How to make your code Python 2/3 compatible

Dr. Brett Cannon brett@python.org 2015-04-10 @ PyCon

This talk is **NOT** about convincing you to use Python 3

See my PyCon 2013 talk if you need convincing

You can start **TODAY!!!**

If you only get one thing out of this talk, let it be this

References

- <u>http://python3porting.com</u>
- "What's New" documents for each Python release
- Porting HOWTO: <u>docs.python.</u> <u>org/3/howto/pyporting.html</u>

Learn to love six

- Compatibility library to smooth out edges
- Supports Python 2.5 Python 3
- Single module for easy vendoring
- <u>https://pypi.python.org/pypi/six</u>

Only support Python 2.7

RHEL users can get Python 2.7 through Red Hat Collections

Good test coverage is critical

- So you don't accidentally break anything when porting
- coverage.py is handy
 - <u>https://pypi.python.org/pypi/coverage</u>

(Basic) new file template

coding: utf-8

Transpilers do all the easy stuff

Other tools help you to not undo your hard work

Modernize

- Harnesses 2to3 to update Python 2 code to work with Python 2.6 3 as much as possible
- <u>https://pypi.python.org/pypi/modernize</u>

Futurize

- Think Modernize but with more of a Python 3 feel
- Provides backports of things from Python 3 such as the bytes type
- Part of future project: <u>https://pypi.python.</u> <u>org/pypi/future</u>

Some fixes require thinking

Sorry.

Need to care about text vs. binary data

Can't conflate the two anymore

Need to make API decisions about text vs. binary data

unicode/str in Python 2, str/bytes in Python 3

Mark all your string literals

- I recommend b prefix + unicode_literals future statement when possible
- u and b prefixes also work
- In the end you should know exactly what type of data a string literal represents

 Tooling will help enforce this

Updating your APIs

- If it's to work with text ...
 - Make it work with Unicode
- If it's to work with binary data ...
 - Watch out for indexing on bytes
- Be strict with whether you pass in text or binary data, not just str in Python 2
- Let six help you

Text/bytes method uniqueness

str



- mod
- encode
- format
- isdecimal
- isnumeric

• decode

Python 3.5 improvements

- Bytes interpolation
 - o b'I %s bytes' % (b'love',)
- -b will warn when comparing bytes to int
 - Helps with the bytes-indexing issue

Division

This shouldn't be a surprise; been coming since Python 2.2

What to watch out for

- 5 / 2
 - 2 in Python 2
 - 2.5 in Python 3
- Python 3 semantics in Python 2
 - from __future__ import division
 -Q flag to interpreter
- Not automatic in case you're using something other than built-in types



- Can warn against some things not allowed or changed in Python 3
- use the --py3k flag to run **only** checks related to Python 3 compatibility

Python flags

- -3
 - Triggers various warnings for things not available in Python 3
 - Can use –W to control how severe to make the warnings
- -b
 - To help with common bytes-related issues
 - Is a no-op in Python 2, so can blindly use

Your code now works in Python 3!

Don't forget python2 -3, python3 -b, and Pylint in your testing/CI

Dealing with those pesky dependencies

Relying on others can be so trying sometimes

caniusepython3

- Checks your (in)direct dependencies to see who is blocking your move to Python 3
- API for test integration
- Has extra checkers to work with Pylint
- https://caniusepython3.com/
- https://pypi.python.org/pypi/caniusepython3

Getting dependencies ported

- Ask
- Do it yourself
- Hire someone to do it for you

Use cffi, Cython, or ctypes for extensions

There is also an official HOWTO on porting hand-written extension code

Now you can use Python 3!

Welcome to the latest version of Python

python3 -bb

Warns about common mistakes from mixing str and bytes

Continuous integration

- Use pylint --py3k to prevent regressions
- Use Tox to run tests under various Python versions
 - o <u>https://pypi.python.org/pypi/tox</u>



Bonus slides

from my Thumbtack talk; search for [thumbtack brett cannon] for YouTube video

Change is good for you!

Stuff in Python 2.7 that's different in Python 3.4

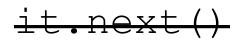
Fewer built-ins

- apply()
- buffer()*
- coerce()
- cmp()
- execfile()
- file()
- raw_input()*
- xrange()*
- StandardError

More iterators

- filter()
- map()
- zip()
- dict.items() et.al.

Advancing iterators



next(it)

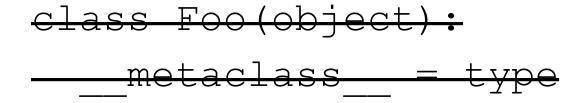
Less syntax, more functions

exec 'print `"Hello!"`' exec('print(repr("Hello!"))')

New-style classes everywhere

class Foo(object): pass
class Foo(): pass
class Foo: pass

Declaring metaclasses



class Foo(object, metaclass=type):
 pass

Parameter unpacking is gone

def func(a, (b, c), d): pass

Catching exceptions

except Exception, exc: ... except Exception as exc: ...

Raising exceptions

raise Exception, 'uh-oh' raise Exception('uh-oh')

Imports

from __future__ import absolute_import
from ..spam import eggs

Octal and binary literals

0720 00720 0b10101

Integer unification

- int went away
- long became int
- *L* suffix is no more

Standard library renamings

- Fixed some bad names
 - o ConfigParser -> configparser
- Turned some things into packages
 - o httplib -> http.client
 - BaseHTTPServer et. al. -> http.server

All of that works in Python 2.6!

And you can have it in an automated fashion!

Decorate/sort/undecorate

Integer division

- int / int returns a float
- int // int does what Python 2 does
- Get the semantics in Python 2
 - o from __future__ import division
 - o -Q new
 - Been around since Python 2.2

Text and binary data

• Python 2

- Text is basestring: (str, unicode), essentially
- Binary data is str (bytes is an alias in Python 2.6)
- Python 3
 - Text is str (similar to unicode in Python 2)
 - Binary data is bytes (sort of similar to str in Python 2)
 - To see differences, try set(dir(str)). difference(dir(bytes))

All of that is still available in Python 2.6!

It just takes some effort to have

New features!

In Python 3.4 that you can't have in Python 2.6

Set literals

$x = \{1, 2, 3, 4\}$

Set & dict comprehensions

{x**x for x in range(10) } {x: x**x for x in range(10) }

All of that is in Python 2.7!

Everything from now on is exclusive to Python 3, I promise

Unicode everywhere

- Source code is UTF-8 encoded by default
- Based on the Unicode standard annex UAX-31 with some tweaks

pycache

- All bytecode files are tagged per interpreter to prevent overwriting when using a different Python version

Extended iterable unpacking

- a, *b, c = range(10)
- a == 0
- b == list(range(1, 9))
- c == 9

Enhanced exceptions

- Chaining connects causal chain of exceptions
 - Implicit from simply raising another exception while another is active
 - Explicit with raise exc2 from exc1
- Traceback now embedded in exception

Keyword-only arguments

def func(a,*, are_you_sure): pass

Function annotations

def func(a:int) -> float: pass

nonlocal

def outer(): x = 0def inner(): nonlocal x x += 1 return x, inner



class Foo(bar): def __init__(self): super().__init__()

Stable ABI

- Hides interpreter details
- Guaranteed not to change
- Define Py_LIMITED_API and your extension module won't require recompilation per Python version



for x in range(10): yield x yield from range(10)

Significant stdlib additions

- ssl.SSLContext
- asyncio
- tracemalloc

pip & venv

- pip is now installed by default
- Virtual environments created by venv install pip by default
- Plans to have platform installers install pip in a future Python 2.7 release

Performance

- decimal implemented in C
- Integer math faster
- More efficient string memory use
- Key-sharing dictionaries
- Custom memory allocators
- Interchangeable hash algorithm

Looking to the future

Preview of Python 3.5

Matrix multiplication

% formatting for bytes

- Supported subset of what % does for strings
- Makes constructing ASCII-based binary data easier
- Will help binary-manipulating Python 2 code also work in Python 3