

# Python 3 Compatibility

Lennart Regebro

<http://regebro.wordpress.com/>

PyCon PL 2008, Rybnik

Python 3.0 is coming!

# Python 3 will be incompatible



Python 3 will be incompatible



# Python 3 will be incompatible



Ya, really!



# Strategies to deal with incompatibility

For applications:  
Just port to Python 3

2to3 helps you port

Tests are really helpful

Libraries need to support both  
Python 2 and Python 3

For stable libraries:  
Just port to Python 3

For unstable libraries:  
Develop in Python 2 and run 2to3  
for Python 3 support

# Platform Extensions

No Plone user will switch  
to Python 3 until important  
extensions are available  
for Python 3

Nobody will make the extensions  
support Python 3 until they  
themselves move to Python 3

Dead lock



How did others do it?

Wanted: Gradual upgrade path

First support 2.5 and 2.6,  
then support 2.6 and 3.0,  
finally dropping 2.x completely

# Python 3 will be incompatible



Python 3 will be incompatible



# Python 3 will be incompatible

No, not really.



OH REALLY!?



Python 2.6 introduces quite a lot of forward compatibility!

Python 2.5:  
except Exception, e:

common mistake:  
except Exception1, Exception2:

correct:  
except (Exception1, Exception2):

Python 2.5:  
except Exception, e:

Python 3.0:  
except Exception as e:

Python 2.6:

Both works!

Python 2.x:  
raise Exception, "Message"

Python 2.x and 3.0:  
raise Exception("Message")

Python 2.5:  
 $3/2 == 1$

Python3.0:  
 $3/2 == 1.5$

Solutions:

$$3//2 == 1$$

```
from __future__ import division
```

$$3/2 == 1.5$$

Python2.5:

```
print >> file, "bla", "bla", "bla",
```

Python3.0:

```
print("bla", "bla", "bla",  
      file=file, ending='')
```

Python 2.6:

```
from __future__ import  
print_function
```

Python2.5:  
u"Üniçodê"

Python3.0:  
"Üniçodê"

Python 2.6:  
from \_\_future\_\_ import  
unicode\_literals

```
from __future__ import \  
    unicode_literals
```

```
try:
```

```
    str = unicode
```

```
except NameError:
```

```
    pass
```

```
isinstance(type("Üniçodê"), str)
```

Python2.5:

`dict.items()/dict.keys()/dict.values()`  
return lists

Python3.0:

`dict.items()/dict.keys()/dict.values()`  
return views

Python 2.5:

```
foo = bar.keys()  
foo.sort()
```

Python 2.4 - 3.0:

```
foo = sorted(bar.keys())
```

Removed:  
dict.iteritems()  
dict.itervalues()  
dict.iterkeys()

# Python 2.6 solution:

```
def get_items(d):  
    try:  
        return d.iteritems()  
    except AttributeError:  
        return d.items()
```

Python2.5:

```
open('filename', 'r').read()  
'A string\n'
```

```
open('filename', 'rb').read()  
'A string\n'
```

Python3.0:

```
open('filename', 'r').read()  
'A string\n'
```

```
open('filename', 'rb').read()  
b'A string\n'
```

Python 2.6:

```
b"A string"[5] == "i"
```

Python 3.0:

```
b"A string"[5] == 105
```

Solutions for 2.6 and 3.0:

```
bytearray(b"A list of bytes")
```

```
bytearray(open(file, "rb").read())
```

Python2.5:

```
open("unicodefile").read()  
'\xc3\x9cni\xc3\xa7od\xc3\xaa\n'
```

Python3.0:

```
'Üniçodê\n'
```

Solution for 2.6 and 3.0:

```
infile = open("unicodefile", "rb")  
uni = infile.read().decode(enc)
```

Preparing yourself now

Already in 2.6:

String exceptions are gone

“as” and “with” are keywords

Python 2.3:  
`alist.sort(cmp=func)`

Python 2.4 and later:  
`alist = sorted(alist, key=func)`

Use // when you want integer  
division.

Use  
from \_\_future\_\_ import division  
already now

Mark binary files with “rb” or “wb”

Maybe use buffer

Easiest way to do this?

1. 2to3
2. Make it run under 3.0
3. Backport to Python 2.6

2.6 compatible 2to3?

<http://code.google.com/p/python-incompatibility>

The future: Python 2.7

<http://code.google.com/p/python-incompatibility>