Python for Perl Hackers

Mark Eichin, SIPB

IAP 2004

Why are you here?

- perl approaches line noise if not disciplined
- perl is subtle and quick to anger
- code is read more than written
- power is still important

Who is this guy?

- perl since perl 3 (ref crypto)
- author of perl journal x-in-perl
- sipb, -c help
- other startup perl stuff

Why

- How I got here

  qmtest

  some code at work

  indentation

- why evangelize

Rigor

- Compare construct by construct
- Especially if you had real metrics for readability
- Possibly based on classes of error
- All you’re getting tonight is anecdotes and examples

General Readability

- whitespace just isn't that hard to work with
- comment-formatting tool

Tools/scripts/pindent.py

```
def foobar(a, b):
   if a == b:
       a = a+1
   else:
       print 'oops!'
   # end if
# end def foobar
```

More Readability

- common string functions: startswith, endswith, strip, lower, replace...
- even one liners can be clear

```
perl -e ’print join("\n",@INC)’
python -c ’import sys; print sys.path’
python -c ’import sys; print "\n".join(sys.path)’
```

Subroutines

- perlintro:

```
sub log {
    my $logmessage = shift;
    print LOGFILE $logmessage;
}
```

- in python:

```
def log(logmessage):
    print >> LOGFILE, logmessage
```

- subroutines have arguments, not an afterthought
- local (not perl-local, perl-my) variables
- later example:

```
my ($logmessage, $priority) = @_;       # common
```

does even less about noticing caller errors

Modules

- The simplest module mentioned in perlnewmod:

```
package Text::Tabs;
require Exporter;
@ISA = (Exporter);
@EXPORT = qw(expand unexpand $tabstop);
use vars qw($VERSION $tabstop $debug);
$VERSION = 98.112801;
```

- 'package' and Exporter bits just go away
- there isn't a version convention
- there is a __doc__ convention, just put in string
- a top level __version__ would server
- err, .expandtabs is a string method anyway

Module Hierarchy

- don’t care if you inherit from
- just care that you provide readline.
- namespace protection is there
- But, you can use from Foo import * to get around it
- (if it really makes things more readable.)
- explicit export-list control is available

Exceptions

- try/except/finally
- raise (old string form, superceded by class-form)
- base-class-except lets you have specialized exceptions

```
class FooWarning(Warning): pass
class BarError(Error): pass
try: this
except Warning: whine()
```

Pack/Unpack

- "import struct" and struct.pack, struct.unpack
- explicit struct.calcsize!
- Arguments actually have to match
- native vs. endian is modifier, not encoded in args
- explicit native vs. little vs. big.

Unicode

- "just there" since 1.6
- perl needs a "use charnames;" pragma
- possibly with the ":full" tag
- and at least perl 5.6

Hashes

- "dictionaries", or "dicts"
- d.keys(), d.values(), d.items()
- vs. keys(%d)
- "tied hashes": class with __getitem__
- dbmopen/DB-tied hashes: "import shelve"

String Interpolation

- "$foo" is pretty powerful
- except when it doesn’t work at all
  (hashes with quoted string key)
- Python uses a % operator (C++ STL-like)
- Usual tricks (%s prints pretty much anything)
- named element lookup

```
>>> "%(hi)s" % { "foo": 2, "hi": "world"}
’world’
```

- combined with locals(), globals() can be excessive

Regexps

- most of what you'd expect
- python has named groups
- perl only has EXPERIMENTAL "(?{ code })"

```
$_ = "The brown fox jumps over the lazy dog";
/the (\S+)(?{ $color = $^N }) (\S+)(?{ $animal = $^N })/i;
print "color = $color, animal = $animal\n";
```

- vs.

```
s = "The brown fox jumps over the lazy dog"
m = re.search("the (?P<color>\S+) (?P<animal>\S+)", s, re.I)
print "color = %(color)s, animal = %(animal)s" % m.groupdict()
```

Loop Constructs

```
for thing in things_to_search:
   if predicate(thing):
       do_something_with(thing)
       break
else:
   print "didn’t find a thing that is predicately"
```

- which is often what you *mean*

C modules:

- Not too hard in either
- python seems to keep the parts in one place a bit more
- yet another zephyr module.
- Surprisingly little need for them so far

Batteries Included

- Builtin modules that everyone will have
- urllib and gzip
- "os" package is quite rich for posix, at least.

Aspect Oriented

- Desperately hookable
- stuff fixes in to existing classes

```
import gzip
save_init_read = gzip.GzipFile._init_read
def fix_init_read(self):
    save_init_read(self)
    self.size = long(self.size)
gzip.GzipFile._init_read = fix_init_read
```

- make unconfigurable logging functions shut up

Wrapping

```
class Collections(CollectionsRaw):
    """Locking wrappers around raw collection functions"""
    def __init__(self, *args):
        CollectionsRaw.__init__(self, *args)
        # operations that need locking
        self.reserve_name = self.lock_wrapper(CollectionsRaw.reserve_na
    def lock_wrapper(self, wrapped_fn):
        def lock_inner(*args):
            self.lock()
            try:
                self.update()
                wrapped_fn(self, *args)
                self.flush()
            finally:
                self.unlock()
        return lock_inner
    def lock(self):
        self.lockfile.lock()
```

Numbers

- perl: BigInt etc. classes
- python: builtin "small" integers (32 bitsigned)
- and "long" (arbitrary length) integers
- also (double) floats
- 2.2 and later, int to long autopromotion

Other

- "sequence unpacking": sequences can assign piecewise:

```
size, value = fun(path, op)
for k,v in d.items(): n[k.lower()] = v
```

References

http://www.python.org/pypi

http://mechanicalcat.net/cgi-bin/log/python/anti-p

http://www.mit.edu/iap/2004/python-for-perl/index.

http://zephyrfalcon.org/weblog/arch_d7_2003_10_25.

sipb-iap-python-for-perl@mit.edu

http://www.thok.org/intranet/python/index.html

Appendix: sample slide

= Pack/Unpack
* "import struct" and struct.pack, struct.unpack
* explicit struct.calcsize, which I’ve always had to kludge.
* Arguments actually have to match
* native vs. endian is modifier, not encoded in the individual args
* explicit native vs. little vs. big.

Appendix: code

```
#!/usr/bin/python
import sys
def entity_quote(txt):
    return txt.replace("&", "&amp;").replace("<", "&lt;").replace(">"
def wrap(elem, txt):
    return "<%s>%s</%s>" % (elem, entity_quote(txt), elem)
```

Appendix: one slide

```
def do_a_slide(f):
    print "<slide>"
    for line in f:
        if line == "\n":
            print "</slide>"
            break
        elif line.startswith("= "):
            print wrap("header", line[2:].strip())
        elif line.startswith("* "):
            print wrap("bullet", line[2:].strip())
        elif line.startswith("@ "):
            print wrap("url", line[2:].strip())
        else:
            sys.exit("Slide Huh?" + line)
    else:
        raise EOFError()
```

Appendix: header

```
def do_a_header(f):
    print "\n".join(["<?xml version=’1.0’?>"
                     ’<!DOCTYPE slideshow SYSTEM "xslides.dtd">’
                     ’<slideshow>’])
    for line in f:
        if line == "\n":
            break
        elif line.startswith("TITLE: "):
            print wrap("title", line.lstrip("TITLE:").strip())
        elif line.startswith("AUTHOR: "):
            print wrap("author", line.lstrip("AUTHOR:").strip())
        elif line.startswith("DATE: "):
            print wrap("date", line.lstrip("DATE:").strip())
        else:
            sys.exit("Header Huh?" + line)
```

Appendix: footer, main

```
def do_a_footer(f):
    print "\n".join(["</slide>", "</slideshow>"])
if __name__ == "__main__":
    f = sys.stdin
    do_a_header(f)
    try:
        while 1:
            do_a_slide(f)
    except EOFError:
        do_a_footer(f)
```