Python vs. Ruby

A work in progress. Listed approximately from most to least important, in my opinion. (They get re-arranged fairly frequently as my feelings on the issues change.)

Ways in which Python is superior to Ruby

- **A richer set of built-in data structures.** Python has lists, which are like Ruby Arrays: `[1, 2, 3, 4]`. It also has tuples, which are like Arrays but immutable, with syntax that's more natural in some cases: `(1, 2, 3, 4)`. Python has dictionaries, which are like Ruby Hashes: `{1: 2, 3: 4}`. It also has sets, which are like … Ruby's Sets, only they have syntax: `{1, 2, 3, 4}` and don't need you to import a library, even one that comes with the language. The collections module in the standard library also contains a deque data structure, a doubly-linked list with head pointers at both ends for when you need a list with O(1) appends and prepends, as well as an ordered dictionary class (though this should be the default behaviour) and a dictionary supporting default values for key access (likewise). ¶
- **Namespacing and modules.** Ruby's approach to avoiding naming collisions is to throw everything in one big namespace pot and hope nothing bad happens. To help avoid collisions, there is a convention of putting things into classes and modules as often as possible, the theory apparently being that the less names there are, the less likely there is to be a collision. Python's approach is to give each file its own namespace to play around in. For the programmer, that means that in Python, even if another file which you imported already imported a module, you must import that module yourself again. You can also rename modules within each file if there is a naming collision. ¶
- **Pervasive use of iterators.** Ruby has iterators, and uses them occasionally, but they're not so central to the language as they are in Python. Their implementation is a hack around blocks and mixins,

and they don't naturally combine so easily as in Python. (Now, it would be nice if Python's iterators were purely functional, but that's not really hugely significant.) ¶

- **Unicode strings vs. byte strings.** Python distinguishes between "Unicode strings," which it thinks of as collections of codepoints (you, the programmer, shouldn't have to know or care about what encoding they're stored in) and "byte-strings," which are classical bags of bytes with no determined encoding, which you can convert to a Unicode string by explicitly stating an encoding to parse with. Ruby, on the other hand, treats all strings as byte-strings with a semi-hidden flag to tell what encoding they're stored with. This causes no end of problems when dealing with badly-encoded data from third-party sources. (If you've never seen an `Encoding::UndefinedConversionError` or an `invalid byte sequence in UTF-8` error in Ruby, you're either lucky, damn smart, or unaware of the existence of Unicode, in which case your app is likely to break the first time someone types some non-ASCII characters into it.) ¶
- **Internal functions.** Python got these from Scheme, as far as I can tell. The idea being that you can nest `def`s to define functions that can only be used within the scope of another, enclosing function. This is terrifically useful. When Ruby encounters nested `def`s, it defines a new function on the class of object the enclosing function was called on. This behaviour is so useless as to be laughable, especially when, if you really wanted that, you could use `define_method`. ¶
- **A single namespace (within each module) for functions and variables.** Python is a Lisp-1, Ruby is a Lisp-*n*, depending on how you measure it. ¶
- **Date and time handling.** Ruby has three (count 'em!) classes for dealing with dates and times. How's this for crazy: `Date` stores a date as a year–month–day tuple; `DateTime` stores a date and time together; `Time`, contrary to what you'd expect, also stores a date and time. The difference between `Time` and `DateTime` is that the former is built in to the language, and (internally) stores the date-time as a POSIX `timespec`, whereas the latter is in stdlib, meaning it comes

with the language but isn't loaded automatically, and it implements the Gregorian calendar "properly" (counting days since some arbitrary epoch nobody cares about, and adding time-of-day on top of that.). Python also has an awareness of the 'naive' vs. 'aware' date-time objects, the former having no conception of the existence of time zones and the latter being extensible to allow support for e.g. the IANA tz database for full historical timezone awareness. ¶

- **Function and class decorators.** It'd be nice to have these as well as mixins. ¶
- **Multiplexed I/O system calls.** Ruby's `IO.select` is nice and simple, but inflexible. Python provides a whole damn library of interfaces to various UNIX/POSIX variants of multiplexed I/O system calls. Python also has an interface to the BSD kqueue system calls, which Ruby lacks. On the downside, `IO.select`, contrary to its name, sensibly picks either `select` or `poll` depending on which is most suited to the situation; Python makes you choose for yourself. ¶

Ways in which Ruby is superior to Python

- **Fragmentation.** Ruby 1.9 was a fairly major change to the language — not as big as Python 3, but still pretty big. As far as I can see, most libraries (every one I've used in the last year) now have full support for running on 1.9. By contrast, Python 3 is still not supported by a number of fairly important libraries, such as Web.py. ¶
- **Blocks.** I can't overstate how much nicer code looks, and how much more elegant "DSLs" become, when have Ruby's blocks are at hand. From what I can tell, Python seems to work around this by using decorators, classes, and (named) first-class functions more often. (Also, it's *still* crazy that lambdas can't have more than one expression in them.) If Python got Ruby-style blocks, it would invalidate pretty much everything else on this list. ¶
- **List comprehensions are incomprehensible** when they're anything more than trivial in what they do. I think that, for example, `(0..99).select {|x| x % 2 == 0 }.map {|x| x ** 2 }` is easier

to understand than `[(x ** 2) for x in range(100) if (x % 2 is 0)]`. ¶

- **String interpolation.** Ruby lets you do `"The sum of 2 and 2 is #{2 + 2}"`; Python makes you use `sprintf`, but at least has some nice sugar for it: `"The sum of 2 and 2 is %d." % (2 + 2)`. Python recently gained a new way to suck at this: `"The sum of 2 and 2 is {0}".format(2 + 2)`, which manages to be both longer and less familiar to those who are used to `printf` format strings from other languages. ¶
- **Symbols.** It's a pain in the arse that Ruby's symbols aren't garbage-collected, but at least it has them. (Python's strings are immutable and can be interned, but this feels like a lame substitute. Speaking of which ...) ¶
- **Everything is an expression.** In Python, why can't I do `a = if b: x else: y`? Statements were a mistake in programming language history: expressions are universally superior. ¶
- **Implicit return values from functions.** In Lisp, every function automatically returns the value of the last expression evaluated, and you (usually, some Lisps are different) have to use a continuation or some other kind of explicit-return mechanism to break out early; in most imperative languages like C and Python you have to use `return` to break out of a function and specify its return value. Ruby gives you the best of both worlds: you can use `return` to return early, but functions still implicitly return the value of the last expression. ¶
- **Case.** `case/switch` blocks have a bad rap because C screwed them up. Even Ruby's implementation leaves something to be desired, but at least it has them, without the opportunity to make yourself look like a fool by forgetting a `break`. Even worse, Python had a proposal for a switch/case statement, but it was rejected! ¶
- **Hashable and unhashable types.** Ruby lets you call `hash` on anything, regardless of whether it really makes sense to use it as a hash key; this allows it to be used to speed up comparisons. Python only allows it to be used on immutable data types, which kind of

makes sense, but really makes assumptions which are not always guaranteed to be true. ¶

- **Private methods and attributes.** ¶
- **Mutable strings.** Not having these is occasionally a pain. ¶
- **Ordered dictionaries (Hashes) and sets.** This doesn't have to involve as much overhead as you'd think. ¶
- **List joining.** `list.join(string)` (the Ruby way) makes a ton more sense than `string.join(list)` (the Python way). ¶
- **Naming of "special functions."** Python is extremely conservative with what it allows you to put in names. You can't have `+` as a name like you can in Lisp or Ruby, so to overload the addition operator, you define a method called `__add__`. There are enough of these that I find myself referring to the documentation of them fairly frequently, whereas in Ruby I know to define a method called `+`; other 'overloading' methods are similarly intuitively named. (Though it does look quite odd in code, I admit, though you soon get used to it.) ¶

Ways in which Ruby and Python both suck

- **HTTP client support.** `urllib` sucks; `Net::HTTP` does too. I'm assured that there are third-party replacements for both available, which don't suck, but it's 2013, and good HTTP support should come packaged with every language. Built-in `libcurl` bindings would be terrific. (Something PHP actually managed to get right …) ¶

— *David Kendal, January 2013*