Writing Code to run on Python 2.x and 3.x

A Decision

- Do you want to run on 2.5? 2.6? 2.7?
- Do you need 3.0? 3.1? 3.2? 3.3? 3.4? 3.5?

- Pypy 4.0.1 is pretty much 2.7
- Pypy3 2.4.0 is pretty much 3.2
- Jython 2.7.0 is pretty much 2.7

What's the difference?

- 2.5 is pretty ignorant of 3.x
- 2.6 and 2.7 were created to ease the transition to 3.x
- 3.x has some of 2.x' oddities cleaned up

- CPython is Python in C, AKA “Python”
- Pypy is Python in Python with a JIT: very fast
- Jython is Python in Java: interop with java libraries

Approaches

- 2to3
- 3to2
- Run on both - we'll mostly talk about this

2to3 vs 3to2

- 2to3: converting code automatically from 2.x to 3.x
- 3to2: converting code automatically from 3.x to 2.x
- 3to2 is the more complete conversion because of strings

These are the main things we'll discuss about running on both

- Automated tests
- Tox
- Six
- Print: statement vs function
- Strings: bytes and unicode
- Try/Except
- String formatting
- Imports
- Division
- Range

Automated tests

- Always useful
- Especially useful when making changes like this

Tox

- Test code under multiple interpreters
- this-interpreter

Six (and python2x3)

- Looks very useful
- I've not used it, but I started a similar project: python23
- Byte strings
- I recommend you use Six instead of python2x3

Print

- 2.x: print 'abc'
- 3.x: print('abc')
- With a single argument, the difference vanishes
- 2.x: print('ab%s' % 'c')
- 3.x: print('ab%s' % 'c')
- 2.6 and up: from __future__ import print_function

Strings: Bytes

- In 2.5, byte strings are just str
- In 2.6 and 2.7, the b”prefix” is bytes, but they're really just str
- In 3.0 - 3.5, the b”prefix” is bytes, and they act like an array of small integers

Strings: Unicode

- In 2.4-2.7 (and likely earlier) the u'prefix' is for unicode
- In 3.0-3.2, the u'prefix' gives an error and unprefixed strings are unicode
- In 3.3, 3.4 and 3.5, the u'prefix' works again, and is just like an unprefixed, unicode string. Also unprefixed strings are unicode

Bytes, strings and I/O

- 2.x: os.read(os.open(), len) and open().read() both return str
- 3.x:
  - os.read(os.open(), len) returns bytes
  - open().read() returns unicode

- bufsock

Bytes and Unicode and your decision

- These are among the main determiners of what Python versions you should support

Try/Except: 2.4 and up (perhaps earlier)

- try:
- print(1/0)
- except ZeroDivisionError, extra:
- print('oops')

- This trips some very good programmers

Try/Except: 2.6 and up

- try:
- print(1/0)
- except ZeroDivisionError as extra:
- print('oops')

Try/Except: both 2.4-2.7 (perhaps earlier) and 3.x

- try:
- print(1/0)
- except ZeroDivisionError:
- extra = sys.exc_info()[1]
- print('oops')
- # I'm told this is slow on Pypy

String Formatting

- 2.7 & 3.0-3.5:

  print("abc{}ghi".format("def"))
- 2.4-2.7 (and much earlier), 3.0-3.5:

  print('abc%sghi' % 'def')
- “'F' strings”

- Warning: % string operator was nearly removed from 3.x, but it's likely here to stay

Imports

- try:
- 	# python 2
- 	from cStringIO import StringIO
- except ImportError:
- 	# python 3
- 	from io import BytesIO as StringIO

Division

- 2.x: 1 / 2 is 0
- 3.x: 1 / 2 is 0.5

- print(int(1/2))
- print(float(1)/2)

- 2.2 and up: from __future__ import division

range vs xrange

- In 2.x, range is always a list, and xrange is an iterator
- In 3.x, range is always an iterator, and xrange doesn't exist
- In 3.x, if you need a list, you can use list(range(5))
- The author sometimes defines a “my_range” to get consistent semantics, with just a for loop and a yield

Questions?

Questions?