

New to Python: my impression v. Perl/Ruby

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-01/2323.html>

From: Wayne Folta (wfolta_at_netmail.to)

Date: 01/19/04

Date: Sun, 18 Jan 2004 22:47:26 -0500

To: python-list@python.org

I've been a long-time Perl programmer, though I've not used a boatload of packages nor much of the tacky OO.

A couple of years ago, I decided to look into Python and Ruby. Python looked OK, but not that different. I did like the indent-as-group idea, which was different. Ruby looked very cool. But it was impossible to get good documentation. It seemed like a Japanese cult with a few western initiates.

Well, MacOS X ships with Perl, Python, and Ruby (and PHP, and ...) so I recently figured I'd try them again. I still find Ruby more intriguing, but I've settled on Python. Why?

Well, Perl is an expedient measure that (for me) doesn't scale up and has that horrible OO syntax. (So I never used it much.) If you're going to read someone else's Perl, you had better be "in the zone".

A couple of months ago, I'd written a quick Perl script that would "unstick" a co-worker's POP email. The problem is the Windows-based POP server is too stupid to realize a message does not end in CRLF, so it just appends a period then CRLF thinking it's doing well. But the period ends up not being alone on a line, so it's not a valid termination to the message. So their email program hangs waiting for a proper termination. Sigh.

A week ago, the script broke because I hadn't properly accounted for the fact that the user might have hundreds of messages queued up. (I hadn't used a Perl POP package, which might've handled it, but just threw it together myself. Yes, that would've made the Perl code more competitive with the other two solutions.)

So I decided this might be a nice exercise to try Python. And it went together very quickly using Python's POP3 package. Then I decided to try it in Ruby. I think one little portion of the code shows the difference between the two cultures.

The problem is that the message is not properly terminated, so you need

to time out and catch that timeout to realize, "hmmm, malformed". In Python I had:

```
M = poplib.POP3 ('172.16.30.1') ;  
M.user ('foo') ;  
M.pass_ ('bar')
```

```
num_mesgs = len (M.list ()[1])  
bad_mesgs = 0
```

```
for msg in range (num_mesgs):  
    try:  
        M.retr (msg + 1)  
... blah blah...
```

How do I get it to time out after 5 seconds and how do I catch that? The online docs are pretty good, but I had to guess that POP3 was built on the socket package. Looking at socket's docs I found the proper command and exception. I had to include:

```
socket.setdefaulttimeout (5.0)
```

before the POP3 commands to set the default socket timeout to 5 seconds. And

```
except socket.timeout:
```

is the proper way to catch the timeout. Both of these things are in the socket documentation.

Contrast this with Ruby. The Ruby docs are less complete, but they did mention that POP was subclassed from protocol and you'd have to look at protocol's source to see how it works. Looking through protocol, I figured out what to do and it was more elegant.

The protocol class had a read_timeout, but since Ruby's mantra might be said to be "Real OO", the POP code had been written such that you could say

```
pop.read_timeout = 5
```

after the POP open and it set the timeout for that pop connection to 5 seconds. Almost as if POP passed the read_timeout upstream to socket automatically. (I don't think Ruby does, but it's coded to look that way.)

My experience is limited, but it feels like this example gives a good feel for the two languages. Python was better documented and things came together quickly. Ruby ultimately had a more elegant solution, but was more poorly documented. This echoes the mantras: Python's is "Batteries Included", while Ruby's might be "Real OO".

On a personal note, I usually prefer an elegant solution, but when the language goes so far as to consider

```
0.step(360, 45)
```

to be reasonable, my head hurts. I know there's syntactic sugar so I don't have to code like that. I know everything's an object. But, dammit, a constant integer is an integer with constant value and causing it to iterate is a step too far.