WSGI and Python 3
About Me

- using Python since version 2.2
- WSGI believer :)
- Part of the Pocoo Team: Jinja, Werkzeug, Sphinx, Zine, Flask
Because I care

Knowing what’s broken makes fixing possible

On the bright side: Python is doing really good
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, ...
You’re expecting too much

- WSGI was not designed with multiple components in mind
- Middlewares are often abused
Callable + dictionary + iterator

```python
def application(environ, start_response):
    headers = [('Content-Type', 'text/plain')]
    start_response('200 OK', headers)
    return ['Hello World!']
```
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.
nobody uses write()
Language Changes

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:

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

>>> b'foobar' == 'foobar'
False
```
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!
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
Problems ahead

ALL PROBLEMS ARE OPPORTUNITIES IN DISGUISE
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 urllib:

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

- FieldStorage does not work with binary data currently on either CGI or any WSGI "standard interpretation"
Weird Specification / General Inconsistencies
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?

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
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]
WSGI environ keys are native strings. Where native strings are unicode, the keys are decoded from ISO-8859-1.
• 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.
• 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?
this code:

```python
rv = cgi.parse_qsl(qs)
for key, value in rv:
    d[key] = value.decode(charset)
```
URL Parsing \[py3x\]

becomes this:

```python
rv = urllib.parse.parse_qsl(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:

```python
path = environ['PATH_INFO'] \ 
  .decode('utf-8', 'replace')
```
Common Env [py3x]

looks like this in 3.x:

```python
path = environ['PATH_INFO'] \n    .encode('iso-8859-1')
    .decode('utf-8', 'replace')
```
Middlewares in \texttt{[py2x]}

this common pattern:

```python
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
```
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

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

• 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.
Remember:

2.7 is the last 2.x release
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/lemispinn/4291448020/