

WSGI and



python 3

Armin Ronacher

<http://lucumr.pocoo.org/> // [armin.ronacher@active-4.com](mailto:armin.ronacher@active-4.com) // <http://twitter.com/mitsuhiko>

# About Me

- using Python since version 2.2
- WSGI believer :)
- Part of the Poccoo Team: *Jinja*,  
*Werkzeug*, *Sphinx*, *Zine*, *Flask*

# “Why are you so pessimistic?!”

- Because I care
- Knowing what's broken makes fixing possible
- On the bright side: Python is doing really good

A close-up photograph of a pencil tip on lined paper. The pencil is positioned diagonally across the frame, with its tip pointing towards the center. The paper has blue horizontal lines and a red vertical margin line on the left. There are some faint, scribbled pencil marks on the paper, including a curved line above the text and some vertical lines to the left. The text 'Why Python 3?' is overlaid in the center in a bold, white, sans-serif font with a slight shadow.

**Why Python 3?**



**What is WSGI?**

# WSGI is PEP 333

Last Update: 2004

**Frameworks:** Django, pylons, web.py, TurboGears 2, Flask, ...

**Lower-Level:** WebOb, Paste, Werkzeug

**Servers:** mod\_wsgi, CherrPy, Paste, flup, ...

# WSGI is Gateway Interface

You're expecting too much

- WSGI was **not** designed with multiple components in mind
- Middlewares are often **abused**

# This ... is ... WSGI

Callable + dictionary + iterator

```
def application(environ, start_response):  
    headers = [('Content-Type', 'text/plain')]  
    start_response('200 OK', headers)  
    return ['Hello World!']
```

# Is *this* WSGI?

Generator instead of Function

```
def application(environ, start_response):  
    headers = [('Content-Type', 'text/plain')]  
    start_response('200 OK', headers)  
    yield 'Hello World!'
```

# WSGI is slightly flawed

This causes problems:

- input stream not delimited
- `read()` / `readline()` issue
- path info not url encoded
- generators in the function cause

# WSGI is a subset of HTTP

What's not in WSGI:

- Trailers
- Hop-by-Hop Headers
- Chunked Responses (?)

# WSGI in the Real World

`readline()` issue ignored

- Django, Werkzeug and Bottle are probably the only implementations not requiring `readline()` with a size hint.
- Servers usually implement `readline()` with a size hint.

# WSGI in the Real World



nobody uses write()



# CHANGE

*WSGI relevant*

**Language Changes**

# Things that changed

## Bytes and Unicode

- no more bytestring
- instead we have byte objects that behave like arrays with string methods
- old unicode is new str

# Only one string type ...

... means this code behaves different:

```
>>> 'foobar' == u'foobar'  
True
```

```
>>> b'foobar' == 'foobar'  
False
```

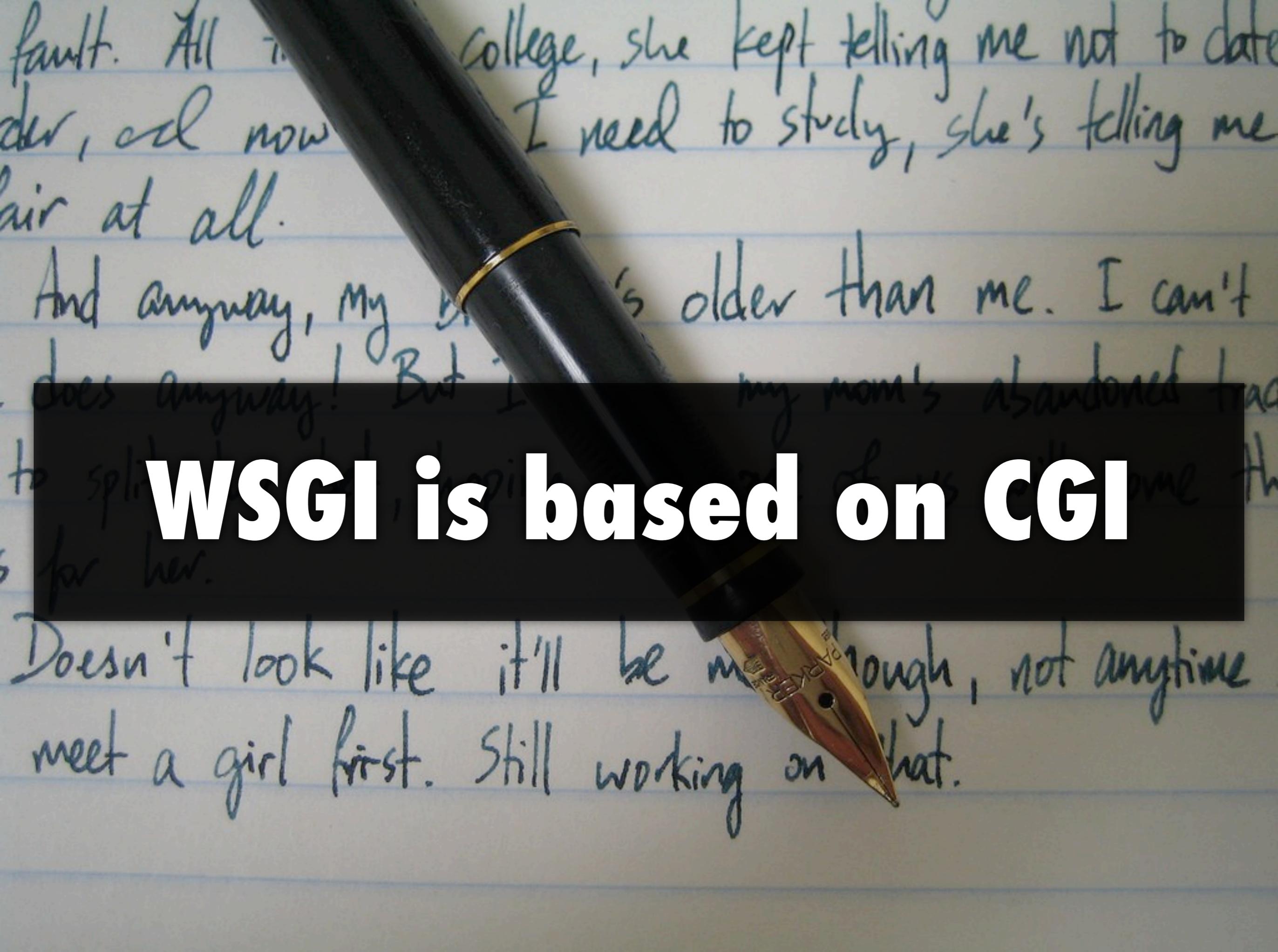
# Other changes

## New IO System

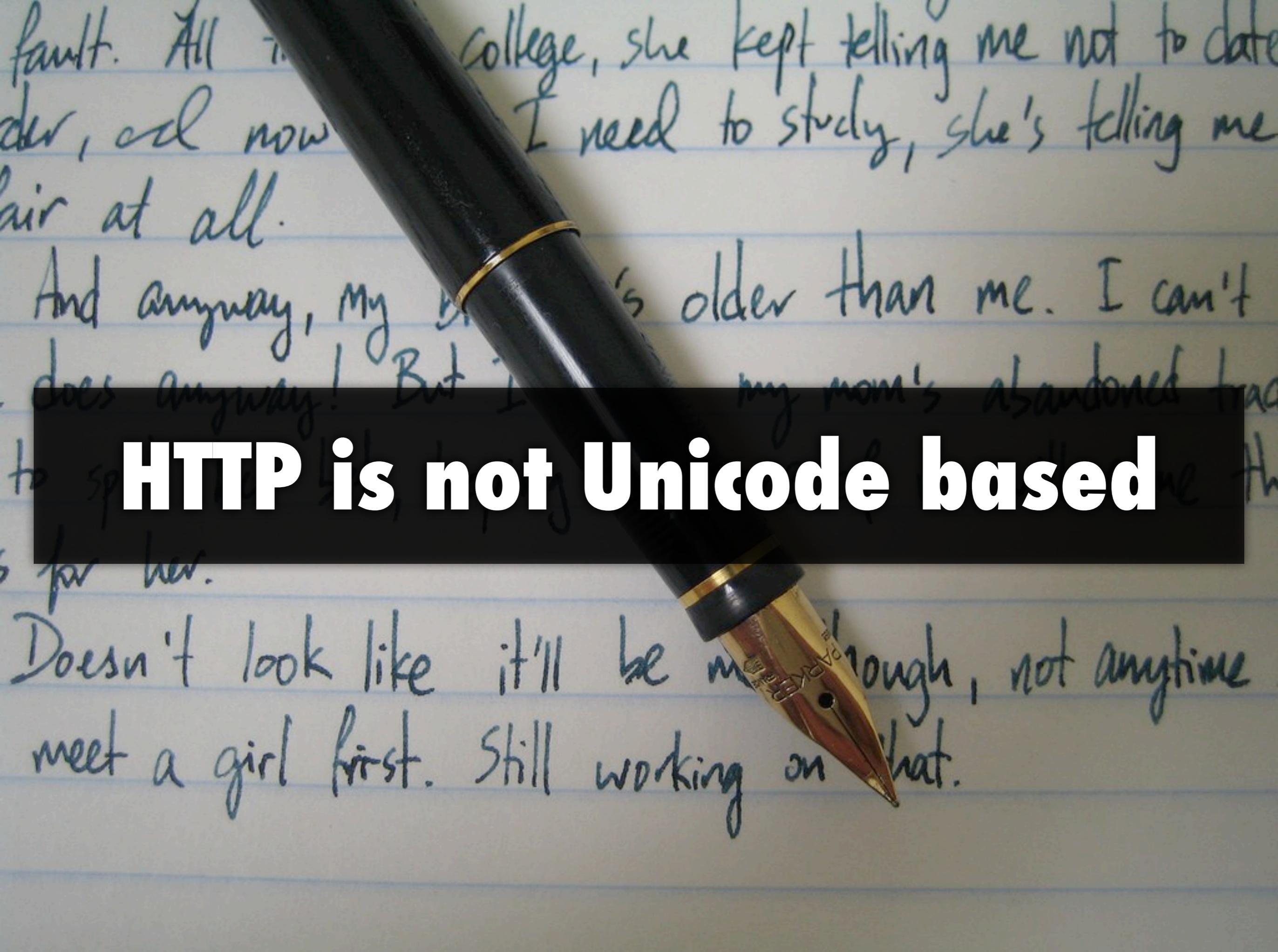
- `StringIO` is now a "str" IO
- `ByteIO` is in many cases what `StringIO` previously was
- take a guess: what's `sys.stdin`?

FACTS

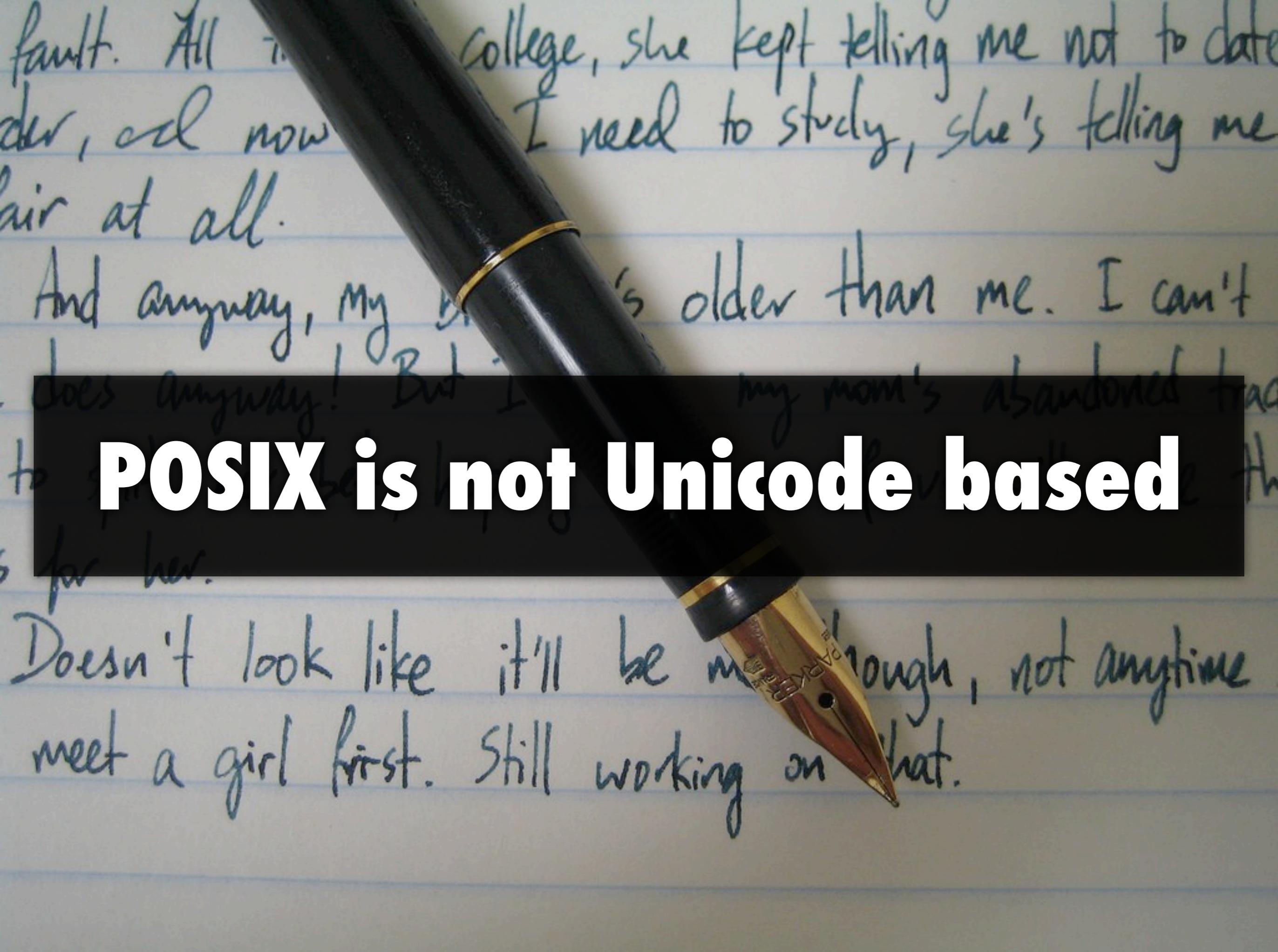
**FACTS!**



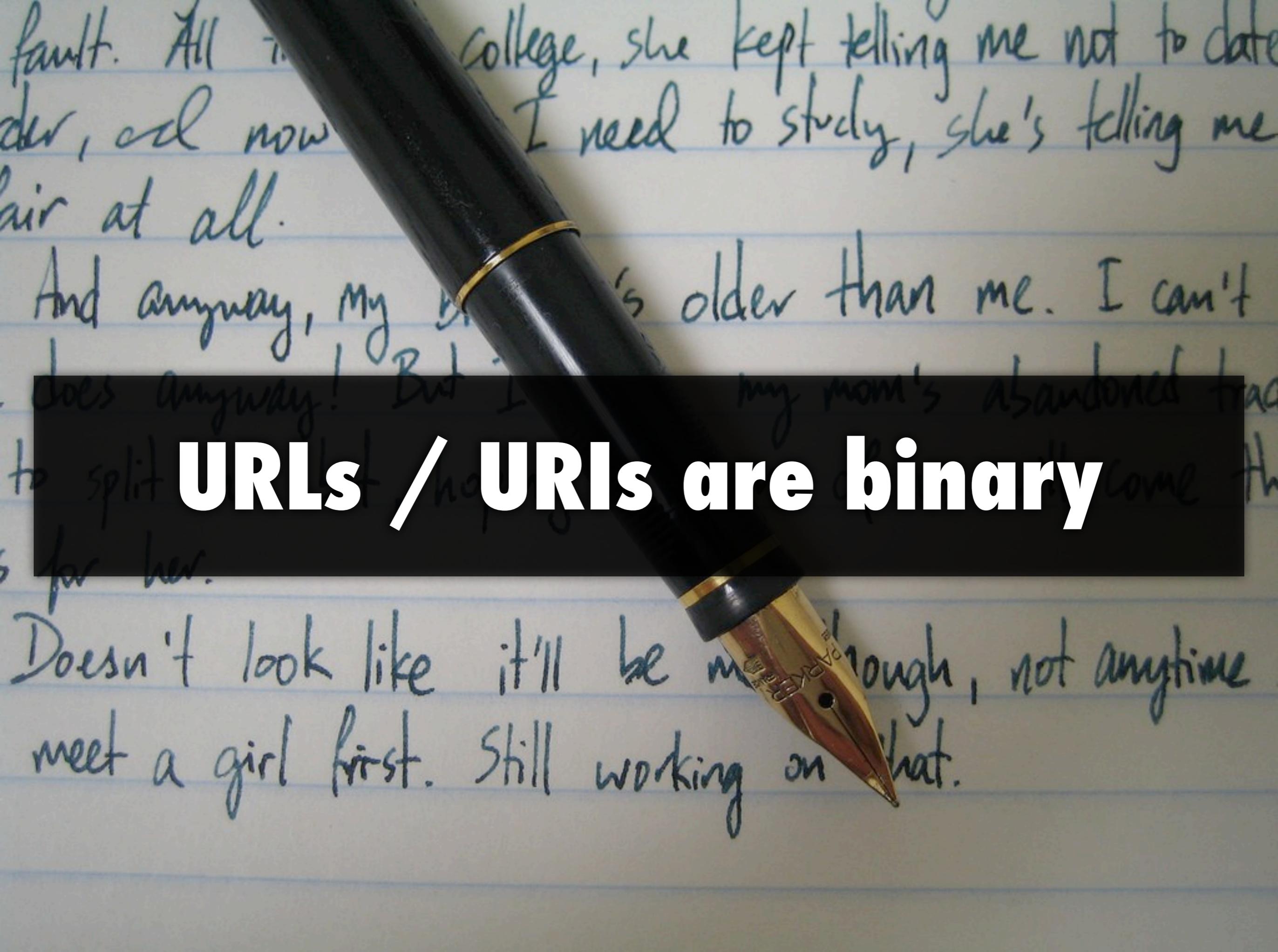
**WSGI is based on CGI**



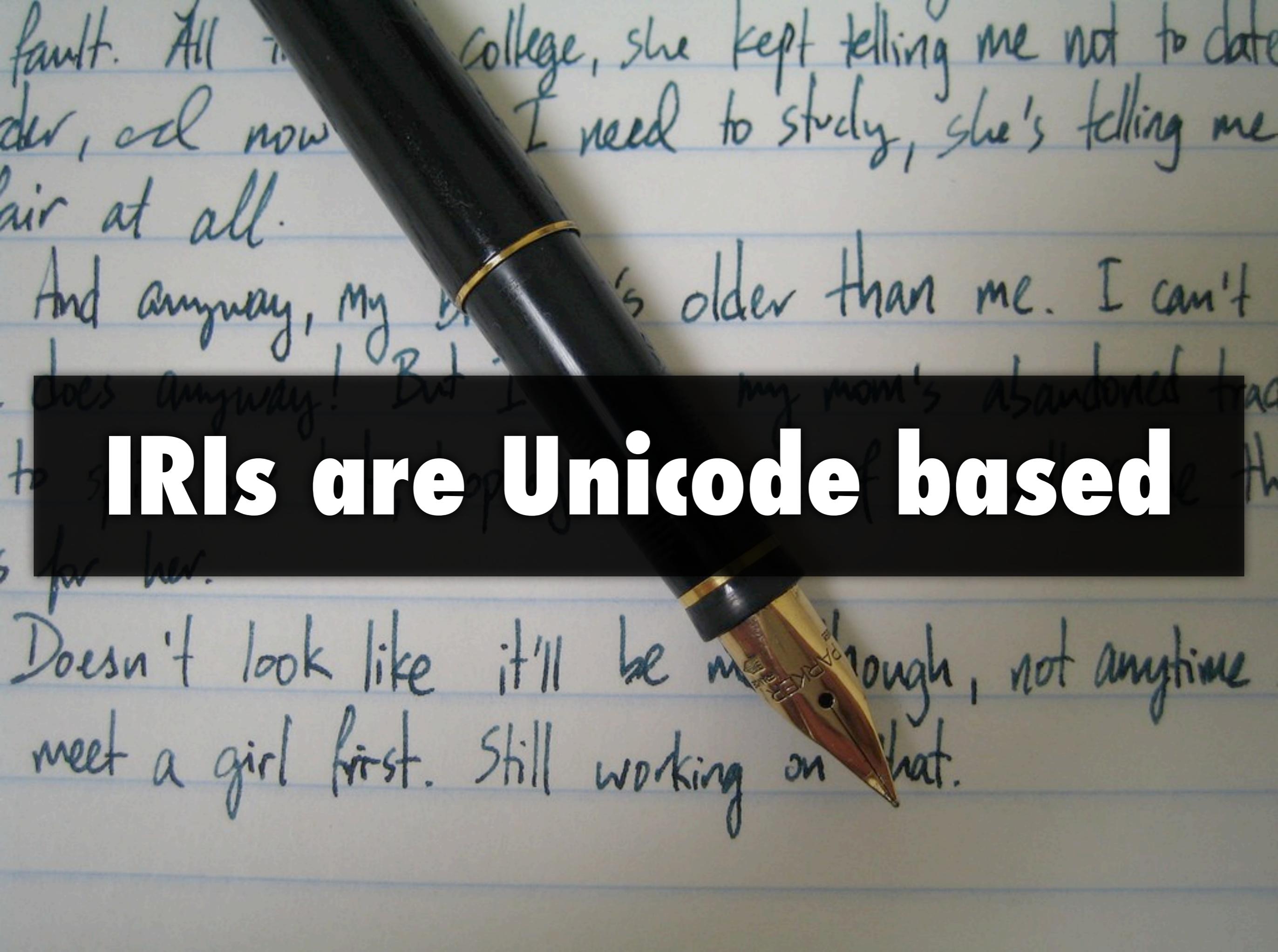
**HTTP is not Unicode based**



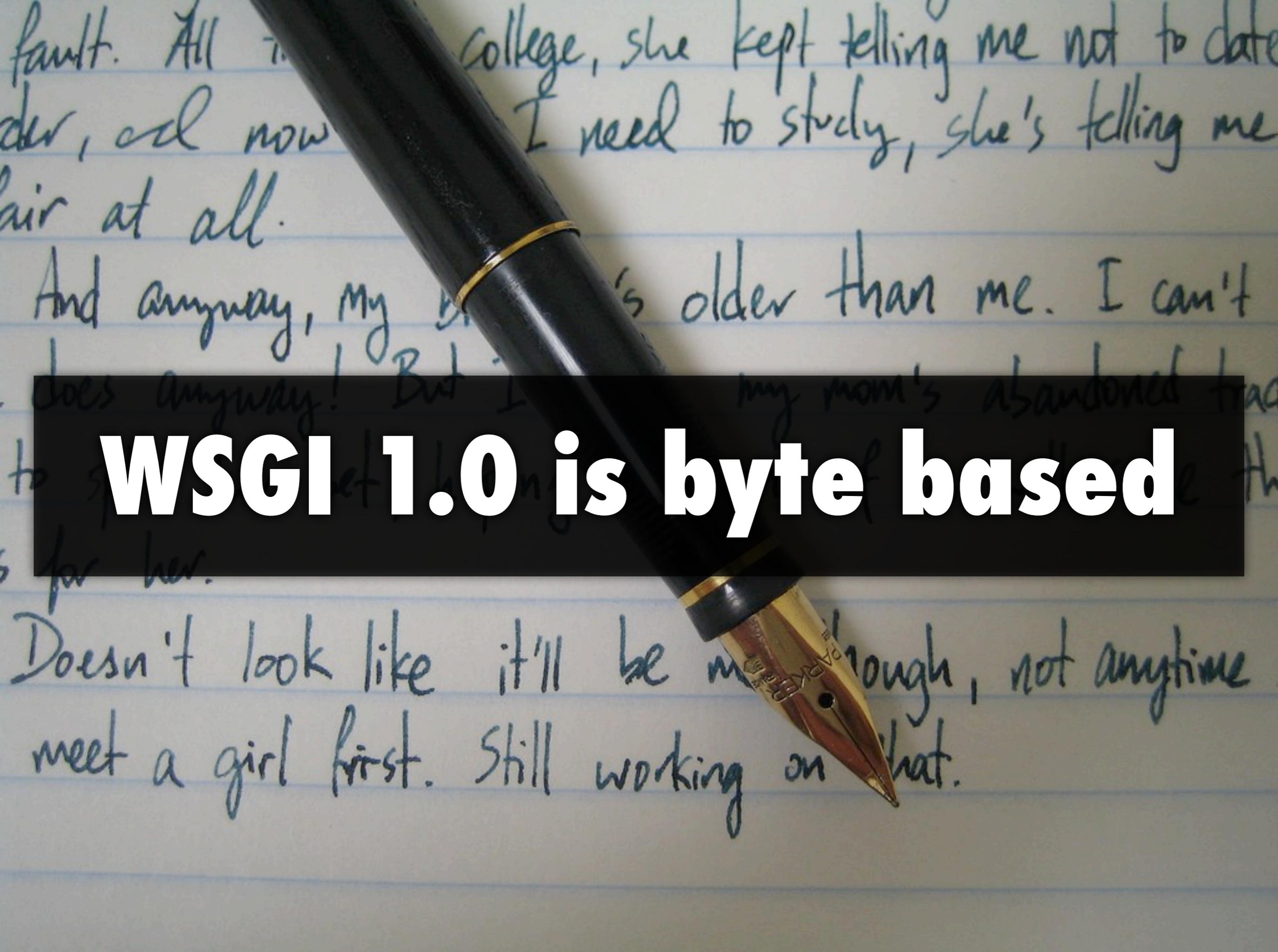
**POSIX is not Unicode based**



**URLs / URIs are binary**



**IRIs are Unicode based**



**WSGI 1.0 is byte based**



ALL PROBLEMS  
ARE OPPORTUNITIES  
IN DISGUISE

**Problems ahead**

# Unicode :(

IM IN UR STDLIB BREAKING UR CODE

- `urllib` is unicode
- `sys.stdin` is unicode
- `os.environ` is unicode
- **HTTP / WSGI are not unicode**

# What the stdlib does

regarding urlib:

- all URLs assumed to be UTF-8 encoded
- in practice: UTF-8 with some latinX fallback
- better would be separate URI/IRI handling

# What the stdlib does

the os module:

- Environment is unicode
- But not necessarily in the operating system
- Decode/Encode/Decode/Encode?

# What the stdlib does

the sys module:

- `sys.stdin` is opened in text mode, UTF-8 encoding is somewhat assumed
- same goes for `sys.stdout` / `sys.stderr`

# What the stdlib does

the cgi module:

- FieldStorage does not work with binary data currently on either CGI or any WSGI “standard interpretation”



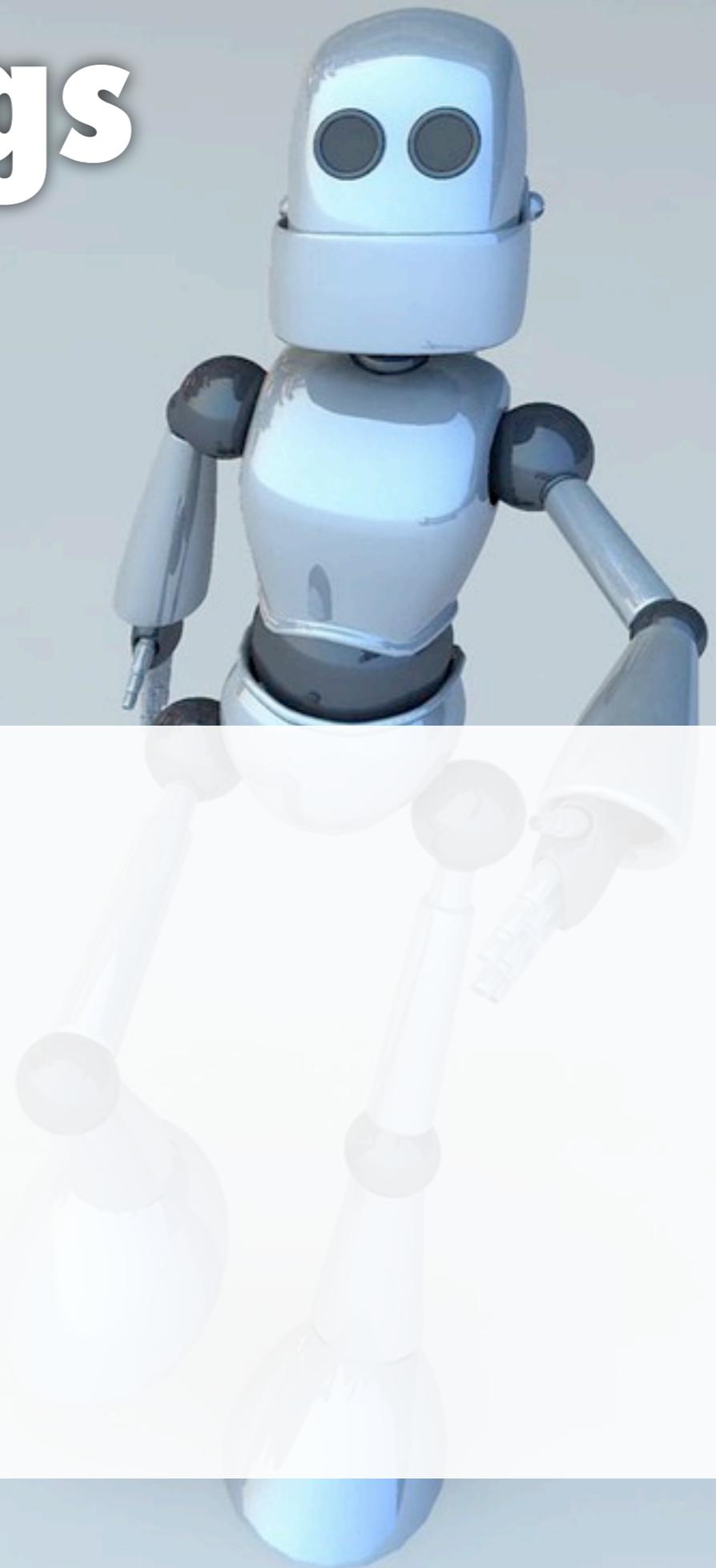
**Weird  
Specification /  
General  
Inconsistencies**

Muricy

# Non-ASCII things

in the environ:

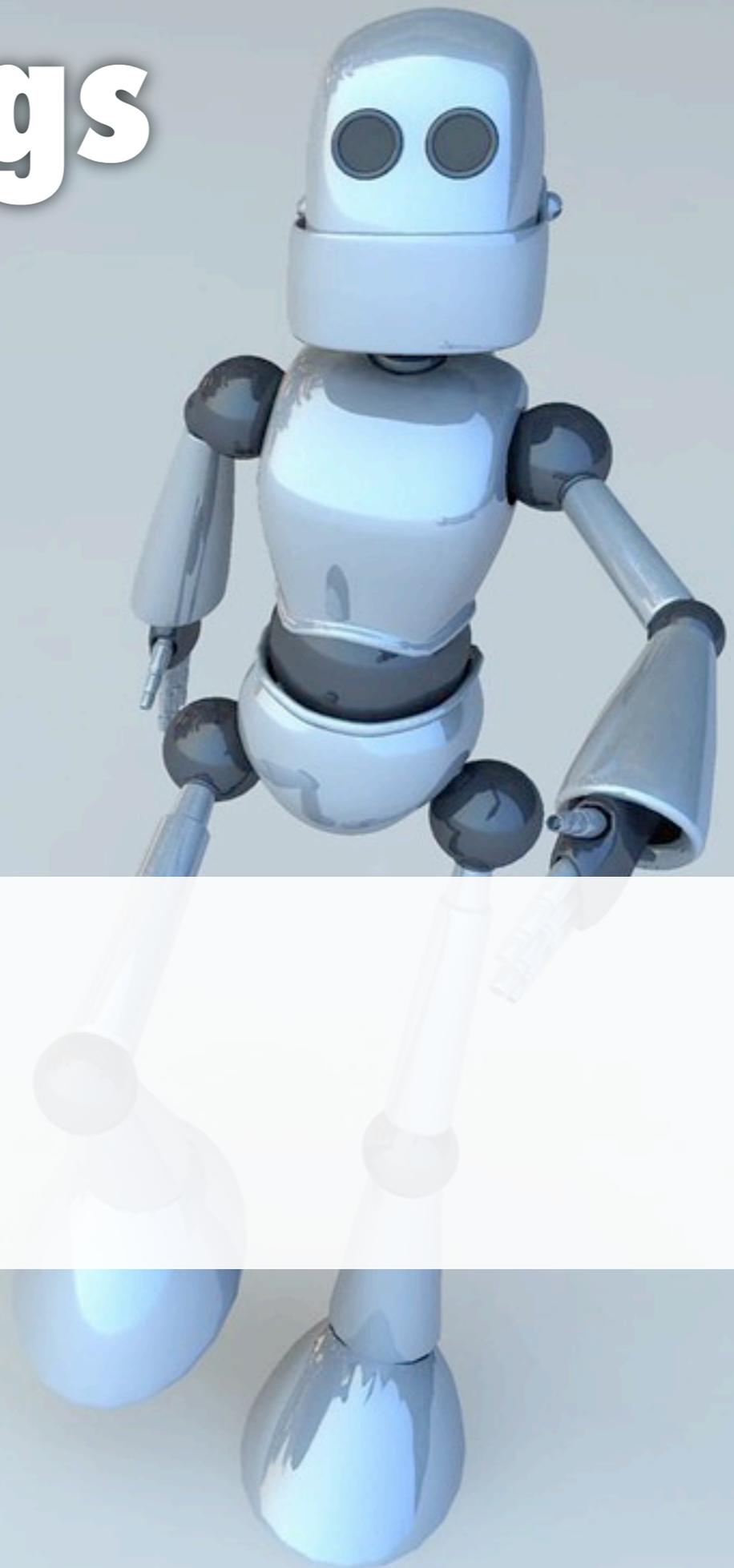
- **HTTP\_COOKIE**
- **SERVER\_SOFTWARE**
- **PATH\_INFO**
- **SCRIPT\_NAME**



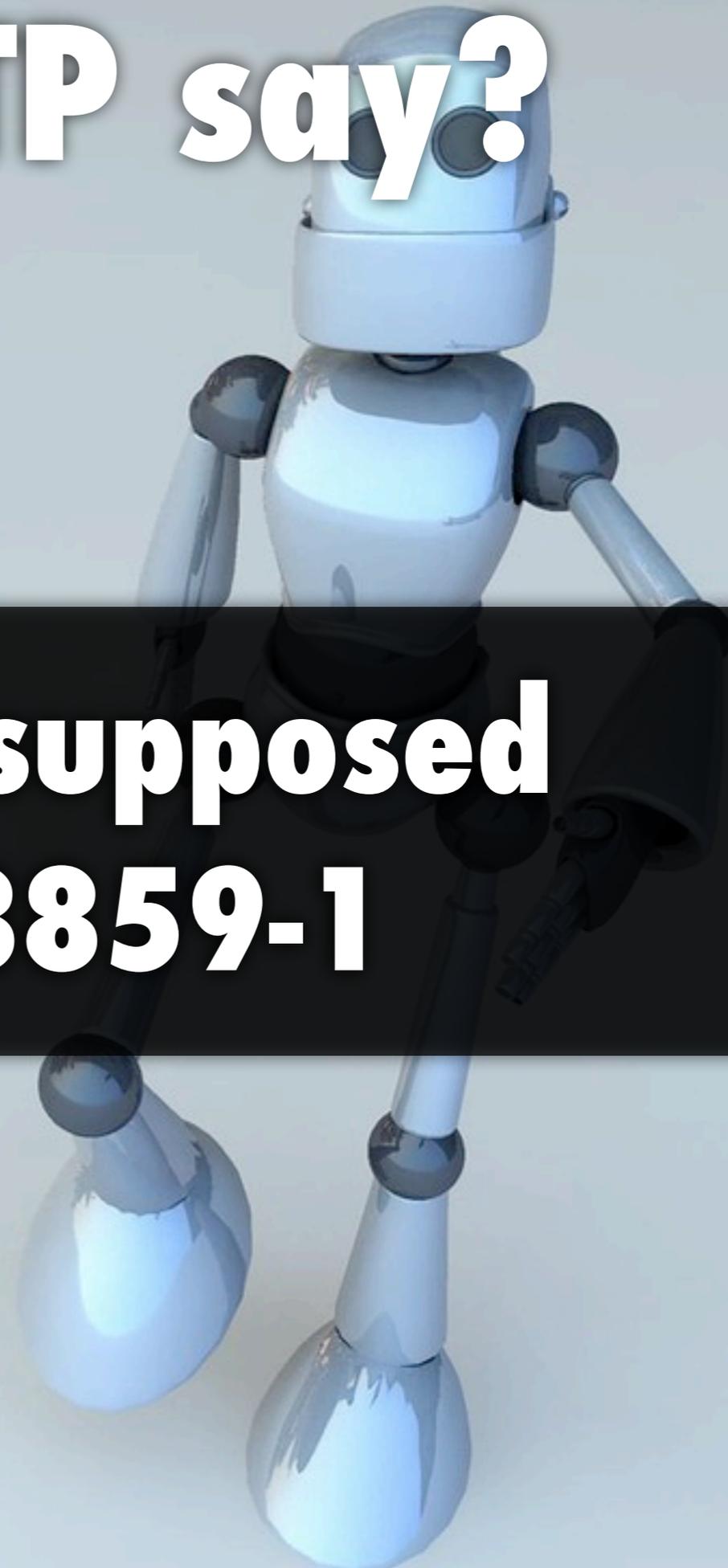
# Non-ASCII things

in the headers:

- **Set-Cookie**
- **Server**

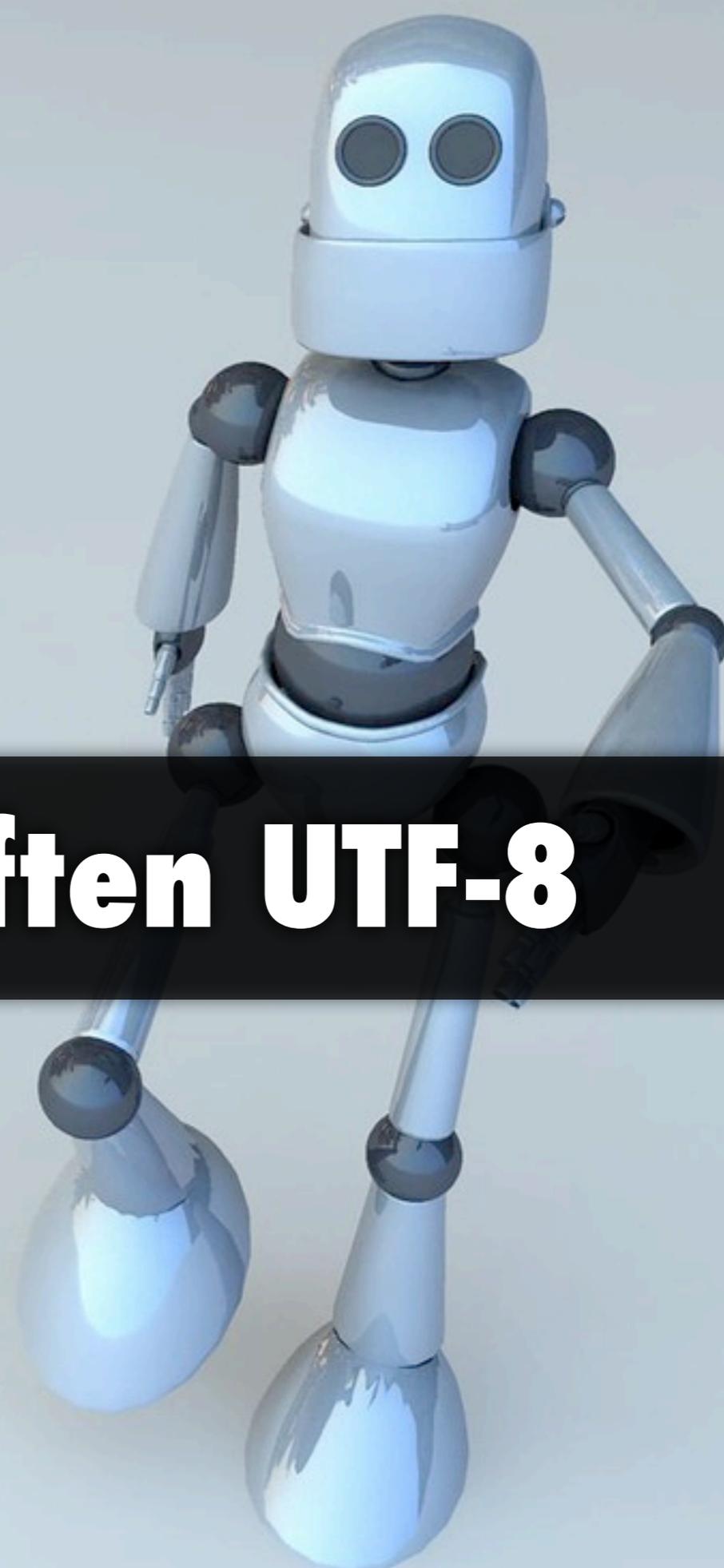


# What does HTTP say?



**headers are supposed  
to be ISO-8859-1**

# In practice?

A 3D rendered white robot with a friendly face, standing behind a black banner. The robot has a rounded head with two circular eyes, a simple body, and thin arms and legs. It is positioned on the right side of the frame, with its arms slightly out to the sides. The background is a plain, light blue-grey color.

**cookies are often UTF-8**

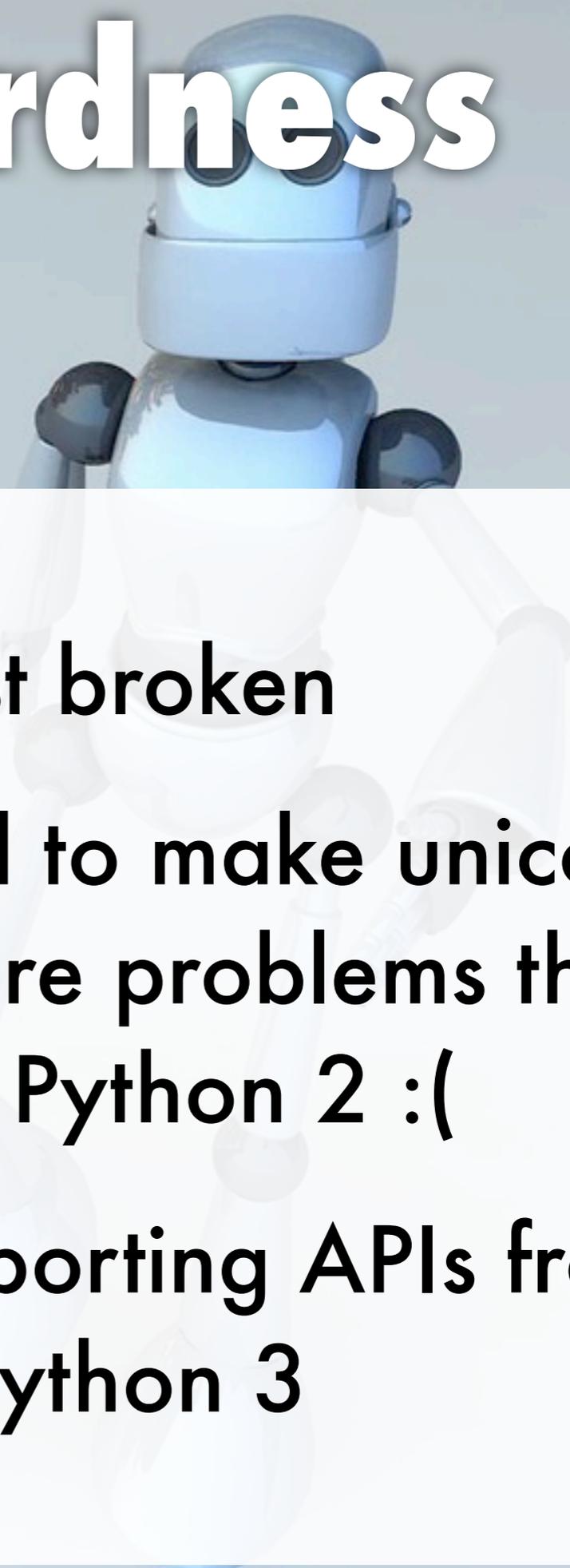
# Checklist of Weirdness



**the status:**

- 1. only one string type, no implicit conversion between bytes and unicode**
- 2. stdlib does not support bytes for most URL operations (!?)**
- 3. cgi module does not support any binary data at the moment**
- 4. CGI no longer directly WSGI compatible**

# Checklist of Weirdness



**the status:**

**5. wsgiref on Python 3 is just broken**

**6. Python 3 that is supposed to make unicode easier is causing a lot more problems than unicode environments on Python 2 :(**

**7. 2to3 breaks unicode supporting APIs from Python 2 on the way to Python 3**



**What would Graham do?**

# Two String Types

- native strings [*unicode* on 2.x, *str* on 3.x]
- bytestring [*str* on 2.x, *bytes* on 3.x]
- unicode [*unicode* on 2.x, *str* on 3.x]

# The Environ #1

- WSGI environ keys are native strings. Where native strings are unicode, the keys are decoded from ISO-8859-1.

# The Environ #2

- `wsgi.url_scheme` is a native string
- CGI variables in the WSGI environment are native strings. Where native strings are unicode ISO-8859-1 encoding for the origin values is assumed.

# The Input Stream

- `wsgi.input` yields bytestrings
- no further changes, the `readline()` behavior stays unchanged.

# Response Headers

- status strings and headers are bytestrings.
- On platform where native strings are unicode, native strings are supported but the server encodes them as ISO-8859-1

# Response Iterators

- The iterable returned by the application yields bytestrings.
- On platforms where native strings are unicode, unicode is allowed but the server must encode it as ISO-8859-1

# The write() function

- yes, still there
- accepts bytestrings except on platforms where unicode strings are native strings, there unicode strings are accepted and encoded as ISO-8859-1



**What does it mean for  
Frameworks?**



# URL Parsing [py2x]

this code:

```
rv = cgi.parse_qs1(qs)
for key, value in rv:
    d[key] = value.decode(charset)
```

# URL Parsing [py3x]

becomes this:

```
rv = urllib.parse.parse_qs1(qs)
for key, value in rv:
    d[key] = value
```

unless you don't want UTF-8, then  
have fun reimplementing

# Form Parsing

roll your own. `cgi.FieldStorage` was broken in 2.x regarding WSGI anyways. Steal from Werkzeug/Django

# Common Env [py2x]

this handy code:

```
path = environ['PATH_INFO'] \
    .decode('utf-8', 'replace')
```

# Common Env [py3x]

looks like this in 3.x:

```
path = environ['PATH_INFO'] \
    .encode('iso-8859-1') \
    .decode('utf-8', 'replace')
```

# Middlewares in [py2x]

this common pattern:

```
def middleware(app):
    def new_app(environ, start_response):
        is_html = []
        def new_start_response(status, headers,
                               exc_info=None):
            if any(k.lower() == 'content-type' and
                   v.split(';')[0].strip() == 'text/html'):
                is_html.append(True)
            return start_response(status, headers, exc_info)
        rv = app(environ, new_start_response)
        ...
    return new_app
```

# Middlewares in [py3x]

becomes this:

```
def to_bytes(x):
    return x.encode('iso-8859-1') if isinstance(x, str) else x

def middleware(app):
    def new_app(environ, start_response):
        is_html = []
        def new_start_response(status, headers,
                               exc_info=None):
            if any(to_bytes(k.lower()) == b'content-type' and
                   to_bytes(v).split(b';')[0].strip() == b'text/html'):
                is_html.append(True)
            return start_response(status, headers, exc_info)
        rv = app(environ, new_start_response)
        ...
    return new_app
```

# My Prediction

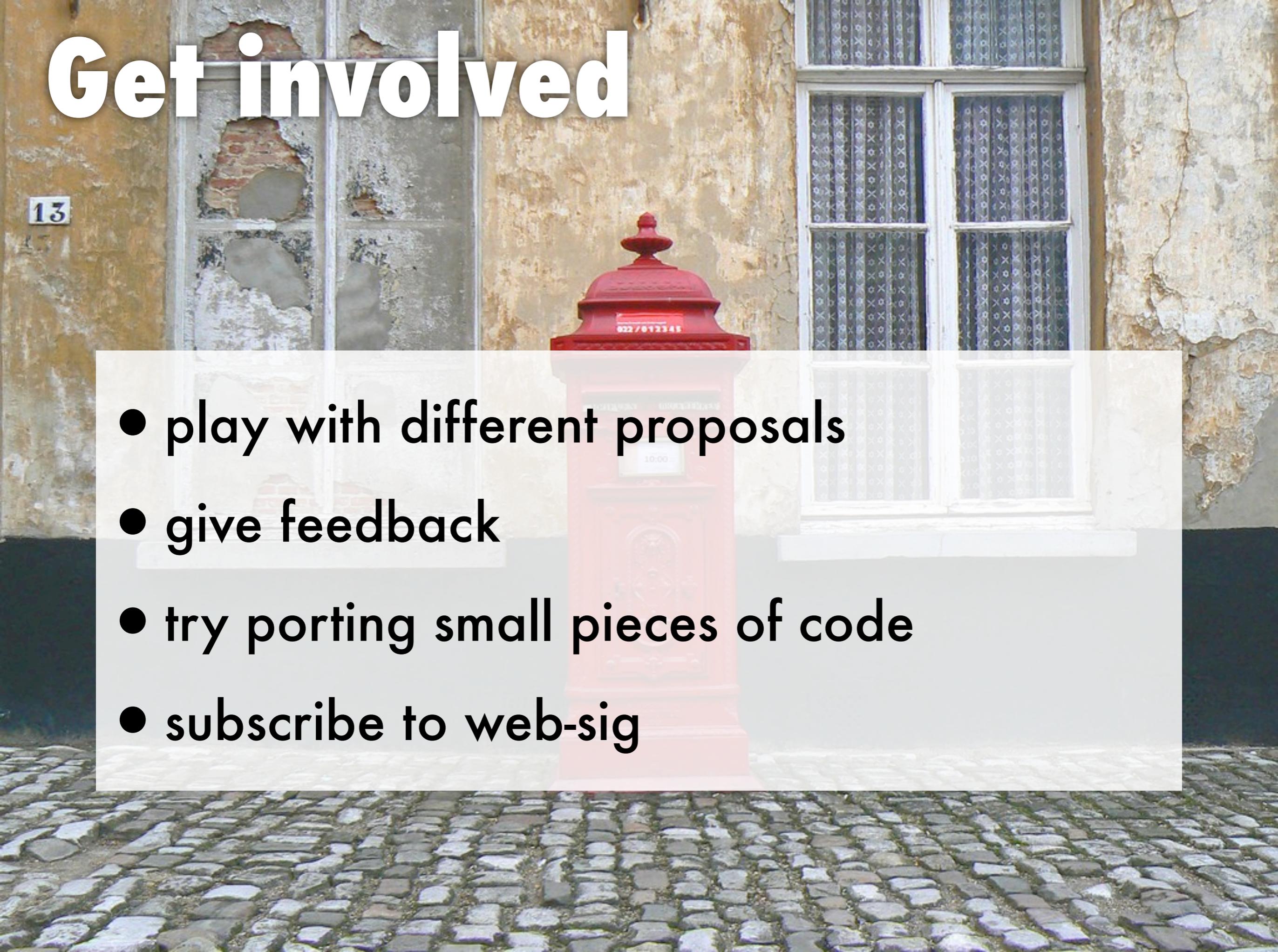
possible outcome:

- stdlib less involved in WSGI apps
- frameworks reimplement urllib/cgi
- internal IRIs, external URIs
- small WSGI frameworks will probably switch to WebOb / Werkzeug because of additional complexity

*My very own*  
**Pony Request**

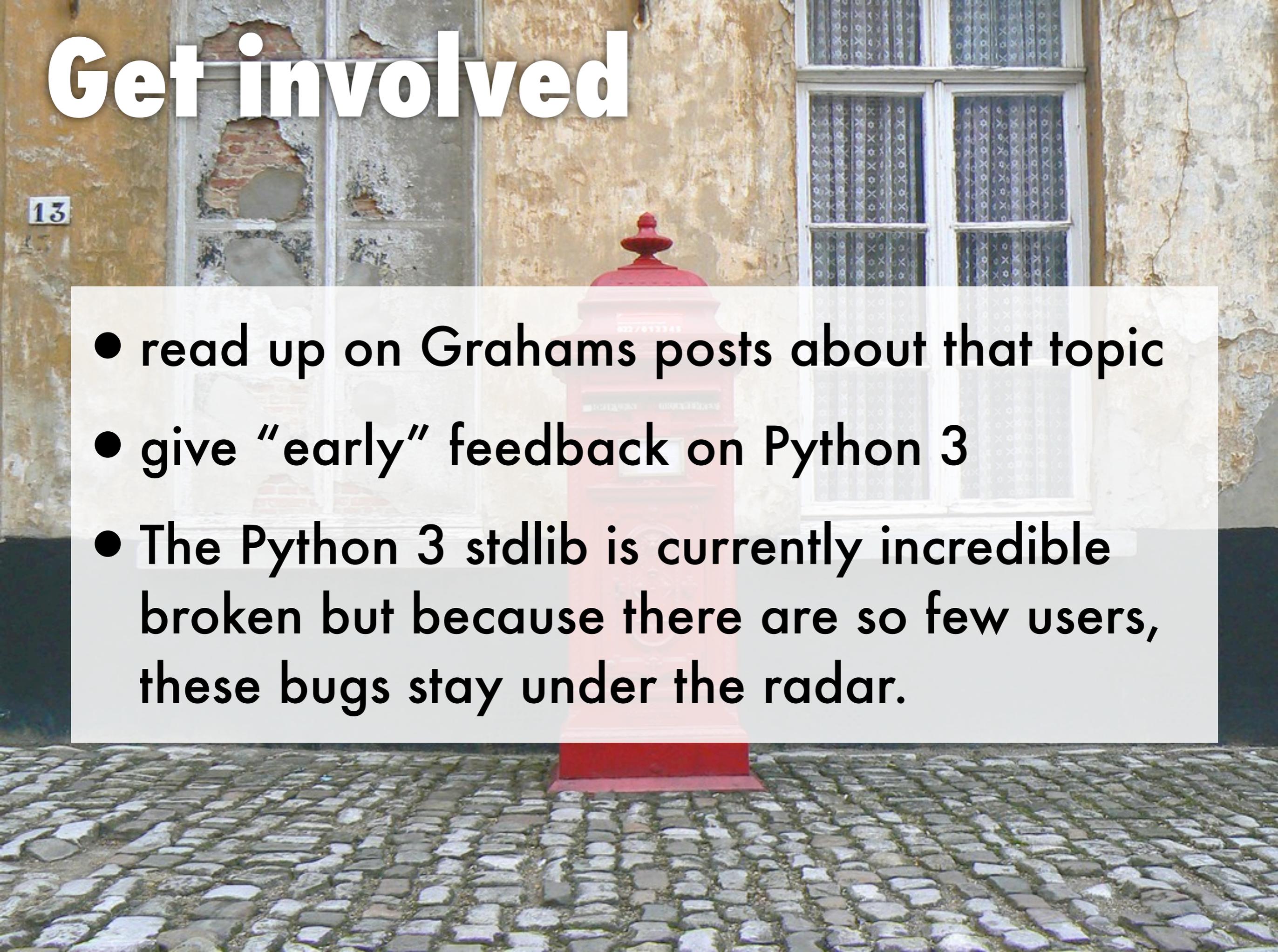


# Get involved

A red mailbox stands on a cobblestone street in front of a building with peeling plaster and a window with blue curtains. The mailbox has a small sign that reads "022 / 012345". The building has a small sign with the number "13" on the left side.

- play with different proposals
- give feedback
- try porting small pieces of code
- subscribe to web-sig

# Get involved

A red fire hydrant stands on a cobblestone street in front of a building with peeling plaster and a window with patterned curtains. The number 13 is visible on the wall to the left.

- read up on Grahams posts about that topic
- give “early” feedback on Python 3
- The Python 3 stdlib is currently incredible broken but because there are so few users, these bugs stay under the radar.

13

*Remember:*

**2.7 is the last 2.x release**





**Questions?**

# Legal

licensed under the creative commons attribution-noncommercial-share alike 3.0 austria license

© Copyright 2010 by Armin Ronacher

**images in this presentation used under compatible creative commons**

**licenses.** sources: <http://www.flickr.com/photos/42311564@N00/2355590508/> <http://www.flickr.com/photos/emagic/56206868/> <http://www.flickr.com/photos/special/1597251/> <http://www.flickr.com/photos/doblonaut/2786824097/> <http://www.flickr.com/photos/1sock/2728929042/> [http://www.flickr.com/photos/spursfan\\_ace/2328879637/](http://www.flickr.com/photos/spursfan_ace/2328879637/) <http://www.flickr.com/photos/svensson/40467662/> <http://www.flickr.com/photos/patrickgage/3738107746/> <http://www.flickr.com/photos/wongjunhao/2953814622/> <http://www.flickr.com/photos/donnagrayson/195244498/> <http://www.flickr.com/photos/chicagobart/3364948220/> <http://www.flickr.com/photos/churl/250235218/> <http://www.flickr.com/photos/hannner/3768314626/> <http://www.flickr.com/photos/flysi/183272970/> <http://www.flickr.com/photos/annagaycoan/3317932664/> <http://www.flickr.com/photos/ramblingon/4404769232/> [http://www.flickr.com/photos/nocallerid\\_man/3638360458/](http://www.flickr.com/photos/nocallerid_man/3638360458/) <http://www.flickr.com/photos/sifter/292158704/> <http://www.flickr.com/photos/szczur/27131540/> <http://www.flickr.com/photos/e3000/392994067/> <http://www.flickr.com/photos/87765855@N00/3105128025/> <http://www.flickr.com/photos/lemsipmatt/4291448020/>