Django & Python 3

Aymeric Augustin - @aymericaugustin

PyConFR - September 16th, 2012

Python 3
is the future

Python 3 is the ~~future~~ present

Django wants a future

how?

porting strategies

- 3to2
- single source
- 2to3

Django's choices

- single source
- Python ≥ 2.6
- six
  - bundled as django.utils.six
- unicode_literals

The plan

Django 1.5 will provide experimental support for Python 3.

Compatibility features will be backported to Django 1.4.2.

Django 1.6 will be usable in production with Python 3.

six

- PY3
- constants
- object model
- syntax
- binary and text data
- renamed modules and attributes

a refresher on strings

strings

- Python 2 : `str` and `unicode`
- Python 3 : `bytes` and `str`

byte strings

- str in Python 2 : `'cafe'`
- bytes in Python 3 : `b'cafe'`
- `six.binary_type` or `bytes` (2.6)
- can be decoded
  - can raise `UnicodeDecodeError`
- ‘bytes’ in Django’s docs

unicode strings

- `unicode` in Python 2 : `u'café'`
- `str` in Python 3 : `'café'`
- `six.text_type`
- can be encoded
  - can raise `UnicodeEncodeError`
- ‘text’ in Django’s docs

native strings

- `str` in Python 2
- `str` in Python 3

- used by the WSGI specification and parts of the standard library
- useful idiom: `str('spam')`

Python 2
implicit conversions

```
>>> 'foo' + u'bar'
u'foobar'
```

Python 2
implicit conversions

```
>>> from StringIO import StringIO
>>> s = StringIO()
>>> s.write('foo')
>>> s.getvalue()
'foo'
>>> s.write(u'bar')
>>> s.getvalue()
u'foobar'
```

Python 2
implicit conversions

```
>>> from cStringIO import StringIO
>>> s = StringIO()
>>> s.write('foo')
>>> s.getvalue()
'foo'
>>> s.write(u'bar')
>>> s.getvalue()
'foobar'
```

Python 2
default encoding

```
>>> import sys
>>> sys.getdefaultencoding()
'ascii'
```

Python 2
default encoding

```
# Objects/unicodeobject.c
static char
unicode_default_encoding[100];

const char
*PyUnicode_GetDefaultEncoding(void)
{ return unicode_default_encoding; }

void _PyUnicode_Init(void)
{ strcpy(unicode_default_encoding,
         "ascii"); }
```

Python 2
implicit conversions

```
>>> u'café'.decode()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't
encode character u'\xe9' in position 3:
ordinal not in range(128)
```

Python 2
implicit conversions

```
>>> 'café'.encode()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't
decode byte 0xc3 in position 3: ordinal
not in range(128)
```

Python 3
no implicit conversions

```
>>> 'foo' + b'bar'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'bytes' object
to str implicitly
```

Python 3
default encoding

```
>>> import sys
>>> sys.getdefaultencoding()
'utf-8'
```

Python 3
default encoding

```
# Objects/unicodeobject.c
const char *
PyUnicode_GetDefaultEncoding(void)
{
    return "utf-8";
}
```

Python 3
no implicit conversions

```
>>> 'café'.decode()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no
attribute 'decode'
```

Python 3
no implicit conversions

```
>>> 'café'.encode()
b'caf\xc3\xa9'
```

```
>>> _.encode()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'bytes' object has no
attribute 'encode'
```

tl:dr; Python 3 has sane string handling

porting Django *or a pluggable app* in 12 steps

0) have a test suite

If you don't,
please stay on Python 2
while the rest of the world
moves on.

You won't be missed.

1) unicode literals

```
from __future__ import unicode_literals
```

- enable it in every module
- avoid bytestrings
  - automation doesn't really help :(

- *or, if you love `u` prefixes, PEP 414 brings the past to the future (3.3)*

2) print function

```
from __future__ import print_function
```

```
print("One day, my fingers will type "
      "this bracket without me even "
      "thinking about it.")
```

3) exceptions

```
def do_stuff():
    raise Exception("Nope!")

def oh_well(exc):
    print(exc.args[0])

try:
    do_stuff()
except Exception as exc:
    oh_well(exc)
```

4) metaclasses

```
from six import with_metaclass

class IHateMyCoworkers(
        with_metaclass(IMetaHateThem)):
    """Go figure."""
```

```
def with_metaclass(meta, base=object):
    return meta("NewBase", (base,), {})
```

5) other syntax fixes

- longs: `1L => 1`
  - `sys.maxint => ---`
- octal literals: `022 => 0o022`
- non-ASCII bytes literals
- `` `obj` => repr(obj) ``
- etc.

6) imports

```
from functools import reduce
from six.moves import xrange

try:
    from urllib.parse import urlparse
except ImportError:     # Python 2
    from urlparse import urlparse
```

7) string types

```
(str, unicode)    =>  six.string_types
(basestr,)        =>  six.string_types

unicode           =>  six.text_type
                  str(b'foo') == "b'foo'" (!)
str               =>  bytes/str           (!)

smart_unicode     =>  smart_text
force_unicode     =>  force_text
smart_str         =>  smart_bytes/str (!)
```

8) representation

- if you use Django
  - define only `__str__`
  - `@python_2_unicode_compatible`
  - `django-2to3.py` helper

- `__repr__` must return a `str`

9) object model

```
next            =>  __next__

 class Iterator(object):
     def next(self):
         return type(self).__next__(self)

__nonzero__   =>  __bool__
__div__       =>  __idiv__ / __truediv_

callable() removed
no default __hash__ implementation
```

10) dict methods

```
for k in d: ...
for k, v in six.iteritems(d): ...
for v in six.itervalues(d): ...
```

```
keys = list(d)
items = list(six.iteritems(d))
values = list(six.itervalues(d))

items = list(d.items())
values = list(d.values())
```

11) iterators

- `map`, `range`, `zip` return iterators
  - wrap in `list()` if you need a list or plan to iterate more than once
  - replace with a list comprehension
- strings are iterable
  - avoid `hasattr(x, '__iter__')`

Tips

- write beautiful Python 3 code that also works on Python 2
- design your APIs carefully before adding `.encode()` and `.decode()`
- `python3 -m py_compile **/*.py`
- experiment with 2to3 fixers and pick from 2to3's output

hunt regressions on Python 2

happy porting

Questions

Further reading

https://docs.djangoproject.com/en/dev/topics/python3/
http://docs.python.org/py3k/howto/pyporting
http://packages.python.org/six/

https://www.djangoproject.com/weblog/2012/mar/13/py3k/
https://www.djangoproject.com/weblog/2012/aug/19/experimental-python-3-support/
http://nedbatchelder.com/text/unipain.html