Python Conquers The Universe

2009/10/03

Python & Java: A Side-by-Side Comparison

Filed under: Java and Python — Steve Ferg @ 7:50 am

I wrote this little piece a long time ago — it was last updated in May 2007. In 2009 I moved it from my website to this blog. At this point I consider it an historical artifact, in the sense that I consider it frozen; I'm no longer going to update it when, for example, Java acquires a new programmer-friendly feature.

During the years when it was on my web site, I received a number of email comments. I responded to them all, but no one — other than their authors and me — ever saw the comments and my responses. By moving this piece onto a blog, I hope that folks who feel like commenting can do so more easily, and can share their comments with a wider audience.

During the years when it was on my web site, virtually all of the comments that I received were from Java programmers. Some were calm and helpful, pointing out (for example) new programmer-friendly features of Java. Many were hot and angry and accused me of writing the piece for the sole purpose of bad-mouthing Java. If this describes you, please don't fire off your angry counter-blast immediately. Go away, calm down, come back and read it again carefully.

I was recently asked if I still hold the opinions that I expressed in this article. I think that's worth talking about… but in another post.

— Steve Ferg

Updated October 14, 2011 to fix a few minor typos

Note that this post has a companion post on the concepts of weak vs. strong, and static vs. dynamic, typing in programming languages.

Comparing Python and Java

A programmer can be significantly more productive in Python than in Java.

How much more productive? The most widely accepted estimate is *5-10 times*. On the basis of my own personal experience with the two languages, I agree with this estimate.

Managers who are considering adding Python to their organization's list of approved development tools, however, cannot afford to accept such reports uncritically. They need evidence, and some understanding of why programmers are making such claims. This page is for those managers.

On this page, I present a list of side-by-side comparisons of features of Java and Python. If you look at these comparisons, you can see why Python can be written much more quickly, and maintained much more easily, than Java. The list is not long — it is meant to be representative, not exhaustive.

This page looks only at programmer productivity, and does not attempt to compare Java and Python on any other basis. There is, however, one related topic that is virtually impossible to avoid. Python is a dynamically-typed language, and this feature is an important reason why programmers can be more productive with Python; they don't have to deal with the overhead of Java's static typing. So the debates about Java/Python productivity inevitably turn into debates about the comparative advantages and drawbacks of static typing versus dynamic typing — or strong typing versus weak typing — in programming languages. I will not discuss that issue here, other than to note that in the last five years a number of influential voices in the programming community have been expressing serious doubts about the supposed advantages of static typing.

For those who wish to pursue the matter, Strong versus Weak Typing: A Conversation with Guido van Rossum, Part V is a good place to start. See also Bruce Eckel's weblog discussion Strong Typing vs. Strong Testing and Robert C. Martin's weblog discussion Are Dynamic Languages Going to Replace Static Languages?. For background, see one of the papers that started it all in 1998 — Scripting: Higher Level Programming for the 21st Century by John Ousterhout.

Several of these discussions contain valuable comparisons of Java and Python. For other language comparisons, see the Python language comparisons page at www.python.org, and the PythonComparedToJava page at Python for Java Programmers.

Finally, it is important to note that asserting that a programmer can be more productive in Python than in Java, is not the same as asserting that one ought always to use Python and never to use Java. Programming languages are tools, and different tools are appropriate for different jobs. It is a poor workman whose toolbox contains only a hammer (no matter how big it is!), and it is a poor programmer (or software development organization) whose development toolkit contains only one programming language. Our toolboxes should contain both Python and Java, so that in any given situation we have the option of choosing the best tool for the job. So our claim is not that Python is the only programming language that you'll ever need — only that the number of jobs for which Python is the best tool is much larger than is generally recognized.

Java vs. Python Productivity – an Overview

There are three main language characteristics that make programmers more productive with Python than with Java.

Java	Python
statically typed In Java, all variable names (along with their types) must be explicitly declared. Attempting to assign an object of the wrong type to a variable name triggers a type exception.That's what it means to say that Java is a *statically typed* language. Java container objects (e.g. *Vector* and *ArrayList*) hold objects of the generic type *Object*, but cannot hold primitives such as *int*. To store an int in a Vector, you must first convert the int to an *Integer*. When you retrieve an object from a container, it doesn't remember its type, and must be explicitly cast to the desired type.	dynamically typed In Python, you never declare anything. An assignment statement binds a name to an object, and the object can be of any type. If a name is assigned to an object of one type, it may later be assigned to an object of a different type. That's what it means to say that Python is a *dynamically typed* language. Python container objects (e.g. lists and dictionaries) can hold objects of any type, including numbers and lists. When you retrieve an object from a container, it remembers its type, so no casting is required. For more information on static vs. dynamic typing, see this post.
verbose "abounding in words; using or containing more words than are necessary"	concise (*aka* terse) "expressing much in a few words. Implies clean-cut brevity, attained by excision of the superfluous"
not compact	compact In *The New Hacker's Dictionary*, Eric S. Raymond gives the following definition for "compact": Compact adj. Of a design, describes the valuable property that it can all be apprehended at once in one's head. This generally means the thing created from the design can be used with greater facility and fewer errors than an equivalent tool that is not compact.

Example

The classic “Hello, world!” program illustrates the relative verbosity of Java.

Java	Python

`public class HelloWorld`
`{`
`    public static void main (String[] args)`
`    {`
`        System.out.println("Hello, world!");`
`    }`
`}`
	`print "Hello, world!"`
	--- 	
	`print("Hello, world!") # Python version 3`

Example

In the following example, we initialize an integer to zero, then convert it to a string, then check to see if it is empty. Note the data declaration (highlighted), which is necessary in Java but not in Python. Notice also how verbose Java is, even in an operation as basic as comparing two strings for equality.

Java	Python

`int    myCounter = 0;`
`String myString = String.valueOf(myCounter);`
`if (myString.equals("0")) ...`
	`myCounter = 0`
	`myString = str(myCounter)`
	`if myString == "0": ...`

`// print the integers from 1 to 9`
`for (int i = 1; i < 10; i++)`
`{`
`   System.out.println(i);`
`}`
	` print the integers from 1 to 9`
	`for i in range(1,10):`
	`    print i`

Example

Your application has 15 classes. (More precisely, it has 15 top-level public classes.)

Java	Python
Each top-level public class must be defined in its own file. If your application has 15 such classes, it has 15 files.	Multiple classes can be defined in a single file. If your application has 15 classes, the entire application could be stored in a single file, although you would probably want to partition it sensibly into perhaps 4, 5, or 6 files.

Example

In your application, method A calls B calls C calls D calls E calls F. You discover that F must throw exception SpecialException, and it must be caught by A.

Java	Python
You must throw SpecialException in F, and catch it in A. **and**	You must raise SpecialException in F, and catch it in A.Exceptions will propagate upward automatically; there is

You must add "throws SpecialException" to the signatures of methods B, C, D, E, and F.	nothing more that you must do.

The reason for this is that Java, virtually alone among object-oriented programming languages, uses *checked exceptions* — exceptions that must be caught or thrown by every method in which they might appear, or the code will fail to compile. Recently (as of June 2003) there seems to be an increasing amount of unhappiness with Java's use of checked exceptions. See Bruce Eckel's "Does Java need Checked Exceptions?" and Ron Waldhoff's "Java's checked exceptions were a mistake".

As ***chromatic***, the Technical Editor of the O'Reilly Network, put it:

> I like the idea of checked exceptions in some situations, but forcing every method to deal with (catching or throwing) all exceptions that its child calls or may call can be tedious. I'd rather be able to ignore an exception and let it propagate upwards. Sometimes, I'd rather not worry about exceptions at all.

Example

Your application has an *Employee* class. When an instance of Employee is created, the constructor may be passed one, two, or three arguments.

If you are programming in Java, this means that you write three constructors, with three different signatures. If you are programming in Python, you write only a single constructor, with default values for the optional arguments.

Java

```
public class Employee
{
    private String myEmployeeName;
    private int    myTaxDeductions = 1;
    private String myMaritalStatus = "single";

    //--------- constructor #1 -------------
    public Employee(String EmployeName)
    {
        this(employeeName, 1);
    }

    //--------- constructor #2 -------------
    public Employee(String EmployeName, int taxDeductions)
    {
       this(employeeName, taxDeductions, "single");
    }

    //--------- constructor #3 -------------
    public Employee(String EmployeName,
           int taxDeductions,
           String maritalStatus)
    {
       this.employeeName    = employeeName;
       this.taxDeductions   = taxDeductions;
       this.maritalStatus   = maritalStatus;
    }
...
```

Python

```
class Employee():

    def __init__(self,
        employeeName
        , taxDeductions=1
        , maritalStatus="single"
        ):

        self.employeeName    = employeeName
        self.taxDeductions   = taxDeductions
        self.maritalStatus   = maritalStatus
...
```

In Python, a class has only one constructor.

The constructor method is simply another method of the class, but one that has a special name: `__init__`

Example

In Why Python? Eric S. Raymond notes that:

> Python … is compact — you can hold its entire feature set (and at least a concept index of its libraries) in your head.

In Why I Love Python Bruce Eckel notes that Java is not compact.

I can remember many Python idioms because they're simpler. That's one more reason I program faster [in Python]. I still have to look up how to open a file every time I do it in Java. In fact, most things in Java require me to look something up.

Java	Python

`import java.io.*;`
`...`
 	
`BufferedReader myFile =`
`    new BufferedReader(`
`        new FileReader(argFilename));`
	`# open an input file`
	`myFile = open(argFilename)`

Example

Java's string-handling capabilities are surprisingly weak. (But they have improved considerably with the addition of the *split* method to the String class in Java 1.4.)

Function or Method	Java	Python
Remove leading and trailing whitespace from string **s**	`s.trim()`	`s.strip()`
Remove leading whitespace from string **s**	(not available)	`s.lstrip()`
Remove trailing whitespace from string **s**	(not available)	`s.rstrip()`

Example

Code to add an *int* to a *Vector*, and then retrieve it.

Prior to Java 1.5, a new *Integer* object had to be created and initialized from the *int* before it could be added to a Vector. In order to retrieve the value, the member of the Vector had to be cast back to an *Integer*, and then converted back to an *int*.

Java (before version 1.5)	Python

`public Vector aList        = new Vector;`
`public int    aNumber      = 5;`
`public int    anotherNumber;`
 	
`aList.addElement(new Integer(aNumber));`
`anotherNumber = ((Integer)aList.getElement(0)).intValue();`
	`aList = []`
	`aNumber = 5`
	
	`aList.append(aNumber)`
	`anotherNumber = aList[0]`

This clumsiness was eliminated in Java 1.5 with the introduction of generics (which allows you to "type" a container object) and autoboxing (which automates conversion between primitive types and their corresponding wrapper classes). With generics, it is possible to code:

```
ContainerType<ContainedType>
```

which reads as:

```
ContainerType restricted to objects of ContainedType
```

Java (after version 1.5)	Python

`public Vector<Integer> aList = new Vector<Integer>;`
`public int    aNumber      = 5;`
`public int    anotherNumber;`
	`aList = []`
	`aNumber = 5`
	
	`aList.append(aNumber)`

`aList.addElement(aNumber);`
`anotherNumber = aList.getElement(0);`
	`anotherNumber = aList[0]`

Example

Verbosity is not just a matter of increasing the number of characters that must be typed — it is also a matter of increasing the number of places where mistakes can be made. The Java code on the left has 5 control characters: `( ) { } ;` where the corresponding Python code has only one control character, the colon. (Or two, if you count indentation. See below.)

Java	Python

`if ( a > b )`
`{`
`    a = b;`
`    b = c;`
`}`
	`if  a > b :`
	`    a = b`
	`    b = c`

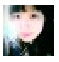
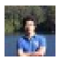
Omitting or duplicating such characters is easy to do accidentally, and constitutes a severe error in the code. In my personal estimate, I spend 5 times as much time fixing such errors in Java as I do in Python. It really cuts into your productivity — and your creative energy — when you spend that much of your time just trying to satisfy the compiler.

Technically, Python has another control character that Java does not — indentation. But the requirement for correct indentation is the same in Java as it is in Python, because in both languages correct indentation is a practical requirement for human-readable code. The Python interpreter automatically enforces correct indentation, whereas the Java compiler does not. With Java, you need an add-on product such as the Jalopy code formatter to provide automated
enforcement of indentation standards.

Acknowledgments

Thanks to Skip Montanaro, Chris Lawrence, Donald McCarthy, Bengt Richter, and Christian Pohlmann for helpful feedback on earlier versions of this page.

Like 2 bloggers like this.

Comments (46)

46 Comments »

1. I have a couple of comments:

   - Regarding the constructor example (3 constructors for an Employee). That's not relevant anymore, now that Java has support for varargs.

   - It is also a bit unfair to compare "==" with "equals". "==" means something different in Java. It is a test for an actual equality, and not two strings that share the same contents.

   Comment by Nja — 2009/10/04 @ 3:50 pm | Reply

   - In Python, "is" stands for identity and "==" for equality.

Comment by Nelson Houillon — 2010/08/22 @ 6:52 am | Reply

2.

This article is full of trivial mistake.

-statically typed

this works
Object a = new int[3];
and this too with autoboxing
int c = 0;
Object b = c;
and don't forget generics.

-verbose
in a world where developer do not can/want document their code,
verbose code is the only help.

-not compact
Exist, genirics, autoboxing, reflection, inline declaratio, anonymous or not inner class,
ecc.. to write compact "unreadable" code.

-Example public class HelloWorld

java is verbose but document that write a string to stdout.

look at print "Hello, world!"

where write? write a LF at the end?

-Example int myCounter = 0;ecc..

if you like compact and unreadable try

int myCounter;
if ("0″.equals("" + (myCounter = 0)))

Example 15 classes.

use Inner Class, you could write an entire webserver in one class,
but you could use only the main method and write procedural..

-Example exception

Use RuntimeException, or a mix of Checked and Runtime

throw new RuntimeException(new Exception(…))

personally i hate checked, but is only my attitude..

-Example employee class

in Java three costructor?

Pass a map to costruct very very dirty unreadable but works..

Dont forget that every costructor could have a different scope.. in phyton?

-Example IO file

myFile = open(argFilename)

documentation? is buffered? kind of stream? text file reader? binary?

libraries are very usefull too, es IOUtils for jakarta apache commons, lookat IOUtils.readLines

-Example trim

as before, StringUtils

-Example Vector

with Java collections is very different from array

there are many outofthebox implementation, sorted, unorderd, from synchronized list (vector) to Stack
but there are many other in libraries es Bags (special kind of set) as FastArrayList, look in org.apache.commons.collections

-Example formatter

Come on guy, there are so many IDE, command line tool, addon to versioning system that format code,
for every language.

Comment by Antonio — 2009/10/07 @ 9:14 am | Reply

3.

Impressive! Never thought one has to type so much boilerplate in Java. IMO, all those IDEs are the cure of this very issue which is artificial.

I've been writing a lot of Python code recently, getting away with gedit only. I can sketch a piece of code on a piece of paper at lunch (I don't like smartphones). Would that be possible in Java?

Seems like using IDEs with auto-completion, utility libraries, and other stuff in Java cures the issues that could have been avoided. It's like buying a Porsche Cayen in a city like Buenos Aires, San Paulo or Moscow: most your driving time you'll be in traffic jams, but there are things to make sitting in the car comfortable (TV, audio, maybe bar, maybe even a toilet xD).

Comment by culebrón — 2009/10/26 @ 7:19 am | Reply

4.

^ Exactly! A good IDE (in my experience Eclipse works best) can cure most, if not all of the above "shortcomings" of the Java syntax

Comment by ron — 2009/11/07 @ 4:17 pm | Reply

5.

If you actullay knew Java then this article actually would make sense. Now it's more fooling rookies.

Experienced programmers would never fall for these bad examples… Comaring python to Java is like comparing a bicyle to a car. Sure, it's easier to learn to ride and you don't need to struggle with the licencse, but as soon as you want to go more than 10 miles ("write an enterprise application"), the only choice of the two is the car.

Comment by Daniel — 2009/11/14 @ 5:16 pm | Reply

- Try that in London, You can easilly go 15 miles in an hour on a bike. In a car it takes two and you have to park the thing.

  Comment by bob — 2012/03/06 @ 10:57 am | Reply

6.

Your argument about compact / not compact is not really valid. Note that your quote from Eric S. Raymond talks about *design*, and not about any specific programming language. How you design the structure of an application is largely independent of the programming language you use. And the box "not compact" in the Java column is empty, you're not even presenting an argument why "Java is not compact".

I do agree that Java is more verbose than other languages such as Python, but a Hello World program is not a realistic example to demonstrate this.

Your second example, converting an int to a string and looking if the string contains "0″, is strange and is not the way you should normally write programs. And note that in Java you can leave off the braces, so the example with the for-loop in Java can be written in two lines (or even one line), just like the Python version.

There's a good reason why in Java you would want to save each public top-level class in its own file. This makes it easy to find the file in which a class is defined.

The example about the exceptions is only true if SpecialException is a checked exception. If you make SpecialException an unchecked exception, you do not need to change the method signatures of B, C, D, E and F. Your example makes it seem as if there are only checked exceptions in Java, which is not true.

So, this is not really a great comparison between Java and Python.

Comment by Jasper — 2009/11/30 @ 11:19 am | Reply

7.

Some notes:
Static vs dynamic typing:
static type checking may not help much with getting less errors but it helps immensely to get better tools. Show me an IDE that can do the same Eclipse can do for Java like
- show all callers of a method / uses of a variable/field project-wide
- change a class/method signature including all callers in a few clicks
- context assists and hyperlinks everywhere
I find bigger refactorings in dynamically typed language much harder because you often have to resort to simple text search/replace.

Multiple classes in one file:
This is nice for small scripts but bad for bigger projects… for best readability you want all your units (metohds, classes, files) to be as short as possible.

Java is verbose / parantheses, etc:
yeah, they are often superflouos. But modern IDEs mostly solve this problem with their content-assist and templates.

the rest:
For nearly everything that is complicated in Java (and all other things, too) there are good (as in enterprise-ready, properly documented (thanks@Sun for Javadoc), free) libraries. And you can assume that a JRE is already present on most PCs and smartphones and that, as long as you don't attempt any low level things, your code will run 100%. For python, you have it on most unixes by default… that's it.
Jython helps a bit, but its incomplete and far behind C-Python.

I don't say that Java is flawless, but I think that it currently is the best cross-OS development platform for big projects.

Comment by Burnstreet — 2009/12/15 @ 4:06 pm | Reply

8.

Some points to add as I find this article very misleading,

1- Java is pure Object Oriented and you MUST declare all objects, Python supports other programming paradigms so you don't need to declare all variables…that's also a Fortran problem and turn codes into "black boxes". For your example is ok….but if you have thousands of variables you have to guess each???

2- PythonDoc is an implementation of JavaDoc for Python…(yes, verbose helps a lot and Python copied that from Java!!!!) see how many men-hour can be saved if a good documentation is generated and attached.

3- Try to run programs that require tons of memory with undeclared variables. The reason Java declares all of them is 'cause it does allow memory allocation (virtual machine) and the Garbage Collector (never mentioned) can run efficiently.

4- Messiness is mostly due to the programmer not the language. Besides IDEs (Eclipse) already fix that problem.

5- Try to run codes in parallel (MPI) libraries to see who wins.

Comment by William — 2009/12/16 @ 4:32 pm | Reply

- import multiprocessing

  Comment by Will M — 2011/02/24 @ 6:42 pm | Reply

9.

there is a good reason for declaring variables. suppose that you have been working with the variable "foo" for a while. Later, you want to reassign it, but you make a typo, instead of:
foo=7
you type:
fooo=7

in a language that requires variables to be declared, this would generate a compile-time error. in a language that doesn't, the program would generate a new variable "fooo" and carry right along.

Comment by Jason — 2010/02/05 @ 8:48 pm | Reply

- W00t? This is the right behaviour. When you mistype it, you got fooo as a new variable. It's you own fault, not the language's!

  oh and btw. When you use a variable, which is not declared in Python, it will raise a NameError.

  Comment by Mike — 2010/05/01 @ 9:12 am | Reply

  - True. The interpreter will not prevent you to make programming errors.
    Java is mostly a newbie language that overprotects the bad programmers from themselves.

    Comment by Nelson Houillon — 2010/08/22 @ 6:57 am | Reply

    - You're joking, right?

      Comment by Luke — 2011/02/03 @ 4:21 pm | Reply

10.

The author claims that the increased productivity of python is due to its dynamic typing, but it seems to me that the overhead of having to declare the type of a variable is actually pretty small. And that small overhead brings many advantages, e.g. the ability of IDEs to autocomplete function names, show helpful tooltips, etc.

Java is verbose, yes, but most of that is not related to static typing. It's more for other reasons, e.g the unwillingness of the language designers to allow operator overloading; the horrible decision to use checked exceptions; no delegates; no multiple class definitions in a file; the unwillingless to "clutter" up the standard library with useful functionality that can be achieved in other, albeit more verbose, ways; the legacy from C syntax. And so on.

If you want to see a statically typed modern language done (mostly) right, check out C#. Sure it's a product of Microsoft, but it's one of the most nicely designed languages out there. Where Java prefers "elegance" and verbosity, C# is designed with practical programming constructs in mind. And since version 3.0, it does type inference as well.

Don't get me wrong, I still like Python, but for production quality large programs, I think C# is just as productive, runs faster, and is far less likely to contain stupid runtime errors caused by typos.

Cheers,

Sy

Comment by Simon Perkins — 2010/02/11 @ 8:02 pm | Reply

11.

Java wins because of refactoring and consistency.
Before super smart ide's like Eclipse, Netbeans, and Intellij I would say Python was a better programming environment then Java.

But these new ide's have changed the game completely. Instead of refactoring at the start of a new development cycle I refactor all the time. Instead of having UML charts and CRC cards and all that good OO planning crap you now (because of refactoring) can just write the code. I find proper language very important in expressing the domain you are working with that I care very much about the names of objects.

With python I would have to plan this in advance because it is a pain in the butt to refactor in dynamic languages.

Python is also not very consistent compared to Java. People can't decide on consistent naming of methods and there are few odd things (like print with out parens).
BTW have you noticed Python 3 looks more and more like Java.

Comment by Adam Gent — 2010/04/11 @ 9:50 am | Reply

- "Python is also not very consistent compared to Java"
  No, it's different form the beginning. Try to import your habits rom a language to another, the result will never be good.
  To understand the power of the Python language, you *must* play the game and conform yourself to the discipline.
  Moreover, why would I need a complex and heavy IDE since simple Gedit/Geany do the work ?
  Finally, you can refactor in any language you want. it does not depend on the syntax or whatever, but on your skills to avoid redundancy in your code.

  Comment by Nelson Houillon — 2010/08/22 @ 7:02 am | Reply

  - Are you seriously suggesting that you would write an enterprise application in Gedit?

    Comment by Luke — 2011/02/03 @ 4:27 pm | Reply

    - I do it in vim

      Comment by Odin — 2012/03/12 @ 4:13 am | Reply

- vim… I use vi

  Comment by adino — 2012/05/11 @ 1:09 pm | Reply

12.

Since everyone is bashing python, I would like to give my point of view from someone who has programmed in java and in python, and that likes python more than java.

For the ones that say that super IDE's solve the problems of java, I would like to say that I have used both Oracle JDeveloper, Eclipse and Sun Netbeans. And from JDeveloper I have to say that it lacks features from the other two, for Eclipse that I need to get an extra 1Gb of RAM in order to run it together with my other applications (Try to configure Eclipse, to show how many memory it is "eating" in order, to check what I say), and them we have Netbeans that is the nicest IDE for java, but the fact that it isn't easily expandable like eclipse with plug-ins means that you will have to download another version if you happen to suddenly develop something that requires J2ME instead of J2EE that you normally use. But all of these IDE's require memory and some experience time in order to be used efficiently, so using a super IDE to correct java problems it's not a way to solve things, at least not the way that I would use.

Now from the python side, it's true that you don't have a super IDE, but you don't need one in order to use python… All you need to use python is the own interpreter, to indent your code and a basic idea of what you want to do. And a simple text editor, like notepad to type your code if you would like to persist it… It's sure that python hasn't a lot of libraries and tools as java has. But the ones that it has, are enough.

Comment by Carlos — 2010/04/16 @ 9:33 am | Reply

13.

I can write my python programs in Microsofts note pad, even on iPod – heck on the back of a paper napkin if I want to… 98% of the time they'll run first time with no exeptions!
by the time you've finished cleaning bugs i.e: "crap that should be an int, hmmm… and the bracket", I'll be watching old episodes of baywatch, having already delivered the software to the client.
To top it all of I can (thanks to jython) write python for your beloved Java VM.

"Jython, lest you do not know of it, is the most compelling weapon the Java platform has for its survival into the 21st century" – SeanMcGrath

————–
print("boo ya")#the simplicity…
————–

But alas, speed/memory… crap JAVA wins…
Mobile platforms… JAVA wins.
Popularity… sigh JAVA wins…again. thanks to Universty brain washing…
Libraries… hmmm… JAVA, just… really, just…

But who really wins?… the programmer who can code in both languages proficiently, and who knows which to use and when.
… and who knows resisting the tide of change is futile…
———-
evil_laugh = "mwa" + "ha "*100
print(evil_laugh)#i havent even tested this code yet i know it just works 🙂 . Python 3.1
———

Comment by Benjamin — 2010/07/29 @ 11:28 am | Reply

14.

Sorry. But your just showing that Python is normally shorter than Java, which has nothing to do with typing.

Thinking about how I would write your examples in a statically typed language of my choice makes your python code look pretty verbose.

You're totally missing the point.

Comment by steve — 2010/08/06 @ 2:26 pm | Reply

15.

The last point about code comments is laughable – "In my personal estimate, I spend 5 times as much time fixing such errors in Java as I do in Python". Here is a secret – next time try using a decent IDE 😃

Overall agree, that py is very terse, readable and to the point – but it still lacks the ability to parallel process and is way too slow when compared to Java.

Comment by NS — 2010/08/17 @ 12:05 am | Reply

- ¿lacks the ability to parallel processing?
  Check this out

  * With python you have direct access to the pthread (posix) library: thread
  * A higher level threading interface easier to work with than thread: threading
  * Jet another library for trivially parallel loops: multiprocessing

  For more info check the documentation
  http://docs.python.org/library/someos.html

  For those on cluster computing, you also have MPI:
  http://mpi4py.scipy.org/

  If you are serious about bottlenecks, you can bind C/C++ code as a python module.

  Have a nice day

  Comment by Ishmael — 2010/10/29 @ 5:34 pm | Reply

  - > If you are serious about bottlenecks, you can bind C/C++ code as a python module.

    So what you're saying is that one of the advantages Python has is that you can rewrite your stuff in C or C++ if you're "serious" about performance bottlenecks. Hahaha that's priceless.

    Have a nice day!

    Comment by Luke — 2011/02/03 @ 4:33 pm | Reply

    -

      You can also extend Python with D: http://pyd.dsource.org/ but i think Cython is easiest: http://cython.org/

      And C++ isn't that hard with ShedSkin: http://code.google.com/p/shedskin/

      Or just use PyPy: http://pypy.org/

      Comment by Cees Timmerman — 2011/09/19 @ 10:14 am | Reply

16.

it is foolish to ignore the importance of a compiler's help. so python would be a disaster for a new large project, BUT today 99% chance is you build a small part of a big system, so python may be more productive and flexible.

Comment by jackpy — 2011/05/21 @ 1:12 pm | Reply

17.

I don't think dynamic typing makes more productive:

- if you don't declare the variable type, then your IDE cannot know what it is and cannot save you a tremendous lot of keystrokes with features like auto-completion.

- if you don't declare the variable type, then there is no compile time (or even pre compile time) check possible to point out typos which will lead in errors at runtime.

- if you don't declare the variable type, then people using your methods won't know what they should pass to it by simply reading the method signature.

- if you don't declare the variable type, then your code will be less easier to understand and to maintain.

Comment by Anonymous — 2011/07/23 @ 10:39 am | Reply

@Anonymous (#17) – You seem to assume that Python is just as overly complex as Java, but this is not the case. Python is quite simple to read and it therefore reduces the amount of errors that are made in the first place so you do not need to declare errors before compilation (and btw, Python checks the files syntax on load). Also, Python functions and methods can have a docstring which can explain exactly what type of data to pass to it and again, without the extra complexity of Java, it is easy to figure out. And your last comment is simply a myth. I have absolutely no problem reading or maintaining Python code which has even a small amount of documentation. Instead, type declarations just make the code more complex than is needed and restricts flexibility.

As far as IDE auto-complete for certain types, it is not needed as the small number of methods used to interface with Python objects can be built-in and it can still save you many keystrokes.

Comment by aj00200 — 2011/08/22 @ 3:05 pm | Reply

18.

Definitely not agreed with your article. Java is far more better for many reasons. I grant you that put down some lines in a bare notepad and compile them in Java is error prone, longer and surely boring (you have probably to search for imports, write long names, etc…) but with all the existent environmnents around JAVA and for Java, you don't have any problem. All is fast, reliable, fit for big and scalable projects, and you can open files in any editor without screwing all with a wrong Tab option set (The Python biggest deficency for me is to be tied with the way you write and so the editor you are using).
I programmed in python but the performance is also poor. You can consider python for little projects, and be aware of some inheritance differences with java.
Also, if you want to use some third party library , you have to hope that the maintainer were careful about documentation sync with the code, thing that I rarely found in python external libs.

As in php (that is currently better than python as a web language), i'm a fan of the static type instead of dynamic. This eliminates the principal cause of errors in compiling and (worse) running sessions. In a lot of coded lines, it's difficult to see an error like :

```
variable=1;
…
print varialbe;
```

python does not warn me (because of dinamic typing the 'varialbe' is initialized to null).

Anyway i think that python could become a good language in the near future if it'll be surrounded by many support and enterprise structures that'll permit to use it as a strong and solid base for enterprise applications. And in any case it will be good surely for fast prototyping.

Comment by Mizar — 2011/10/11 @ 9:22 am | Reply

-

  So your argument is both Java and Python suck without a decent editor. Good thing that Python is so terse, then. As for your example, it doesn't work as you described it:

  ```
  C:\Python32>python
  Python 3.2 (r32:88445, Feb 20 2011, 21:29:02) [MSC v.1500 32 bit (Intel)] on win32
  Type "help", "copyright", "credits" or "license" for more information.
  >>> variabler
  Traceback (most recent call last):
  File "", line 1, in
  NameError: name 'variabler' is not defined
  >>> variable = 1
  >>> print varialbe
  File "", line 1
  print varialbe
  ^
  SyntaxError: invalid syntax
  >>> print(varialbe)
  Traceback (most recent call last):
  File "", line 1, in
  NameError: name 'varialbe' is not defined
  >>>
  ```

  Comment by Cees Timmerman — 2011/10/16 @ 7:18 pm | Reply

19.

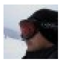
i m a student and using java and found it too useful for big projects.i used c and c++ already and now unfortunately learning python 😟 and found it too difficult bcoz sometimes its too difficult to use such a simple syntax and i often got stuck in it 😟 ((((((((

Comment by nazish — 2011/10/27 @ 11:24 am | Reply

-

  That's rather vague. Could you be more specific about your problem? In Java I spend more time figuring out the frameworks than actually solving business problems.

  Comment by Cees Timmerman — 2011/10/31 @ 3:35 pm | Reply

20. 

I have used Java for over 12 years now, and I agree that Java has become cumbersome in some areas, which is why open minded Java developers are looking elsewhere. However, definitely not for the above reasons. Most of them are either syntactic sugar variations of the same thing, or otherwise subjective opinions.

When one looks objectively at the above examples, they definitely do not demonstrate the 5 to 10-fold speed optimisations claimed in the article. In most examples its simply a question of syntactic sugar (e.g. Java requires { } to surround a block of code while Python does it with indentations… no big deal or optimisation, especially considering that most IDEs do these automatically, so if you're a manager thinking that this will make your developers faster, it won't.)

In most cases its just a question that Java requires some structure and not things floating around. The latter is available in

Python simply because its a scripting language.

If one considers Example 1, on its own yes Java requires a class to put your stuff in so its 2 extra lines. But realistically no minimally useful script or class will be that simple. So its just a question of wrapping things in a structured way in Java.

== Static vs Dynamic ==

The question of static typing versus dynamic typing is a purely a question of opinion and the application domain you're developing.
One downside to dynamic typing is that larger enterprise applications become more difficult to refactor. In static typing, changing a couple of classes will automatically break the compilation and you'll immediately know what else you need to fix in order to have the program working again.With most modern IDEs this is also facilitated by refactoring tools which tell you immediately what will be impacted before the change and also automate the changes.

Another problem with dynamic typing is that its more difficult for IDEs to auto-complete fields and methods if the type cannot be disambiguously deduced.

== Multiple classes in one file ==

One can't seriously state that this is an advantage. Java enforces you to do it not because the Java guys couldn't do it, its because it is good practice to do so. If you don't do this the code becomes unmaintainable as soon as it exceeds the 15 classes mark. You want to change the Employee class in java, you just have to open Employee.java, you need to do it in Python… good luck searching through the files.

I also use PHP and even though PHP carries the same 'advantages' from Python (it is both dynamically typed and allows multiple classes in one file) I still put my classes in separate files which carry the same name. Any serious work I've seen also adopts the same approach.

Again, if you're a manager, this will slow your developers due to decreased maintainability not make them faster, so dont even consider it. If you do go for Python, it still makes sense to have classes in one place.

== Compactness and Readability ==

I tend to disagree with most statements in the examples.
Java adopts a very simple convention, that constructors have the same name of the class. Its logical, intuitive and makes perfect sense.
Why should a class have a method with the same name? It only makes sense to be the constuctor.

Instead in Pyton you have to use a cryptic method called __init__
Apart from the varargs issue, which is in fact supported (maybe not as cleanly) in Java, constructors in Java are far more intuitive.

You might also have noticed the 'self' argument in the constructor. So essentially, the class's constructor also needs to be polluted with this parameter because there is no inherent 'this' that can be used within the class. Java supports this much more cleanly with a supplied 'this' keyword. Its also consistent because all developers use that, while Python all developers do what they want, some use 'self', some use 'this', some use 'myself'…. confusion welcome.

With regards to using indentations instead of curly brackets. Again this is not an advantage at all. It just invites confusion in. Most IDEs put the closing curly automatically so its not something that will contribute to slower development. On the other hand it clearly delineates code blocks.
Using indentations just invites problems with different text editors. The curly also has the advantage that most IDEs highlight the opening curly when you select hover over the closing curly (even though if you adapt the allman-style coding this won't be needed anyway).

———

I am not saying that all is rosy on the Java side. I am just saying that these examples are the perfectly wrong reasons to consider.

Some of the problems of Java are:

1. JAR hell.

The way libraries are managed in Java were elegant in the early 2000s where you would have a handful of Jar files and you would know what each was doing. Nowadays, there are so many of them that you would need to include in an application that it becomes a nightmare to manage, especially due to interdependencies between them and conflicting versions.

Maven came along which kind of solves this problem, but its still there, just managed a bit more intelligently.

2. Desktop performance

The performance of Java on the desktop for some reason never got anywhere near native or .Net applications. You immediately know that you have a Java desktop application because its slow to start up, hangs intermittently (due to garbage collection) and has a lousy UI.
In fact the only 'successful' desktop apps written in Java are in fact Java IDEs.

This is in fact a true source of slowness. Java developers typically waste 20% of their time playing around getting their environment to work. Starting up the IDE takes minutes. Then it starts hanging after an hour or so of developing. You can throw more computing power at it obviously, but we've been throwing more computing power at it for the last 10 years with newer PCs, more memory and faster CPUs, and the problem is still there.

3. Web Applications

Developing a web application in Java is in fact very complex.You have to understand the whole framework of web application servers, servlet containers, application contexts, and all just to even start doing a simple web application. When you've done that you realise that what you know is not really enough to make a maintainable reasonably sized application, so you have to learn other technologies such as Spring, Spring MVC, or JSF. After you've done that you realise you also want to hit the database, and you have to learn JDBC and JPA.

By that time, a developer in another more web-oriented language would have already completed the project, got the pat on the back and moved on to another one.

The end result in Java is very neat but the learning curve is very steep. You can easily go wrong somewhere and the moment you have a problem the tools are less than helpful. The only advantage of going for Java for web apps is when you have a large application which requires thought, discipline and good object oriented design, maybe with some scalability needs and requirements for message queuing (JMS etc.) or integration with more Java friendly components.

Developing web services is also similarly complex. Conflicts between the bundled JAX-WS and any updated version often drive Java developers crazy and is a major cause of suicides.

So, yes this is a true source of slowness of Java.

Bottom line is that most of the slowness of Java is not with the language itself, which is just C-style like 70% of the other languages (C, C++, C#, Javascript, PHP…). Its because expectations in certain areas have outgrown its original design and other more domain-specific languages provide a more agile framework.

Comment by jbx — 2011/11/16 @ 6:11 am | Reply

21.

Disclaimer: I'm not a proper programmer, I'm a mathematician looking for unbiased advice.

I like python, don't know Java. I find it highly productive. I don't get the arguments that it's not suitable for large scale applications because it doesn't enforce "discipline" or "good object oriented design". I don't want the language to enforce anything, but maybe it helps if your coworkers write sloppy code.

However, it seems like Python is used more by the academic community (most notably the academics without a CS background, such as myself), whereas Java is used more for large enterprises, can someone enlighten me as to why this is the case? I'm currently in the process of writing a prototype for a large data retrieval and analysis framework. I get results quickly in Python (hence why I use it for prototyping) but I'm worried that I might have to rewrite everything in a non-scripting language later on. Should I be worried? The plan is to get investment capital so we can employ some "proper" programmers who knows these things!

Basically, is this article as biased towards Python as it appears to be from all the comments?

Comment by Rickard — 2011/12/18 @ 7:33 am | Reply

- “…whereas Java is used more for large enterprises, can someone enlighten me as to why this is the case…”

  There are perfectly valid reasons for using Java on enterprise projects (strong server infrastructure, frameworks, portability, etc), but there are also other reasons why Java is often used for enterprise projects, even where it is not necessarily the best choice, especially when large outsourcing consultancies are involved:

  1. Universities teach Java, so CS grads know it (a bit…) => cheap grunt programmers easily available.
  2. Indian universities turn out lots of Java-enabled CS grads => even cheaper grunt programmers available offshore.
  3. Java “culture” encourages the idea that you can do everything in Java => no need to teach your grunt programmers about anything else, even if it would be useful to do so.
  4. Java EE is very complex and becomes ever more complex in line with Moore’s Law i.e. complexity doubles every few years. Complexity => costs => profits for consultancies.
  5. Efforts to combat that complexity e.g. Spring have themselves now become almost as complex as the “problem” they were intended to solve => complexity generates complexity.
  6. Once you’ve invested in a complex Java EE system, you’re pretty much trapped in Java-land => more profits for long-term support agreements etc.

  Comment by Matt — 2012/06/01 @ 7:06 am | Reply

  - “… Java is used more for large enterprises, can someone enlighten me as to why this is the case…”

    I’d like to add to Matt’s last point, which I’ll rephrase like this: “For any given programming language J, once you’ve got a major investment in J, you’re pretty much trapped in J land.”

    Suppose you are an enterprise IT manager. Your staff know programming language J. You’ve developed a number of large systems in J; you’ve created build, test, and deploy procedures and tools for programs written in J; you and your staff are comfortable with J and have mastered the art of system development in J.

    Now suppose that you have some new kind of programming task to perform. The task can be done using J (albeit it will take a while, and the result will be not very elegant) or it can be done using using some other programming language P, where the task can be done in P much more easily and elegantly.

    The problem is that you already know J — there is no learning curve to climb with J — but you’d have to expend some time and effort to learn P. And you know that there will be other tasks down the road that might be best done in some other language P2, and some other tasks for which P3 is the best tool, and then P4, and P5, and so on. And you know that that way lies madness. If you go down that path, you’ll soon have to maintain systems written in a dozen languages, and eventually you’ll find yourself in a situation where some programmer leaves; he wrote some system in programming language P9; but he was the only one who knew P9; now he’s gone and none of your staff can maintain the system.

    So you decide to standardize on J, where J is some general-purpose widely-used language that has a lot of add-on packages and frameworks that allow you to do just about anything in J. You make the decision: “From now on, we do EVERYTHING in J.”

    The managers of many (most?) enterprise shops are in this situation, and they choose to standardize on Java because they need to control costs, and to control the number of programming languages that they are using. This decision is partly a matter of economics (the need to control costs), and partly a matter of management (the need to manage the number of languages used in your shop). These are good reasons, and the basic ideas behind the decision are undeniably good ones.

    So that is my take on why “Java is used more for large enterprises.” Now: here is my take on why that is a bad idea.

    The problem is that — although the basic reasons motivating the decision are good ones — the idea that ONE size fits all is simply too restrictive. A tool box that contains only one tool is simply not enough. If you confine yourself to just ONE language, you will find that there are cases that are simply too much of a stretch for that language. Personally, I think that every shop needs to standardize on at least *two* general-purpose languages — one "system" language for major systems where performance is the highest priority, and one scripting language where ease-of-use, flexibility, and speed of development are the highest priority.

Personally, I think a combination of Java and Python is a good choice. And I believe that a management decision that "From now on, we do EVERYTHING in Java," is a big mistake.

But that was the situation that I was in when I originally wrote this article. Our shop at the time used a systems language and a scripting language— Java and Perl. We had found through experience that the Perl was unmaintainable. And we had decided to migrate our Perl applications to something else. I thought that Python would be an excellent replacement for Perl.

But the division chief felt that we should standardize on a SINGLE language — *only* on Java — and that we should migrate all of our Perl apps to Java. In my opinion, this would have been a very bad (actually, a ridiculous) decision, so I wrote this post to explain (in part) why I thought that *for some applications* Python is a better choice than Java.

So that's all I have to say on that subj… What? Oh, you want to know the end of the story? Well, OK…

There was a management turn-over. The new division chief had a completely different "let a thousand flowers bloom" philosophy. It was like a breath of fresh air. Soon the group — in addition to its older systems in Java and Perl — had new systems written in Python and Ruby. I eventually developed a large application (30,000+ lines) in Java, and a testing framework for the application (the first such framework in our group) written in Python.

That was the upside. The downside is that because the group never decided to standardize on a scripting language (Perl, Python, Ruby, whatever), it now faces the problem that I mentioned earlier. I was the only one in the group that used Python. When I left the group, they were left with a large, sophisticated test harness written in Python… but without anyone who knows Python to maintain it.

Comment by Steve Ferg — 2012/06/01 @ 1:41 pm | Reply

◦

is this post as biased towards Python as it appears to be from all the comments?

As the author of this post, I'd like to talk a little bit about why (I think) there are so many allegations (most of them, in fact, attached to the companion article on typing) that the post is biased, and about whether or not I agree with those allegations. I'm going to paint with a broad brush here.

I think that the first reason for the charge of bias is that **the wording of the post is carefully nuanced, but many readers missed the nuances.** In fact, many readers completely missed the point that I was trying to make.

If you read the post carefully, you will see that I make no claims that Java is a better programming language than Python, or that you don't need Java because Python is the only programming language that you will ever need, or that you can do all of your programming in Python, or that Python is superior to Java in any way other than productivity. I make no claim that I'm offering a comprehensive, technical feature comparison of the two languages. In fact, the post contains several statements denying that I'm making any such claims.

But I think that many Java programmers quickly skimmed the post and responded as if I'd simply grunted "PYTHON! GOOD! JAVA! BAD!"

Naturally, their reactions were negative and, in many cases, angry. If you read the comments, you can see that some of the commentators are almost incoherent with rage.

I want to note, however, there were a few Java programmers who read the post more carefully, and responded with helpful information and calm, reasonable, comments. It was a pleasure and a learning experience to read those comments, and (amazing as it may seem) I agree with many of them.

The second reason is that **most of the comments are from Java programmers.** After reading the post, I think that most Python programmers simply shrugged — "Yeah. So what. I know that. Nothing new here." — and moved on. In contrast, many Java programmers read the post as an ignorant yet vicious attack on their favorite technology (see reason number one, above). Naturally, many of them responded, and responded with vigorous dissent.

So I think the charges of bias can be accounted for by the fact that most comments are from Java programmers, many or most of those Java programmers are angry, and many of the angry Java programmers did not read the article carefully or objectively. I don't think the post was biased, but I think that many readers read it as making more sweeping claims about Python and Java than I was actually making. (Probably, I should have worded the post differently, although I'm not sure how much difference that might have made. For what I would do differently if I were writing the post today, May 2012, see below.)

Let me talk a bit about the kinds of comments on the post. There were many kinds of comments, but a few (sometimes overlapping) categories stand out.

The first category includes comments citing (a) the superiority of Java IDEs (over simple text processors or Python IDEs), and/or (b) the superiority of Java's declarative static typing over Python's dynamic typing. Some of these comments asserted that — "Yes, Java is more verbose than Python. But Java IDEs are so good that they make up for the difference, and in effect allow a Java programmer to be just as productive as a Python programmer (as measured by function points per keystroke)."

Some of these comments asserted simply that you cannot build large (enterprise-scale) applications in a dynamically-typed language. Some of these comments asserted that — "The declarative statically-typed nature of Java allows Java IDEs to be much more powerful and useful than simple text editors or Python IDEs; this makes Java programmers more effective when developing — and especially when refactoring — large code bases."

A second category asserted that I was comparing apples and oranges. Python may be OK for small systems, but for large systems (a) Java's static typing, (b) Java's superior performance, and (c) the superior powers of Java IDEs make Java the only rational choice. A nice example of this type of comment came from Daniel:

> Comparing Python to Java is like comparing a bicyle to a car. Sure, it's easier to learn to ride and you don't need to struggle with the license, but as soon as you want to go more than 10 miles ("write an enterprise application"), the only choice of the two is the car.

A third category of comments focused on features of Java. Such comments typically included assertions like — You don't really know Java; your example of Java feature F was wrong; the latest version of Java has a new feature F which invalidates one of your points; or you should have mentioned the following Java features (followed in some cases by quite a long list of features). In this category, many comments began with words to the effect that either (a) "You're wrong. Java is by far the best language because Java …[list of Java features]" or (b) "I agree. Python is by far the best programming language. I can't stand Java because Java …[list of Java features]".

I'm not really sympathetic to the third category of comments, because they completely miss my point. I don't think that there is any "best" programming language. As I wrote in the introduction to the post:

> Our toolboxes should contain both Python and Java, so that in any given situation we have the option of choosing the best tool for the job. So our claim is not that Python is the only programming language that you'll ever need — only that the number of jobs for which Python is the best tool is much larger than is generally recognized.

I'm much more sympathetic to the first two categories of comment. The comments citing the superiority of Java IDEs were appropriate a few years ago, although they now seem less compelling with the release of the JetBrains PyCharm Python IDE. But I think that I would agree that the statically-typed nature of Java allows Java IDEs to be much more powerful than Python IDEs, and this helps Java programmers immensely as they develop and refactor large code bases.

The downside, of course, is that it is virtually impossible to develop a Java program outside of an IDE. Java programmers are virtually chained to their IDEs (and their specific IDE — Eclipse, NetBeans, whatever) in a way that Python programmers are not. Technically, this may not be a *productivity* issue, but for some developers it is definitely a personal taste issue.

In conclusion — If I was writing the post today, I would be less inclined simply to talk about the "productivity" of Python programmers compared to Java programmers. There are many dimensions for measuring productivity, and many kinds of applications, and a language feature that enhances programmer productivity for one type of application might actually impair programmer productivity for another type of application.

If I was writing the post today, I think that instead of talking about "productivity" I would instead talk about matching tools and tasks. Any large organization is in the business of building many kinds of software applications. I would emphasize that for building some types of applications, a systems language such as Java is the best tool; and for building other types of applications, a dynamically-typed language such as Python (or Ruby) is the best choice.

Comment by Steve Ferg — 2012/06/01 @ 4:17 pm | Reply

- thanks for the clarification. Java is now vindicated 😐

  Comment by Anonymous — 2012/06/09 @ 9:57 am | Reply

22.

There is a Linux distribution written in Python: Pardus. That proves you can do big projects in Python.

Comment by Kiyas — 2012/01/13 @ 7:51 am | Reply

- Come on man, an open source linux distro will have 100′s if not 1000′s of developers working on it.

  I've read most of the comments here and I've used both Java and Python. Leaving aside the comparisons of the two languages I agree mostly with what JBX and what Steve Ferg had to say in the end.

  I think as programmers and software "houses" we shouldn't be comparing languages, each language has a specific purpose because of their differences.

  A heavy language like Java, C, .NET and objective C is a good weapon in your arsenal for systems programming (Java and C would be the clear winner since .NET and objective-C are platform dependant). A scripting language like ruby, python, hell even PHP is a good weapon for web development and rapid prototyping. A lightweight powerful language like Ada is great for embedded programming and for use in safety critical situations where program proof of correctness is more important than lack of bugs (which beats strongly typed by a long mile if bugs are a huge issue).

  So at the end of the day, having a plethora, yet minimal amount of languages suited for different purposes is key to being a good programmer or software 'house'. I can't speak for development time (that is a none issue in a perfect world, and our aim should be to make the world perfect, because we're engineers, not bankers or financiers).

  So using the right tool for the right job is more important than which programming language is better. Oh and coming up with the right architecture for your system, might give you the ability to use more than one language for different parts of your system possibly making development time faster.

  Comment by Venison — 2012/06/25 @ 6:11 pm | Reply

23.

I am a Pythonista myself, but I think it is fair to add http://www.ibm.com/developerworks/java/library/j-ce/index.html to the discussion about checked exceptions. It actually made me think they are not bad idea as I thought they are before.

Comment by Matěj Cepl — 2012/01/27 @ 3:12 am | Reply

24.

Two things. In Java I use an IDE, which is a machine that helps me write code. It can do this well, much better than any Python IDE can, because Java is strongly typed. I use Jython to script my Java application, and Jython is great for building small applications, but it's looseness makes it difficult to build anything large. My Java app is 400K lines long and I can get around no problem because of Java's precision. Jython scripts that drive it get long and a bit unmanageable at 500 lines…

Comment by Anonymous — 2012/02/17 @ 9:54 pm | Reply

- agree. verbosity with Java is not a problem unless you write your codes in a text file, and no one does that for a big application. python is a great language and it has its own purpose. This is just another meaningless article.

  Comment by Anonymous — 2012/04/19 @ 3:41 pm | Reply

RSS feed for comments on this post. TrackBack URI

Leave a Reply

Enter your comment here...

- Search

- ## Recent Posts
  - Python Decorators
  - Unicode – the basics
  - Python's magic methods
  - Gotcha — Mutable default arguments
  - Backing up your email
  - Unicode for dummies — Encoding
  - How to post source code on WordPress
  - Beautiful Code
  - Python3 pickling
  - Yet Another Lambda Tutorial
  - Read-Ahead and Python Generators
  - In Java, what is the difference between an abstract class and an interface?
  - Newline conversion in Python 3
  - Why import star is a bad idea
  - URL for PyCon 2011 videos

- ## Topics
  - Decorators (2)
  - Java and Python (3)
  - Keyboards (2)
  - Miscellaneous (6)
  - Moving to Python 3 (7)
  - Python & JSD (2)
  - Python debugger (1)
  - Python features (7)
  - Python Globals (3)
  - Python gotchas (5)
  - Software Development (6)
  - Subversion (1)
  - Unicode (4)

- RSS - Posts

Theme: Rubric. Blog at WordPress.com.