you declare

the range right there in the case. Also, no forward references across units. – Loren Pechtel Jan 2, 2010 at 5:51

1 @AlexanderN: No, it has never been more alive, popular or great. – Andreas Rejbrand May 23, 2010 at 17:31

16 votes

JavaScript

- Every script is executed in a single global 'namespace'...something which you have to look out for when working with scripts from different sources
- If a variable is used but hasnt been defined before hand, it is considered a global variable
- Browser vendors making up standards as they please, making coding for us developers using such a beautiful language harder than it should be
- Case-Sensitivity - considering that there is no decent IDE for developing js with compile-time checking
- Workarounds (such as the use of `hasOwnProperty` method) to perform some, otherwise simple operations.

Share

answered Dec 7, 2008 at 15:19

community wiki
Andreas Grech

AFAIK, all extensions to the JS *language* (not the DOM) by browser vendors have at least been pushed for standard adoption—even if the standards process has failed to achieve that. hasOwnProperty/workarounds: double-edged sword. To force the "simplicity", we lose a lot of power and flexibility. That complaint always pisses me off. Write your loops right (and check your object members right too)! – eyelidlessness Jul 23, 2009 at 5:44

15
votes

Haskell:

1. Space leaks from lazy evaluation.
2. Numeric Hierarchy not constructed with regard to mathematical abstractions.
3. Strict monadic IO can make it harder to debug.
4. The big implementations handle I/O in ways that don't seem quite compatible with the standard. (In particular, outputting characters only outputs the low 8 bits -- and then code gets built that uses this assumption to do binary I/O. Ick.)
5. Associativity of `($)` operator could be changed to make some expressions prettier.

Most of these don't rise to the level of hate, and there are people trying to fix or construct solid workarounds for each of these.

Edit: There's been some confusion about point 5. In particular some people seem to think I meant the order of arguments, which I don't. Rather than explaining what I meant, I'll just point people to the following link, http://hackage.haskell.org/trac/haskellprime/wiki/ChangeDollarAssociativity , which expresses it well.

Share

edited Mar 1, 2011 at 20:56

community wiki
3 revs, 2 users 90%
wnoise

3 Why would you want to change the associativity of ($)? 'f g h x' brackets as '((f g) h) x' and 'f $ g $ h $ x' brackets as 'f (g (h x))'... – Erik Hesselink Nov 13, 2008 at 14:41

1 I <3 Haskell. The standard library needs to include mountains of mathematical abstractions, including vector spaces et al. The prelude also needs an operator which chains just like ($) but from left to right { source |> func1 |> filter func2 |> map (func3 10) }. – yfeldblum Nov 24, 2008 at 4:30

10 You missed out the really bad one: the tendency of Haskell programmers to use one letter variable names. – Benjamin Confino Apr 4, 2009 at 16:22

1 A left-associative ($) operator is just function application, which in Haskell is represented by the space character. @Justice: Try the flip function. (|>) = flip ($) – Apocalisp Jun 16, 2009 at 1:43

1 Can someone explain the point of #5? I thought right associativity was the whole point of ($). – Tim Matthews May 26, 2010 at 16:08