Tales of rejected PEPs

Jacek Szpot

and @maligree after hours
me!

me me me me me me me me me me

let's talk about me
PEPs
What the hell are PEPs
python enhancement proposals
python enhancement proposals

of course you know this
I am a little upset.
CELEBRITY QUIZ
Who's your daddy?
Who's your daddy?
BDFL
Benevolent Dictator for Life
idea + champion + github PR = new pep draft
What's the PEP that everyone knows?
PEP8
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
Sparse is better than dense.

In the face of ambiguity, refuse the temptation to guess.
“Don't try to stick too much code on one line.”
No one knows about the other one.
Let’s talk about pepes peps!
# PEP 504 -- Using the System RNG by default

<table>
<thead>
<tr>
<th>PEP:</th>
<th>504</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Using the System RNG by default</td>
</tr>
<tr>
<td>Author:</td>
<td>Nick Coghlan &lt;ncoghlan at gmail.com&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Withdrawn</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created:</td>
<td>15-Sep-2015</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.6</td>
</tr>
<tr>
<td>Post-History:</td>
<td>15-Sep-2015</td>
</tr>
</tbody>
</table>
Abstract

Python currently defaults to using the deterministic Mersenne Twister random number generator for the module level APIs in the random module, requiring users to know that when they're performing "security sensitive" work, they should instead switch to using the cryptographically secure os.urandom or random.SystemRandom interfaces or a third party library like cryptography.

Unfortunately, this approach has resulted in a situation where developers that aren't aware that they're doing security sensitive work use the default module level APIs, and thus expose their users to unnecessary risks.

serious business?

os.urandom / random.SystemRandom

over

random.* whatever
import random

>>> random._inst
<random.Random object at 0x7fc536051618>

# PEP proposed: make _inst a SystemRandom instance
# by default and provide a call to
# switch to random.Random:
# random.ensure_repeateable()

# oooh and this has a performance impact
PEP 506 -- Adding A Secrets Module To The Standard Library

<table>
<thead>
<tr>
<th>PEP:</th>
<th>506</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Adding A Secrets Module To The Standard Library</td>
</tr>
<tr>
<td>Author:</td>
<td>Steven D'Aprano &lt;steve at pearwood.info&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Accepted</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created:</td>
<td>19-Sep-2015</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.6</td>
</tr>
<tr>
<td>Post-History:</td>
<td></td>
</tr>
</tbody>
</table>
15.3. `secrets` — Generate secure random numbers for managing secrets

*New in version 3.6.*

**Source code:** `Lib/secrets.py`

The `secrets` module is used for generating cryptographically strong random numbers suitable for managing data such as passwords, account authentication, security tokens, and related secrets.

In particular, `secrets` should be used in preference to the default pseudo-random number generator in the `random` module, which is designed for modelling and simulation, not security or cryptography.

**See also:** [PEP 506](https://www.python.org/dev/peps/pep-0506/)
depressingly civil and polite and nice.
**PEP 666 -- Reject Foolish Indentation**

<table>
<thead>
<tr>
<th>PEP:</th>
<th>666</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Reject Foolish Indentation</td>
</tr>
<tr>
<td>Author:</td>
<td>lac at strakt.com (Laura Creighton)</td>
</tr>
<tr>
<td>Status:</td>
<td>Rejected</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created:</td>
<td>3-Dec-2001</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Post-History:</td>
<td>5-Dec-2001</td>
</tr>
</tbody>
</table>
- `python -TNone` will refuse to run when there are any tabs.
- `python -Tn` will refuse to run when tabs are not exactly `n` spaces.
- `python -Tonly` will refuse to run when blocks are indented by anything other than tabs.

People who mix tabs and spaces, naturally, will find that their programs do not run. Alas, we haven't found a way to give them an electric shock as from a cattle prod remotely. (Though if somebody finds out a way to do this, I will be pleased to add this option to the PEP.)

This proposal, if accepted, will probably mean a heck of a lot of work for somebody. But since I don't want it accepted, I don't care.
PEP 3117 -- Postfix type declarations

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEP:</td>
<td>3117</td>
</tr>
<tr>
<td>Title:</td>
<td>Postfix type declarations</td>
</tr>
<tr>
<td>Author:</td>
<td>Georg Brandl &lt;georg at python.org&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Rejected</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created:</td>
<td>01-Apr-2007</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.0</td>
</tr>
<tr>
<td>Post-History:</td>
<td></td>
</tr>
</tbody>
</table>
Therefore, this PEP combines the move to type declarations with another bold move that will once again prove that Python is not only future-proof but future-embracing: the introduction of Unicode characters as an integral constituent of source code.

Instead of the single Unicode character, they can type:\$\text{UNICODE NAME OF THE DECLARATOR}\$. For example, these two function definitions are equivalent:

```python
def foo\$x\$:
    return None

and

def foo\$\text{LATIN SMALL LETTER LAMBDA WITH STROKE}\$(\$x\$\$\text{DOUBLE-STRUCK CAPITAL C}\$):
    return None\$\text{ZERO WIDTH NO-BREAK SPACE}\$
```

This is still easy to read and makes the full power of type-annotated Python available to ASCII believers.
def normpathA(path\%):%:
    """Normalize path, eliminating double slashes, etc."""
    if path\% == '\':
        return '\'

    initial_slashes/= = path\%.startswithA('(//)\v
    # POSIX allows one or two initial slashes, but treats three or more
    # as single slash.
    if (initial_slashes/= and
        path\%.startswithA('//\v and not path\%.startswithA('///\v)):v
        initial_slashesN = 2
    
    comps//= = path\%.splitA('://\v
    new_comps//= = []
    for comp\% in comps//=:
        if comp\% in ('\', '\.')(t):
            continue
        if (comp\% != '..' or (not initial_slashesN and not new_comps//=) or
            (new_comps//= and new_comps//=[-1]% == '..')):
            continue

SAY NO TO DRUGS
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEP</td>
<td>3125</td>
</tr>
<tr>
<td>Title</td>
<td>Remove Backslash Continuation</td>
</tr>
<tr>
<td>Author</td>
<td>Jim J. Jewett &lt;JimJJewett at gmail.com&gt;</td>
</tr>
<tr>
<td>Status</td>
<td>Rejected</td>
</tr>
<tr>
<td>Type</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created</td>
<td>29-Apr-2007</td>
</tr>
</tbody>
</table>
I'M HELPING
<table>
<thead>
<tr>
<th>PEP:</th>
<th>394</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>The &quot;python&quot; Command on Unix-Like Systems</td>
</tr>
<tr>
<td>Author:</td>
<td>Kerrick Staley &lt;mail at kerrickstaley.com&gt;, Nick Coghlan &lt;ncoghlan at gmail.com&gt;, Barry Warsaw &lt;barry at python.org&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Active</td>
</tr>
<tr>
<td>Type:</td>
<td>Informational</td>
</tr>
<tr>
<td>Created:</td>
<td>02-Mar-2011</td>
</tr>
<tr>
<td>Resolution:</td>
<td><a href="https://mail.python.org/pipermail/python-dev/2012-February/116594.html">https://mail.python.org/pipermail/python-dev/2012-February/116594.html</a></td>
</tr>
</tbody>
</table>
# PEP 374 -- Choosing a distributed VCS for the Python project

<table>
<thead>
<tr>
<th>PEP</th>
<th>374</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Choosing a distributed VCS for the Python project</td>
</tr>
<tr>
<td>Author</td>
<td>Brett Cannon &lt;brett at python.org&gt;, Stephen J. Turnbull &lt;stephen at xemacs.org&gt;, Alexandre Vassalotti &lt;alexandre at peadrop.com&gt;, Barry Warsaw &lt;barry at python.org&gt;, Dirkjan Ochtman &lt;dirkjan at ochtman.nl&gt;</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Type</td>
<td>Process</td>
</tr>
<tr>
<td>Created</td>
<td>07-Nov-2008</td>
</tr>
<tr>
<td>History</td>
<td>07-Nov-2008-22-Jan-2009</td>
</tr>
</tbody>
</table>
Okay let’s find some treasure.
git clone $pep_repo
# Find me some dirt!
$ git grep -i fuck
# nothing :(
$ git grep -i shit
# nothing :( 
$ git grep -i dick
# …
$ git grep -i dick
#
…

pep-3133.txt: express the works of Charles Dickens in Python: ::
pep-3133.txt: >>> from dickens import Urchin, Gentleman
express the works of Charles Dickens in Python:

```
>>> from dickens import Urchin, Gentleman
```

```
$ git grep -i dick
# ...
```

pep-3133.txt:express the works of Charles Dickens in Python: ::
pep-3133.txt: >>> from dickens import Urchin, Gentleman
C’est la vie.
I look for profanity, I find Dickens.
With the invention of both dogs and trees, we were no longer able to …

$ git grep -i dogs
# ...

pep-3133.txt:With the invention of both dogs and trees, we were no longer able to …
WHAT'S GOING ON IN THIS THR-

OH LAWD
<table>
<thead>
<tr>
<th>PEP:</th>
<th>3133</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Introducing Roles</td>
</tr>
<tr>
<td>Author:</td>
<td>Collin Winter &lt;collinwinter at google.com&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Rejected</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Requires:</td>
<td>3115 3129</td>
</tr>
<tr>
<td>Created:</td>
<td>1-May-2007</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.0</td>
</tr>
<tr>
<td>Post-History:</td>
<td>13-May-2007</td>
</tr>
</tbody>
</table>
pies.szczeka()
Okay let’s look for envy.
$ git grep -i $other_lang
$ git grep -i php | wc -l
74
$ git grep -i "[^.]php" | wc -l
40
# PEP 505 -- None-aware operators

**PEP:** 505  
**Title:** None-aware operators  
**Author:** Mark E. Haase <mehaase at gmail.com>  
**Status:** Draft  
**Type:** Standards Track  
**Created:** 18-Sep-2015  
**Python-Version:** 3.6
Abstract

Several modern programming languages have so-called "null-coalescing" or "null-aware" operators, including C# [1], Dart [2], Perl, Swift, and PHP (starting in version 7). These operators provide syntactic sugar for common patterns involving null references.
criticism is that None-aware operators are akin to PHP's @ operator. Therefore, behavior around it.
Criticism is that None-aware operators are akin to PHP's @ operator. Therefore, behavior around it.
One criticism is that None-aware operators are easier to use than single-argument operators, akin to PHP's `@` operator. Therefore, it's common to define behavior around it.
criticism is that `None`-aware operators are not `None`-ary, akin to PHP's `@` operator. Therefore, avoid `None` around it.
One basicism is that None-aware operators are (  ) akin to PHP's @ operator. Therefore, the error around it.
The core of the problem is that None-aware operators are very ( ) akin to PHP's @ operator. Therefore, it is often around it.
None-aware operators do not have a direct equivalent in PHP's `@` operator. Therefore, we fall back on it.
Get None-aware operators.

PHP's `@` operator.
(that was 6 useless slides)
This operator is affectionately known by veteran phpers as the stfu operator.
git grep -i "[Pp]erl[^\(ly\|pod\|\.org\)|ink]\)" | wc -l
83
They even mention Perl 6!
Java, Perl, and JavaScript, treat a sequence of digits with Perl 6 (#perl6-s12) where it is called "roles", and it is
(Examples adapted from an article on Perl 6 roles by Curtis
Perl 6 allows instances to perform different roles than the
In Perl 6, this is done by creating an anonymous class that
 http://www.perlmonks.org/?node_id=384858
Perl 6 allows instances to...

In Perl 6, this is done by

http://www.perlmongers.org

http://dev.perl.org/perl
There is a Perl package installer also named "pip."
git grep -i "JavaScript" | wc -l
29
```bash
git grep -i "ECMAScript" | wc -l
9
```

Yes, `const cthulhu = { RECOMFILE\u0042: 42 }` is valid ECMAScript 2016.
other languages too
other languages too.
let’s wrap this up
what did we learn?
Questions?
the koniec

goodbye

Jacek Szpot

and @maligree after hours