



So You Think You Know Python?

Jordan Duabe
Pycon JP, 2026-08-21



Agenda

- Quirks and easter eggs
- Gotchas and real-world examples
- Quiz (?)

whoami



Jordan Duabe

Software Engineer at Anaqua

me@jduabe.dev

[@j4ckofalltrades@infosec.exchange](https://twitter.com/j4ckofalltrades)



Quirks and easter eggs



Finding your Zen



```
>>> import this
```

```
The Zen of Python, by Tim Peters
```

```
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!
```




PEP 20 – The Zen of Python

Abstract

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

PEP 20 – The Zen of Python

Author: Tim Peters <tim.peters at gmail.com>

Status: Active

Type: Informational

Created: 19-Aug-2004

Post-History: 22-Aug-2004

► Table of Contents

[Abstract](#)

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

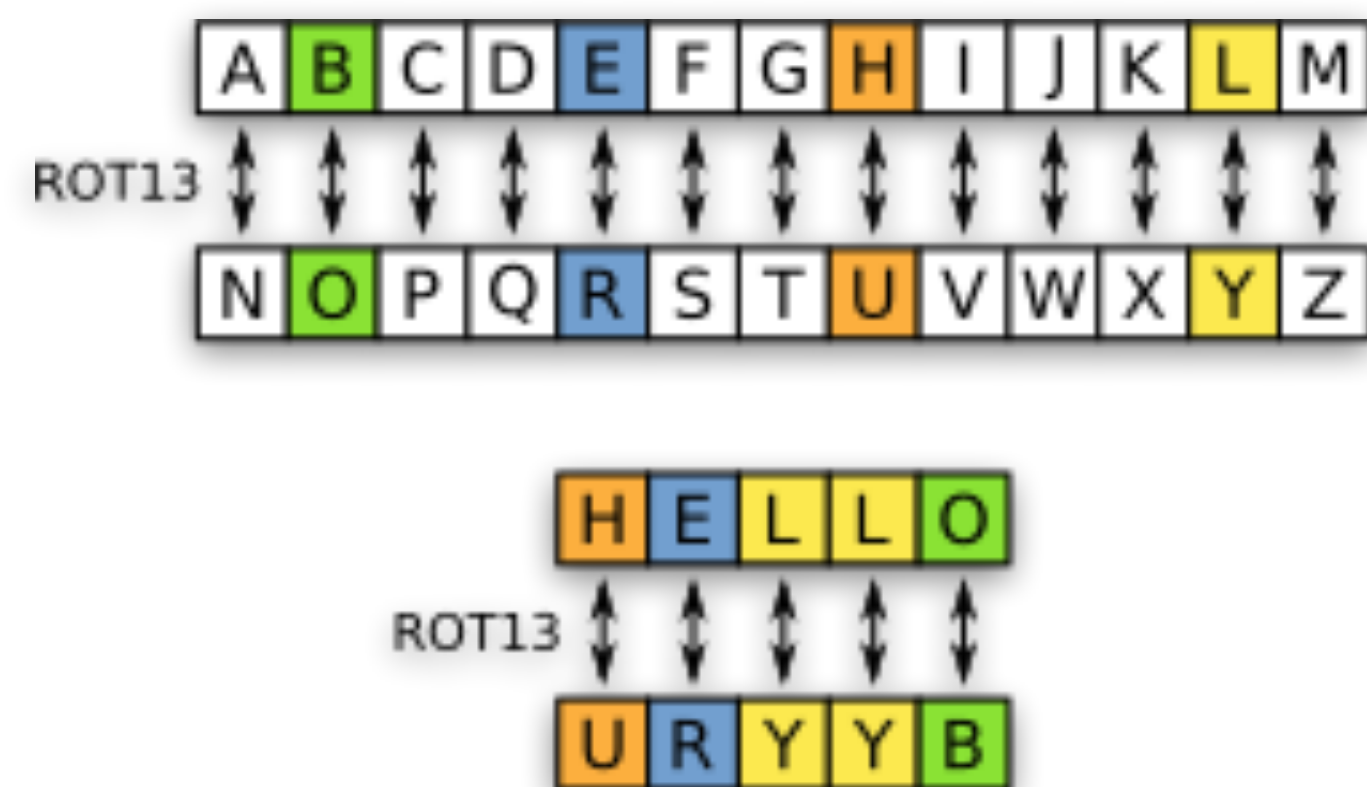
[The Zen of Python](#)

```
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!
```

[Easter Egg](#)

```
>>> import this
```


Zen of Python



```
Lib/this.py

s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpgv vf orggre guna vzcypvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcyngva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcyngva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```



Hello world!



```
>>> # < 3.11
```

```
>>> import __hello__
```

```
Hello world!
```

```
>>> import __phello__
```

```
Hello world!
```




Hello world!



```
>>> # >= 3.11
>>> import __hello__
>>> __hello__.main()
Hello world!
>>> import __phello__
>>> __phello__.main()
Hello world!
```

Regression: Importing `__hello__` doesn't print "Hello world!" #100136

🔒 Closed as not planned

 hugovk (Hugo van Kemenade) opened on Dec 9, 2022

Member ...

Bug report

Importing `__hello__` or `__phello__` on Python 3.7-3.10 prints "Hello world!"

Importing `__hello__` or `__phello__` on Python 3.11-3.12 prints nothing.

```
$ python3.7
Python 3.7.13 (default, Jul 8 2022, 20:05:46)
[Clang 13.1.6 (clang-1316.0.21.2.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import __hello__
Hello world!
>>> import __phello__
Hello world!
>>>
```





Breakfast of champions



```
>>> # >= 3.11
>>> import __phello__.spam
>>> __phello__.spam.main()
Hello world!
>>> import __phello__.ham
>>> import __phello__.ham.eggs
```


Frozen modules



Python/frozen.c

```
/* In order to test the support for frozen modules, by default we
   define some simple frozen modules: __hello__, __phello__ (a package),
   and __phello__.spam. Loading any will print some famous words... */

static const struct _frozen test_modules[] = {
    {"__hello__", _Py_M____hello__, (int)sizeof(_Py_M____hello__), false},
    {"__hello_alias__", _Py_M____hello__, (int)sizeof(_Py_M____hello__), false},
    {"__phello_alias__", _Py_M____hello__, (int)sizeof(_Py_M____hello__), true},
    {"__phello_alias__.spam", _Py_M____hello__, (int)sizeof(_Py_M____hello__), false},
    {"__phello__", _Py_M____phello__, (int)sizeof(_Py_M____phello__), true},
    {"__phello__.__init__", _Py_M____phello__, (int)sizeof(_Py_M____phello__), false},
    {"__phello__.ham", _Py_M____phello__ham, (int)sizeof(_Py_M____phello__ham), true},
    {"__phello__.ham.__init__", _Py_M____phello__ham, (int)sizeof(_Py_M____phello__ham), false},
    {"__phello__.ham.eggs", _Py_M____phello__ham_eggs, (int)sizeof(_Py_M____phello__ham_eggs), false},
    {"__phello__.spam", _Py_M____phello__spam, (int)sizeof(_Py_M____phello__spam), false},
    {"__hello_only__", _Py_M__frozen_only, (int)sizeof(_Py_M__frozen_only), false},
    {0, 0, 0} /* test sentinel */
}
```



Frozen modules

```
Lib/test/test_frozen.py

"""Basic test of the frozen module (source is in Python/frozen.c)."""

# The Python/frozen.c source code contains a marshalled Python module
# and therefore depends on the marshal format as well as the bytecode
# format. If those formats have been changed then frozen.c needs to be
# updated.

class TestFrozen(unittest.TestCase):
    def test_frozen(self):
        name = '__hello__'
        if name in sys.modules:
            del sys.modules[name]
        with import_helper.frozen_modules():
            import __hello__
        with captured_stdout() as out:
            __hello__.main()
        self.assertEqual(out.getvalue(), 'Hello world!\n')
```




PEP 401 – BDFL Retirement

Summary

The BDFL, having shepherded Python development for 20 years, officially announces his retirement effective immediately (Barry Warsaw aka Uncle Barry aka FLUFL).

His official title is now Benevolent Dictator Emeritus Vacationing Indefinitely from the Language (BDEVIL). Guido trains for his lifelong dream of climbing Mount Everest.

PEP 401 – BDFL Retirement

Author: Barry Warsaw, Brett Cannon

Status: April Fool!

Type: Process

Created: 01-Apr-2009

Post-History: 01-Apr-2009

► Table of Contents

Abstract

The BDFL, having shepherded Python development for 20 years, officially announces his retirement, effective immediately. Following a unanimous vote, his replacement is named.

Rationale

Guido wrote the original implementation of Python in 1989, and after nearly 20 years of leading the community, has decided to step aside as its Benevolent Dictator For Life. His official title is now Benevolent Dictator Emeritus Vacationing Indefinitely from the Language (BDEVIL). Guido leaves Python in the good hands of its new leader and its vibrant community, in order to train for his lifelong dream of climbing Mount Everest.

After unanimous vote of the Python Steering Union (not to be confused with the Python Secret Underground, which emphatically does not exist) at the 2009 Python Conference ([PyCon](#) 2009), Guido's successor has been chosen: Barry Warsaw, or as he is affectionately known, Uncle Barry. Uncle Barry's official title is Friendly Language Uncle For Life (FLUFL).



Not Equals



```
>>> from __future__ import barry_as_FLUFL
```

```
>>> 1 <> 2
```

```
True
```

```
>>> 1 != 2
```

```
File "<stdin>", line 1
```

```
    1 != 2
```

```
      ^
```

```
SyntaxError: invalid syntax
```


Brace yourselves



```
>>> from __future__ import braces
      File "<python-input-0>", line 1
        from __future__ import braces
                                ^^^^^^
SyntaxError: not a chance
```



Python/future.c

```
if (!feature)
    return 0;
if (strcmp(feature, FUTURE_NESTED_SCOPES) == 0) {
    continue;
}
// ...
} else if (strcmp(feature, "braces") == 0) {
    PyErr_SetString(PyExc_SyntaxError,
                    "not a chance");
    PyErr_RangedSyntaxLocationObject(filename,
                                     name->lineno,
                                     name->col_offset + 1,
                                     name->end_lineno,
                                     name->end_col_offset + 1);

    return 0;
}
```



Look away!



```
$ cat printer.by
def print_message(num_of_times) {
    for i in range(num_of_times) {
        print("Bython is awesome!");
    }
}

if __name__ == "__main__" {
    print_message(3);
}

$ uvx bython printer.by
Bython is awesome!
Bython is awesome!
Bython is awesome!
```



Don't leave me hanging



```
>>> r"Why can't raw strings end with a backslash?"
"Why can't raw strings end with a backslash?"
>>> r"Why can't raw strings end with a backslash? \"
File "<python-input-10>", line 1
    r"Why can't raw strings end with a backslash? \"
    ^
```


NOT like this



```
>>> True | False
True
```

```
>>> 1 | 0
1
```

```
>>> True & False
False
```

```
>>> 1 & 0
0
```

```
>>> ~True
```

```
<python-input-13>:1: DeprecationWarning: Bitwise inversion '~' on
bool is deprecated and will be removed in Python 3.16. This
returns the bitwise inversion of the underlying int object and is
usually not what you expect from negating a bool. Use the 'not'
operator for boolean negation or ~int(x) if you really want the
bitwise inversion of the underlying int.
```

```
-2
```

```
>>> ~False
```

```
<python-input-14>:1: DeprecationWarning: Bitwise inversion '~' on
bool is deprecated and will be removed in Python 3.16. This
returns the bitwise inversion of the underlying int object and is
usually not what you expect from negating a bool. Use the 'not'
operator for boolean negation or ~int(x) if you really want the
bitwise inversion of the underlying int.
```

```
-1
```



To Infinity and NaN



```
>>> # Python 2
>>> hash(float("Infinity")) # Pi
314159
>>> hash(float("-Infinity")) # Euler's number
-271828
>>> hash(float("NaN"))
0
```



```
>>> # Python 3
>>> hash(float("Infinity")) # Pi
314159
>>> hash(float("-Infinity"))
-314159
>>> # Python >= 3.10
>>> hash(float("NaN"))
8764389982649
```



π thon



```
(venv) user@machine$  $\pi$ thon
Python 3.14.3 (main, Feb 4 2026, 00:00:00) [GCC 15.2.1 20260123 (Red Hat 15.2.1-7)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```


π thon

gh-119535: Support π thon in Python 3.14 venvs #125035

Merged by [hugovk](#) on Oct 15, 2024 [main](#) ← [foreignmeloman:main](#) [v3.14.0a1](#)

Conversation 14 Commits 1 Checks 0 Files changed 1

All commits 0 / 1 viewed Submit comments

Lib/venv/__init__.py

↑

306

307

308

309

310

311

312

↓

@@ -306,7 +306,7 @@

.....suffixes = ['python', 'python3', f'python3.

{sys.version_info[1]}']

.....if sys.version_info[:2] == (3, 14):

.....suffixes.append('python π ')

.....for suffix in suffixes:

.....path = os.path.join(binpath, suffix)

.....if not os.path.exists(path):

306

307

308

309

310

311

312

.....suffixes = ['python', 'python3', f'python3.

{sys.version_info[1]}']

.....if sys.version_info[:2] == (3, 14):

.....suffixes.append('python π ')

.....for suffix in suffixes:

.....path = os.path.join(binpath, suffix)

.....if not os.path.exists(path):

Comment on line R309

minus π

gh-119535: 3.15 minus π #134037 Code Preview

Merged by [hauntsaninja](#) on May 15, 2025 main ← hugovk:3.15-de-egg 📄 👁 🔍 v3.15.0a1

💬 Conversation 2 🔗 Commits 2 📄 Checks 0 📄 Files changed 1 +2 -5

📄 All commits 0 / 1 viewed Submit comments ⚙️ 🔍 💬 0 👤

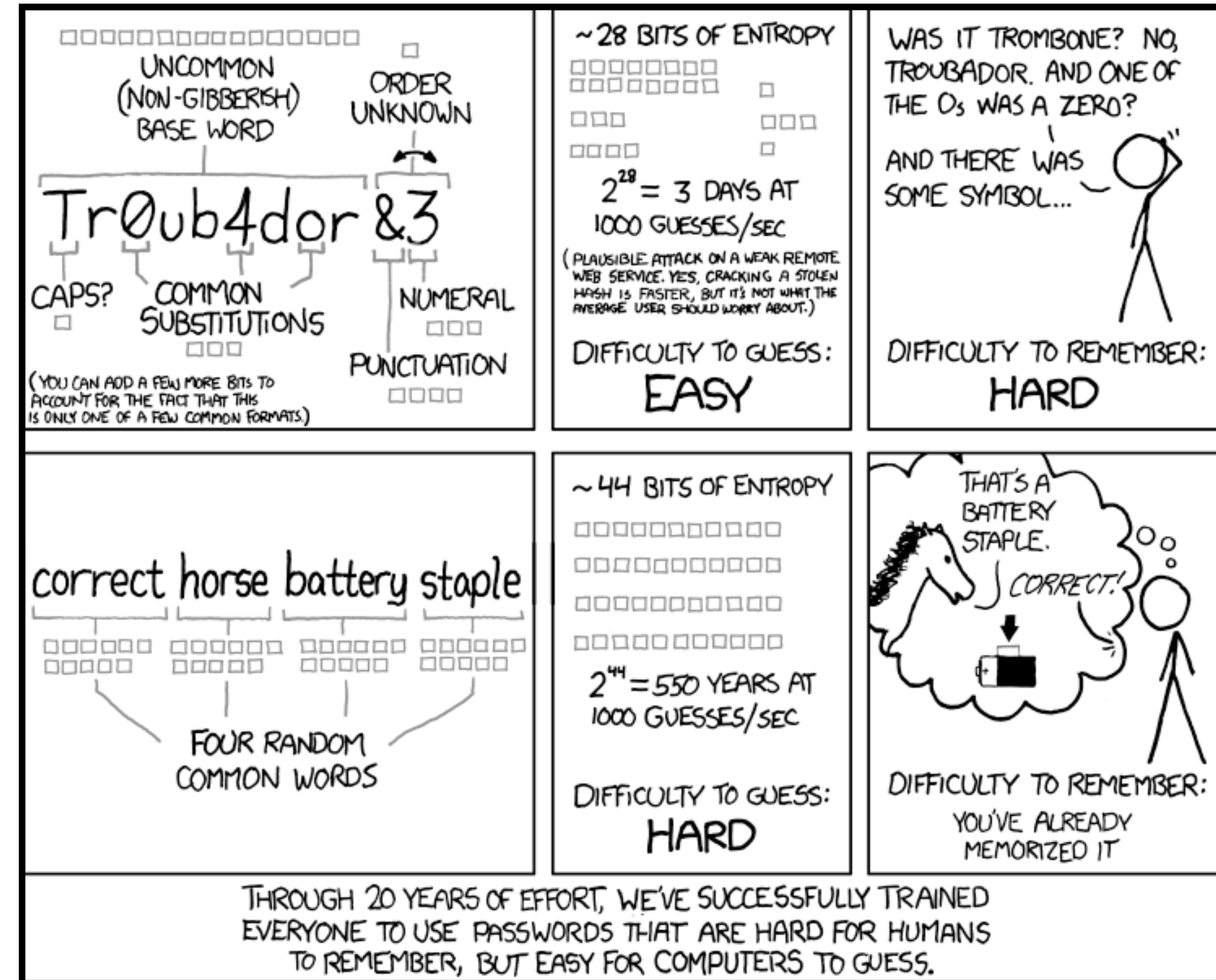
Lib/venv/__init__.py +2 -5 👤 Viewed 💬 ⋮

↑	@@ -313,11 +313,8 @@ def setup_python(self, context):	
313	copier(context.executable, path)	313copier(context.executable, path)
314	if not os.path.islink(path):	314if not os.path.islink(path):
315	os.chmod(path, 0o755)	315os.chmod(path, 0o755)
316	suffixes = ['python', 'python3', f'python3.	316for suffix in ('python', 'python3',
317	{sys.version_info[1]})	317f'python3.{sys.version_info[1]})':
318	if sys.version_info[:2] == (3, 14):	
319	suffixes.append('python')	
320	for suffix in suffixes:	
321	path = os.path.join(binpath, suffix)	318path = os.path.join(binpath, suffix)
322	if not os.path.exists(path):	319if not os.path.exists(path):
323	# Issue 18807: make copies if	320# Issue 18807: make copies if
↓		

XKCD 936

Generate an [XKCD-style passphrase](#):

```
import secrets
# On standard Linux systems, use a convenient dictionary file.
# Other platforms may need to provide their own word-list.
with open('/usr/share/dict/words') as f:
    words = [word.strip() for word in f]
    password = ' '.join(secrets.choice(words) for i in range(4))
```



XKCD 2200

```
Include/pymacro.h

#if defined(RANDALL_WAS_HERE)
#  define Py_UNREACHABLE() \
    Py_FatalError( \
        "If you're seeing this, the code is in what I thought was\n" \
        "an unreachable state.\n\n" \
        "I could give you advice for what to do, but honestly, why\n" \
        "should you trust me? I clearly screwed this up. I'm writing\n" \
        "a message that should never appear, yet I know it will\n" \
        "probably appear someday.\n\n" \
        "On a deep level, I know I'm not up to this task.\n" \
        "I'm so sorry.\n" \
        "https://xkcd.com/2200")
#elif defined(Py_DEBUG)
#  define Py_UNREACHABLE() \
    Py_FatalError( \
        "We've reached an unreachable state. Anything is possible.\n" \
        "The limits were in our heads all along. Follow your dreams.\n" \
        "https://xkcd.com/2200")
```

! ERROR

IF YOU'RE SEEING THIS, THE CODE IS IN WHAT I THOUGHT WAS AN UNREACHABLE STATE.

I COULD GIVE YOU ADVICE FOR WHAT TO DO. BUT HONESTLY, WHY SHOULD YOU TRUST ME? I CLEARLY SCREWED THIS UP. I'M WRITING A MESSAGE THAT SHOULD NEVER APPEAR, YET I KNOW IT WILL PROBABLY APPEAR SOMEDAY.

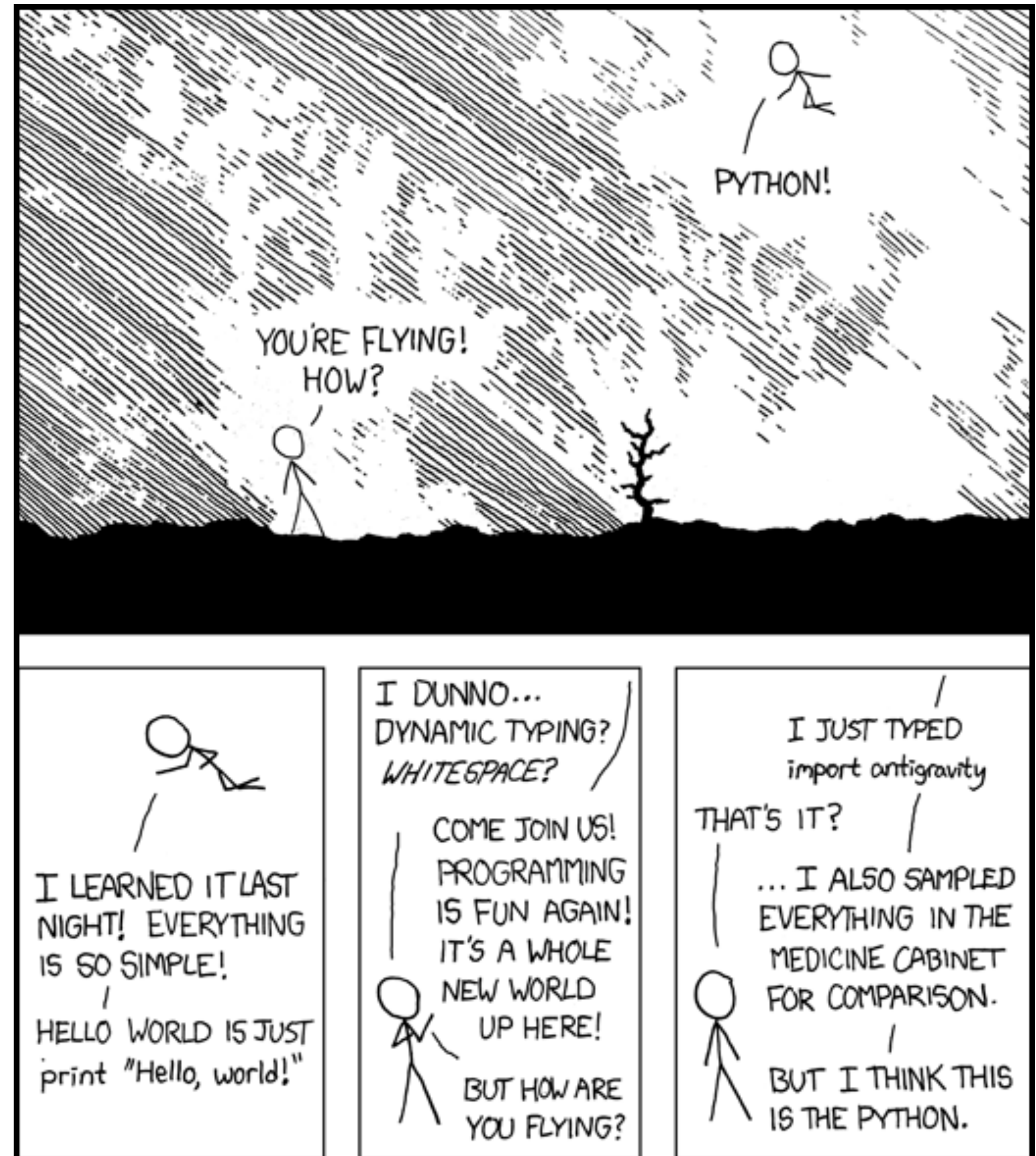
ON A DEEP LEVEL, I KNOW I'M NOT UP TO THIS TASK. I'M SO SORRY.



NEVER WRITE ERROR MESSAGES TIRED.

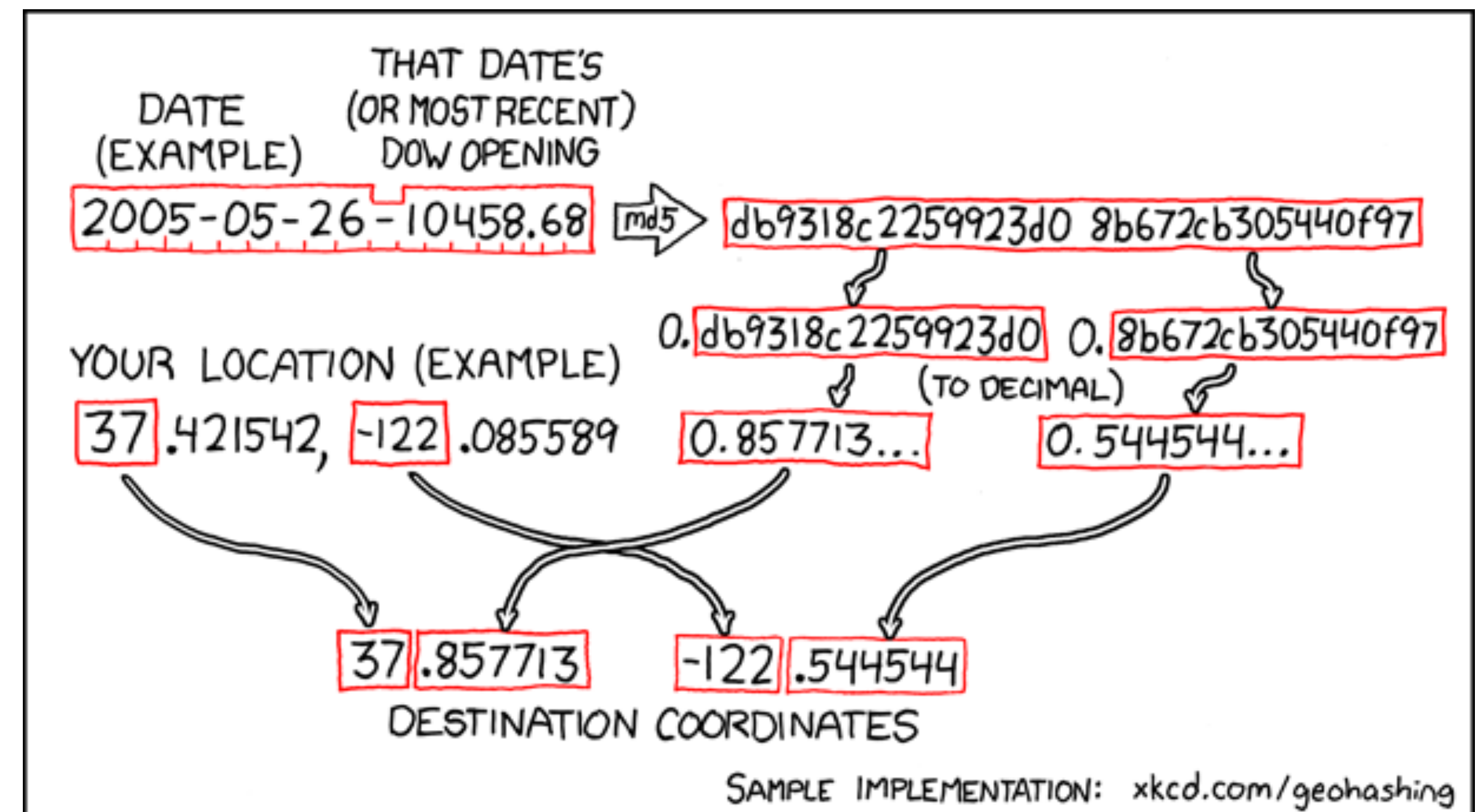
XKCD 353

```
>>> import antigravity
```



XKCD 426

```
>>> from antigravity import geohash
```



```
Lib/antigravity.py

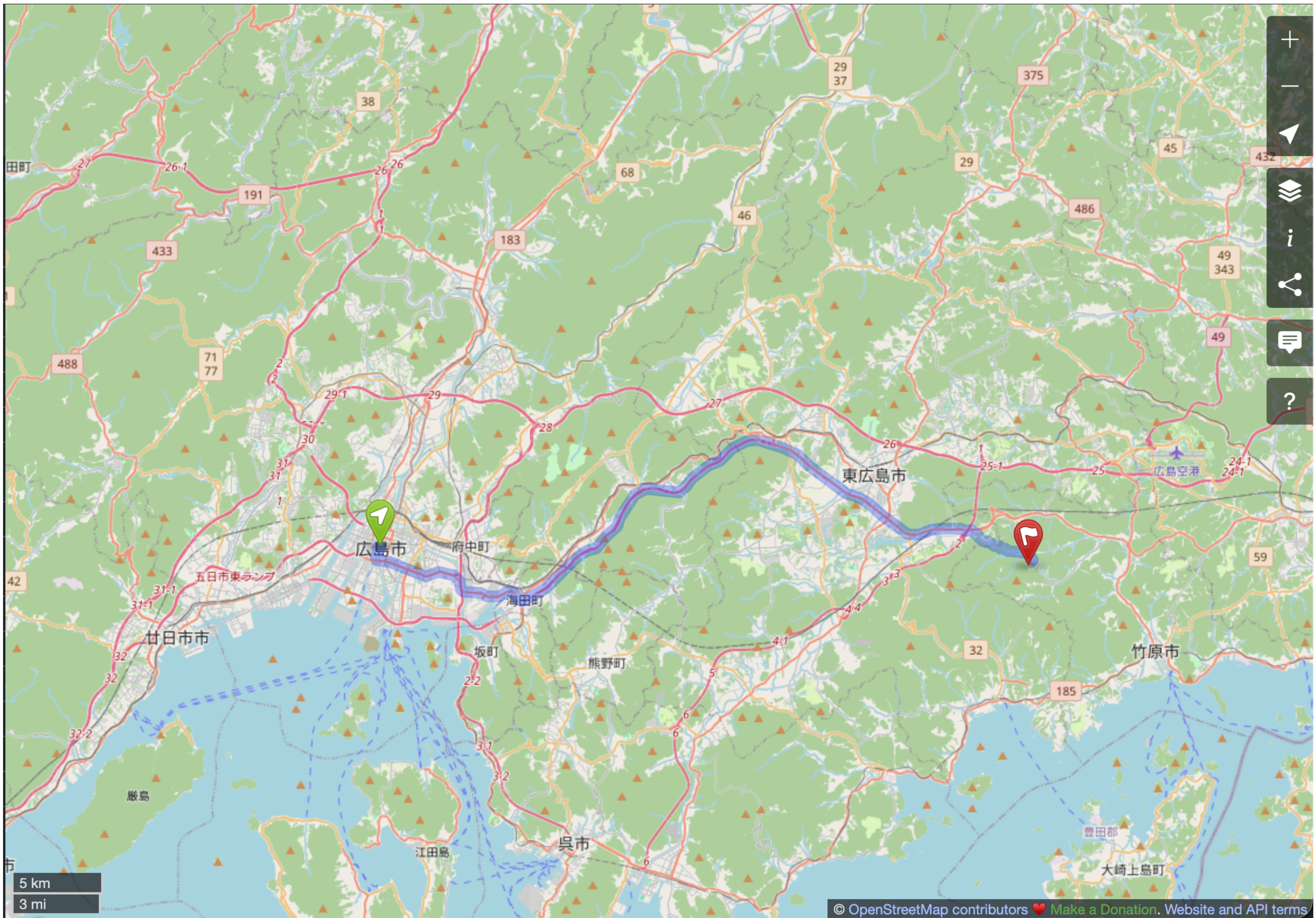
import webbrowser
import hashlib

webbrowser.open("https://xkcd.com/353/")

def geohash(latitude, longitude, datedow):
    '''Compute geohash() using the Munroe algorithm.

    >>> geohash(37.421542, -122.085589, b'2005-05-26-10458.68')
    37.857713 -122.544543

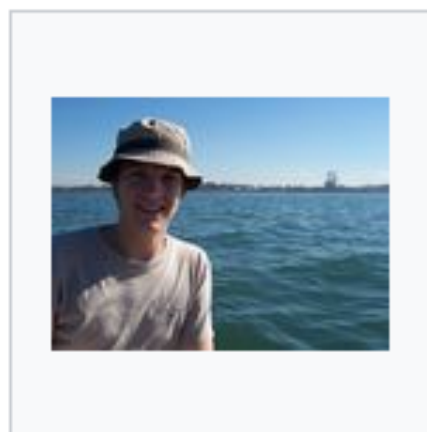
    '''
    # https://xkcd.com/426/
    h = hashlib.md5(datedow, usedforsecurity=False).hexdigest()
    p, q = [('%f' % float.fromhex('0.' + x)) for x in (h[:16], h[16:32])]
    print('%d%s %d%s' % (latitude, p[1:], longitude, q[1:]))
```

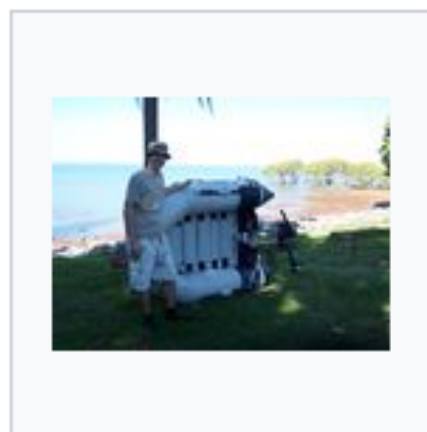
2008-06-06 -27 153

Moreton Bay, Redland Shire, Australia

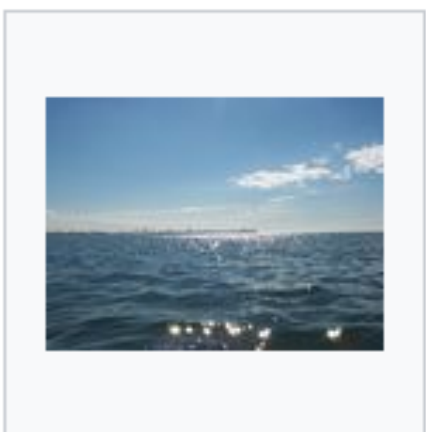
[Ubethatway](#) set out from the home at around 1pm with a tender for the [water geohash location](#) in Moreton Bay, just of Cleveland Point in the Redland Shire. It was a beautiful day on the bay, and the water geohash coordinates were successfully reached.



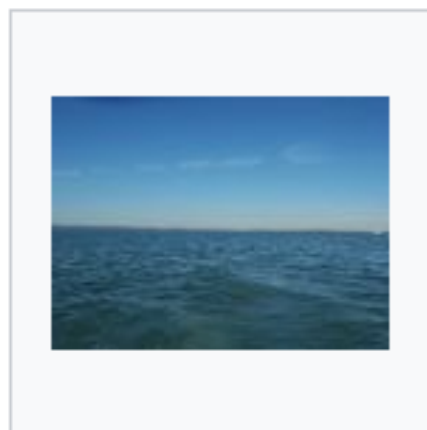
[Ubethatway](#) at the geohash



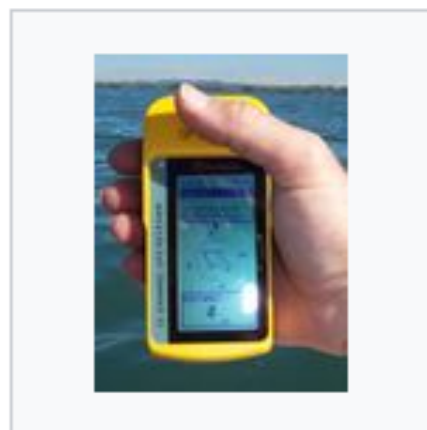
The craft



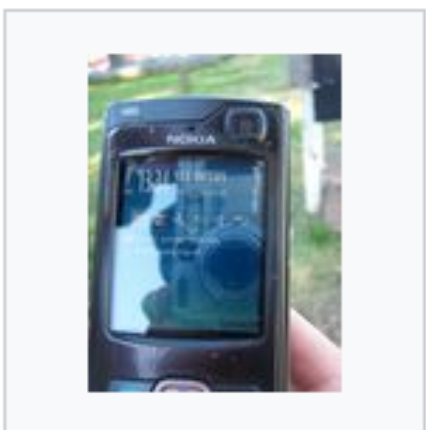
Cleveland Point from the geohash location



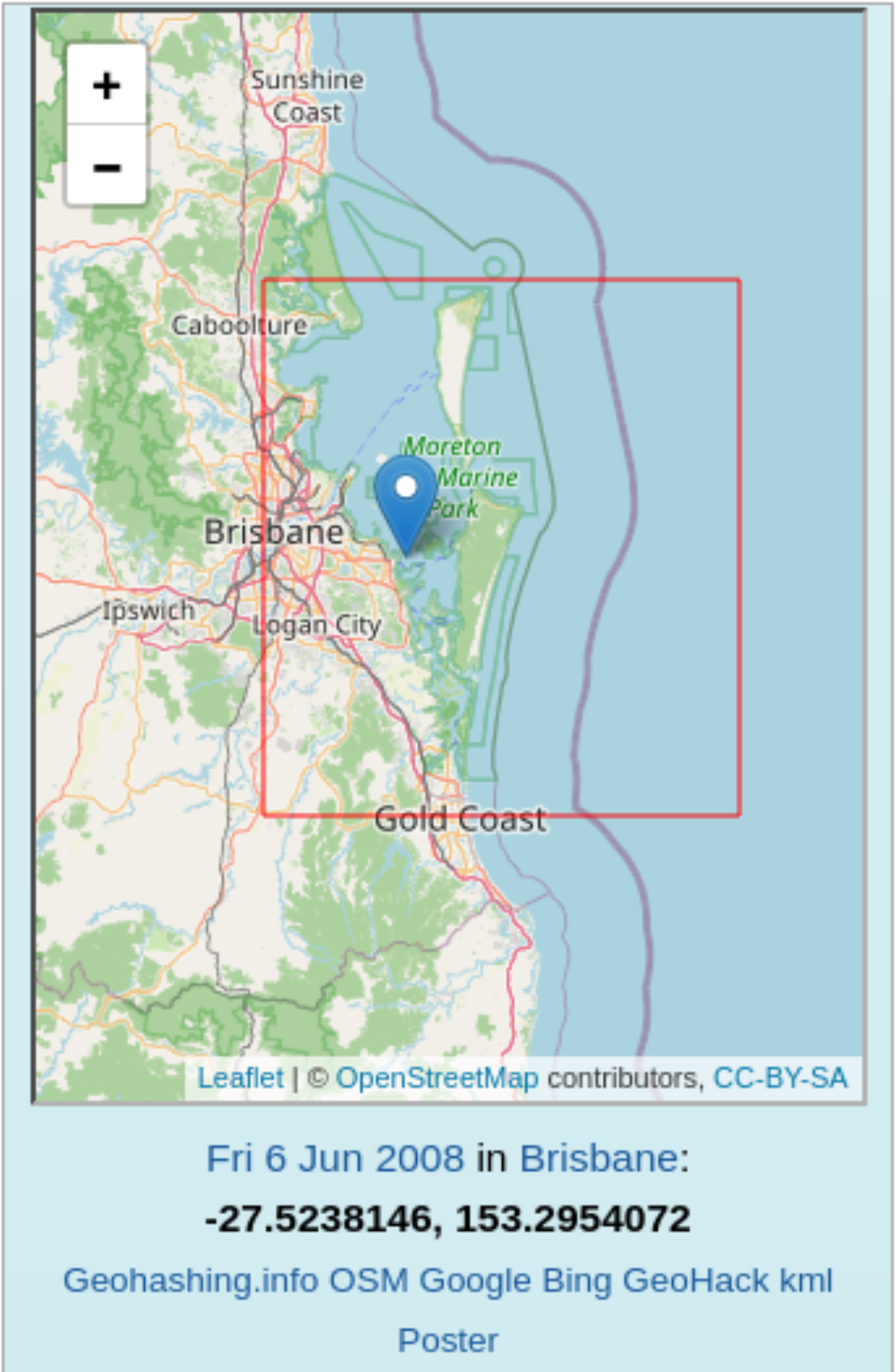
North Stradbroke and Peel Islands from the geohash location



GPS coordinates of the geohash



Time and date of the geohash



Gotchas and real-world examples

Is midnight falsy?



```
>>> # <= 3.4
>>> bool(datetime.time(0, 0, 0))
False
>>> bool(datetime.time(0, 0, 0))
True
```



```
>>> # >= 3.5
>>> bool(datetime.time(0, 0, 0))
True
>>> bool(datetime.time(0, 0, 0))
True
```



tim-one (Tim Peters) on Feb 4, 2012

Member



From the docs, at:

<http://docs.python.org/library/datetime.html#time-objects>

"""

in Boolean contexts, a time object is considered to be true if and only if, after converting it to minutes and subtracting utcoffset() (or 0 if that's None), the result is non-zero.

"""



Content

I disagree, I think this bug should be reopened and fixed.

A use case that I just ran into: I'm using a Django form with time fields that aren't required, but that are only valid in combination (if there's a open time there has to be a close time).

```
if not (self.closed or (self.open_time and self.close_time )):  
    raise ValidationError("Invalid times")
```

This leads to a Validation Error when using the times 12:00 and 00:00.

Of course, this case is debatable and can be worked around by using `self.open is not None and self.close is not None`, but there are even more problems when using the times inside the template.

I'm using the following widespread pattern inside my templates:

```
<p>Close time: {{ close_time|default:"- " }}</p>
```

The "default" filter used in this case displays the string "-" if the value on the left side of the | symbol evaluates to False. That makes sense in almost all of the cases.

In the case of the `datetime.time(0, 0)` object, the default value is displayed, even though `datetime.time(0, 0).\_\_str\_\_()` results in a valid string (in this case '00:00:00').

(By the way, through these experiments I also found a bug in Django's date formatting template function caused by this inconsistency, which I will report separately.)

I agree that casting time objects to a boolean value doesn't make much sense. But especially because of that, inconsistencies in the resulting boolean value should NOT exist. Yet another argument for this is that in many regions midnight isn't considered 00:00, but 24:00, which would obviously not evaluate to False.

Please fix this. Otherwise it will lead to a whole lot of weird bugs in software using the datetime library.



Re: Please reconsider the Boolean evaluation of midnight

If I had to do it over again I would *definitely* never make a time value "falsy". The datetime module was conceived long ago, when the dangers of falsy objects that weren't clearly trying to be "like numbers" or "like collections" weren't so clear. Today I would argue that times aren't enough "like numbers" to qualify.

--Guido van Rossum (python.org/~guido)

Without warning

```
resp = urllib3.request("GET", "https://example.com")

# Deprecated APIs
resp.getheader("Content-Length")
resp.getheaders()

# Recommended APIs
resp.headers.get("Content-Length")
resp.headers
```

```
src/urllib3/response.py

# Compatibility methods for http.client.HTTPResponse
def getheaders(self) → HTTPHeaderDict:
    warnings.warn(
        "HTTPResponse.getheaders() is deprecated and will be removed "
        "in urllib3 v2.1.0. Instead access HTTPResponse.headers directly.",
        category=DeprecationWarning,
        stacklevel=2,
    )
    return self.headers

def getheader(self, name: str, default: Optional[str] = None) → Optional[str]:
    warnings.warn(
        "HTTPResponse.getheader() is deprecated and will be removed "
        "in urllib3 v2.1.0. Instead use HTTPResponse.headers.get(name, default).",
        category=DeprecationWarning,
        stacklevel=2,
    )
    return self.headers.get(name, default)
```


Class	Description
Warning	This is the base class of all warning category classes. It is a subclass of Exception .
UserWarning	The default category for warn() .
DeprecationWarning	Base category for warnings about deprecated features when those warnings are intended for other Python developers (ignored by default, unless triggered by code in <code>__main__</code>).
SyntaxWarning	Base category for warnings about dubious syntactic features (typically emitted when compiling Python source code, and hence may not be suppressed by runtime filters)
RuntimeWarning	Base category for warnings about dubious runtime features.
FutureWarning	Base category for warnings about deprecated features when those warnings are intended for end users of applications that are written in Python.
PendingDeprecationWarning	Base category for warnings about features that will be deprecated in the future (ignored by default).
ImportWarning	Base category for warnings triggered during the process of importing a module (ignored by default).
UnicodeWarning	Base category for warnings related to Unicode.
BytesWarning	Base category for warnings related to bytes and bytearray .
ResourceWarning	Base category for warnings related to resource usage (ignored by default).

Restore getheaders() and getheader() #3732

Code

Preview

Merged on Dec 8, 2025 main ← illia-v:restore-methods 2.6.1

Conversation 2 Commits 1 Checks 41 Files changed 4

+16 -4

All commits

0 / 4 viewed

Submit comments



0



changelog/3731.feature.rst

+2



Viewed



...

docs/v2-migration-guide.rst

-2



Viewed



...

```
@@ -30,8 +30,6 @@ Here's a short summary of which changes in urllib3 2.x are most important:
30     - Removed the ``urllib3.contrib.ntlmpool`` module.
31     - Removed the ``urllib3.contrib.securetransport`` module.
32     - Removed the ``urllib3[secure]`` extra.
33     - Removed the ``HTTPResponse.getheaders()`` method in favor of ``HTTPResponse.headers``.
34     - Removed the ``HTTPResponse.getheader(name, default)`` method in favor of ``HTTPResponse.headers.get(name, default)``.
35     - Deprecated URLs without a scheme (ie 'https://') and will be raising an error in a future version of urllib3.
36     - Changed the default minimum TLS version to TLS 1.2 (previously was TLS 1.0).
37     - Changed the default request body encoding from 'ISO-8859-1' to 'UTF-8'.
```

src/urllib3/response.py

+7



Viewed



...

```
@@ -671,6 +671,13 @@ def readinto(self, b: bytearray) -> int:
671     671         .....b[: len(temp)] = temp
672     672         .....return len(temp)
673     673
674     674         ....# Methods used by dependent libraries
675     675         ....def getheaders(self) -> HTTPHeaderDict:
676     676         .....return self.headers
677     677
678     678         ....def getheader(self, name: str, default: str | None = None) -> str | None:
679     679         .....return self.headers.get(name, default)
680     680
681     681         ....# Compatibility method for http.cookiejar
682     682         ....def info(self) -> HTTPHeaderDict:
683     683         .....return self.headers
```



Solution



```
import warnings

warnings.warn(
    "HTTPResponse.getheader() is deprecated",
    category=FutureWarning,
    stacklevel=2
)
```

is equal



```
>>> a = 256
```

```
>>> b = 256
```

```
>>> a is b
```

```
True
```

```
>>> a == b
```

```
True
```

```
>>> c = 257
```

```
>>> d = 257
```

```
>>> c is d
```

```
False
```

```
>>> c == d
```

```
True
```

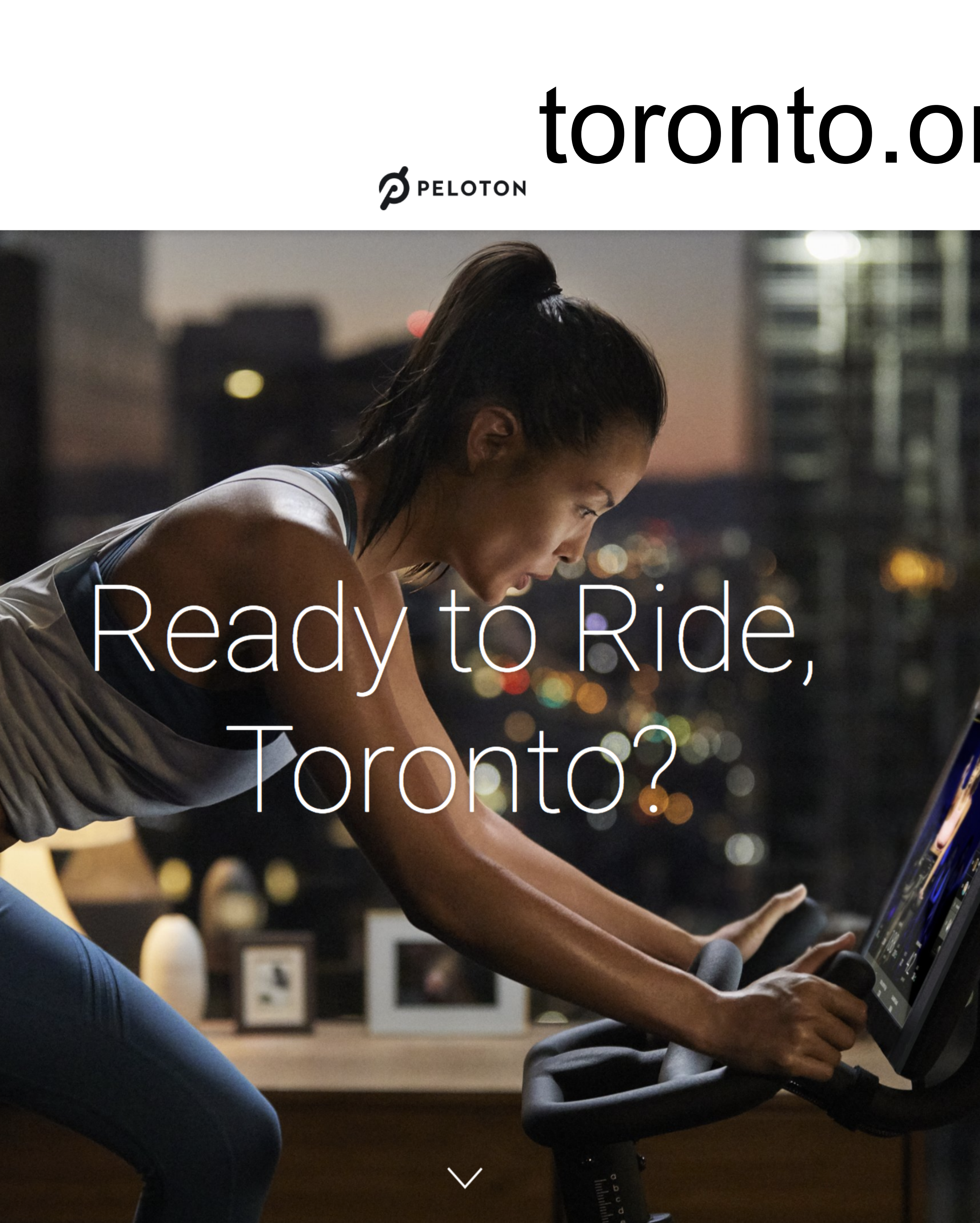


PyObject *PyLong_FromLong(long v)

Return value: New reference. Part of the Stable ABI.

Return a new PyLongObject object from v, or NULL on failure.

CPython implementation detail: CPython keeps an array of integer objects for all integers between -5 and 256. When you create an int in that range you actually just get back a reference to the existing object.



toronto.onepeloton.com

<	S	M	T	W	T
	28	29	30	31	1
SUNDAY, OCTOBER 28					
11:00	30 minutes			3 AVAILABLE >	
11:15	30 minutes			4 AVAILABLE >	
11:45	30 minutes			5 AVAILABLE >	

Source: What a Bug can Teach You about Python by Brad Dettmer

The Bug



paul 👍 7:08 PM

I booked two at 7am on 9/25. The appointments still return an empty array.

Buggy code in backend server

257

257

```
if appointment.time_slot_id is ts.id:
```

```
    time_slot_appointments.append(appointment)
```

```
a.time_slot.id is ts.id
```

```
a.time_slot_id == ts.id
```

Sources

- <https://hub.docker.com/r/j4ckofalltrades/legacy-python>
- <https://mail.python.org/pipermail/python-list/1999-June/001951.html>
- <https://docs.python.org/3/faq/design.html#why-can-t-raw-strings-r-strings-end-with-a-backslash>
- <https://docs.python.org/3/library/secrets.html>
- <https://lwn.net/Articles/590299/>
- <https://lwn.net/Articles/590331/>
- <https://sethmlarson.dev/deprecations-via-warnings-dont-work-for-python-libraries>
- <https://docs.python.org/3/c-api/long.html#:~:text=CPython,object>
- <https://docs.google.com/presentation/d/1mDeGnYIkYEvluAfEGTyTk-vUaYvALkdqfnuGIRqz0Hs>

Quiz



So You Think You Know Python Quiz

How well do you know Python's quirks?

All questions verified on Python 3.14.

Start Quiz

Thank You

