Porting your code to Python 3

Presented by
Alexandre Vassalotti

Overview

- **Introduction**
- **What's new?**
- **10 minutes break**
- **Migrating to Python 3**
- **Conclusion**

Introduction

- What is Python 3?
  - Not a complete rewrite
  - A backward-incompatible release
  - Clean-up old warts
- This presentation is about:
  - The major changes
  - How to port your code.

What's new?

- **print is a function**
- **Keyword-only arguments**
- **Unicode throughout**
- **New I/O library**
- **Standard library reorganization**
- **Iterators and views**
- **Special methods**
- **Syntax changes**

print is now a function!

- Not a big deal
- More flexible
  - The string sperator is customizable

```
>>> print("value=", number, sep="")
value=34
```

  - You can override the function

```
import builtins
builtins.print = my_custom_logger
```

print is now a function!

- The weird `>>sys.stderr` syntax is gone

Python 2

```
print >>sys.stderr, "system failure!"
```

Python 3

```
print("system failure!", file=sys.stderr)
```

Keyword-only arguments

- The keyword needs to be explicitly written out.
- Needed for variadic functions.

```
def print(*args, file=sys.stdout):
    ...
```

This is valid syntax in Python 3!

- Useful for forcing users to state their intent.

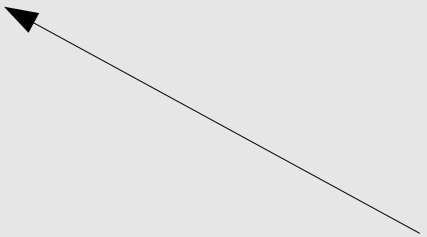
```
my_list.sort(key=lambda x: x[1])


sorted(my_list, reverse=True)
```

Keyword-only arguments

Syntax

```
def sorted(iterable, *, reverse=False, key=None):
    ...
```



The bare `*` indicates the following arguments are keyword-only.

Beware: the error message is surprising!

```
>>> sorted([1,2], True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sorted() takes exactly 1 positional argument (2 given)
```

What's new?

- print is a function
- Keyword-only arguments
- ▶ Unicode throughout
- New I/O library
- Standard library reorganization
- Iterators and views
- Special methods
- Syntax changes

Unicode

All strings use Unicode by default.

Python 2

```
u"hello world"


ur"\.write\(.*?\)"


unicode(anything)
```

Python 3

```
"hello world"


r"\.write\(.*?\)"


str(anything)
```

Unicode: bytes datatype

New `bytes()` datatype for data

```
b"this is data"


>>> bytes([1, 2, 3, 4])
b'\x01\x02\x03\x04'


>>> bytes("héhé", "utf-8")
b'h\xc3\xa9h\xc3\xa9'


>>> b"hello" + "world"
TypeError: can't concat bytes to str
```

Unicode: bytearray datatype

Mutable version for more fancy operations

```
b = bytearray(20)
file.readinto(b)
b.reverse()
b += b"hello world"
b[-1] = 0
```

Only 20 bytes are read into the buffer.

Unicode

- The distinction between data and text is not always clear.
- Many system APIs accept bytes as well.

```
>>> os.listdir(os.getcwd())
['eggs', 'monkey', 'spam']
>>> os.listdir(os.getcwdb())
[b'eggs', b'foo\x8f', b'monkey', b'spam']
```

Unicode: Other improvements

- `repr()` no longer escapes non-ASCII characters. It still escapes non-printable and control characters, however.

Python 2

```
>>> "allô!\n"
'all\xc3\xb4!\n'
```

Python 3

```
>>> "allô!\n"
'allô!\n'
>>> ascii("allô!\n")
"'all\\xf4!\\n'"
```

Recall that `repr()` is implicitly called at the interpreter prompt.

The old behaviour is still available if you need it.

Unicode: Other improvements

- Non-ASCII identifiers are supported

```
def holà(α, β):
    return α + β * 360
```

- But don't use them!
- Beware of characters that looks like latin letters.

```
>>> ascii("special")
"'\\u0455\\u0440\\u0435\\u0441\\u0456\\u04301'"
```

New I/O library

- Designed with Unicode in mind.
- Currently being rewritten in C for performance.
- Good news: you don't have to think about it.

```
with open("readme.txt", "w") as f:
    f.write("hello")

open("encoded.txt", "r", encoding="latin-1")
```

New I/O library

- 3-layers: raw, buffered and text.
- Great way to reuse code.

```
class StringIO(io.TextIOWrapper):
    def __init__(self, initial_value=""):
        super().__init__(io.BytesIO(), encoding="utf-16")
        self.write(initial_value)
        self.seek(0)
    def getvalue(self):
        return self.buffer.getvalue().decode("utf-16")
```

What's new?

- print is a function
- Keyword-only arguments
- Unicode throughout
- New I/O library
- Standard library reorganization
- Iterators and views
- Special methods
- Syntax changes

Standard library reorganization

- Remove the "silly old stuff"
- Modules renamed to be PEP-8 conformant.
- 2to3 handles most of the work for you.

Standard library reorganization

Python 2

```
import _winreg
import ConfigParser
import copy_reg
import Queue
import SocketServer


import __builtin__
import repr
import test.test_support
```

Python 3

```
import winreg
import configparser
import copyreg
import queue
import socketserver


import builtins
import reprlib
import test.support
```

PEP 8 violations

Poorly choosen names

Standard library reorganization

Python 2

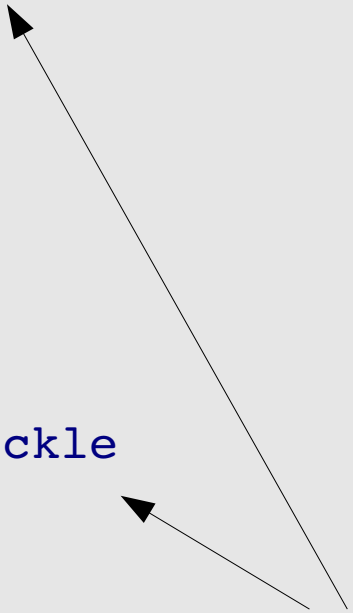
```
try:
  import cStringIO as StringIO
except ImportError:
  import StringIO


try:
  import cPickle as pickle
except ImportError:
  import pickle
```

Python 3

```
import io


import pickle
```



Use the optimized implementations automatically

Standard library reorganization

Python 2

```
import HTMLParser
import htmlentitydefs


import xmlrpclib
import DocXMLRPCServer
import SimpleXMLRPCServer


import dbhash
import dbm
import gdbm
import anydbm
import whichdb
```

Python 3

```
import html.parser
import html.entities


import xmlrpc.client
import xmlrpc.server


import dbm.bsd
import dbm.ndbm
import dbm.gnu
import dbm
```

Standard library reorganization

- Some modules were removed: `compiler`, `popen2` `htmllib`, `sgmllib`, `urllib`, `md5`, and many more.
- 2to3 does not handle these.
- Rewrite your code to avoid the deprecated modules.
- See PEP 3108 for replacements

Standard library reorganization

Side note: pickle data need to be regenerated.

```
$ python2.6
>>> import pickle
>>> pickle.dump(map, open("test.pickle", "wb"))
>>> pickle.load(open("test.pickle", "rb"))
<built-in function map>


$ python3.0
>>> import pickle
>>> pickle.load(open("test.pickle", "rb"))
Traceback (most recent call last):
  ...
ImportError: No module named __builtin__
```

Iterators and views

- Many APIs no longer return lists.
- `dict.keys(), .values()` and `.items()` return views.

```
>>> {1: 0}.keys()
<dict_keys object at 0x7ffdf8d53d00>
```

- A view is a set-like object.

```
for node in (graph.keys() - current_node):
    ...
```

Iterators and views

`map()`, `filter()`, `zip()` return iterators.

Python 2	Python 3
`a = map(lambda x: x[1], items)`	`a = [x[1] for x in items]`
`for name in map(str.lower, names):` `    ...`	*no change*
`a = filter(lambda n: n%2==0, nums)`	`a = [n for n in nums if n%2==0]`
`for key in filter(str.isdigit, keys):` `    ...`	*no change*
`dict(zip(sins, persons))`	*no change*

Iterators and views

- `xrange()` is the new `range()`.
- No changes are needed for most code.

What's new?

- print is a function
- Keyword-only arguments
- Unicode throughout
- New I/O library
- Standard library reorganization
- Iterators and views
- ▶ Special methods
- Syntax changes

Special methods: slicing

- `__getslice__` and friends are no longer supported.
- Use `__getitem__` instead.

```
class Array:
    def __getitem__(self, x):
        if isinstance(x, slice):
            start, stop, step = x.indices(len(self))
            ...
        else:
            try:
                index = x.__index__()
            except AttributeError:
                raise TypeError("indices must be integers")
            ...
```

Special methods: rich comparaisons

3-way comparaisons are gone.

Python 2

```
class Number:

    ...

    def __cmp__(self, other):
        if self.value == other.value:
            return 0
        elif self.value < other.value:
            return -1
        else:
            return 1

    ...
```

Special methods: rich comparaisons

Python 3

```
class Number:

    ...

    def __eq__(self, other):

        return self.value == other.value

    def __lt__(self, other):

        return self.value < other.value:

    def __gt__(self, other):

        return self.value > other.value:

    def __le__(self, other):

        return self.value <= other.value:

    def __ge__(self, other):

        return self.value >= other.value:

    ...
```

What's new?

- print is a function
- Keyword-only arguments
- Unicode throughout
- New I/O library
- Standard library reorganization
- Iterators and views
- Special methods
- ▶ Syntax changes

Syntax changes: exceptions

Python 2

```
try:
    with open(fn, 'r') as f:
        lines = list(f)
except (IOError, OSError), err:
    log_error(err)
```

Python 3

```
try:
    with open(fn, 'r') as f:
        lines = list(f)
except (IOError, OSError) as err:
    log_error(err)
```

This variable does not leak anymore.

Syntax changes: relative imports

```
json/
 |- encoder.py
 |- decoder.py
 |- __init__.py
```

In the `__init__.py` file:

```
from .encoder import JSONEncoder
from .decoder import JSONDecoder
```

Syntax changes: set and dict comprehension

- New syntax for set literals

```
{1, 3, 5}
```

```
set()
```

No syntax for empty sets.

- Set comprehension

```
{x for x in iterable}
```

- Dictionary comprehension

```
{k : v for k, v in iterable}
```

Syntax changes: many other niceties

- Extended iterable unpacking

```
a, b, *c = (1, 2, 3, 4, 5)
```

- The `nonlocal` declaration for accessing variables in outer scopes.
- Function annotations

```
def readinto(b: bytearray) -> int:
    ...
```

Migrating to Python 3

- Introduction
- Migration strategy
- Runtime warnings
- Backported features
- 2to3 source code translator

Introduction

- There is more than one way to do it.
- Porting C extensions is another beast.

Migration strategy

1. Improve your test suite.
2. Port your code to Python 2.6
3. Enable Python 3 warnings
4. Fix all the warnings
5. Modernize your code
6. Run 2to3

Code modernization

- Reduce the semantic gap
- Decrease the amount of work 2to3 has to do.
- Examples:
  - Use `dict.iterkeys()`, `xrange()`, etc
  - Avoid implict `str` and `unicode` coercion
  - Prefer `__getitem__` over `__getslice__`

Runtime Warnings

```
python2.6 -3 scriptname.py
```

- Warn about features that were removed in Python 3.
- Warn about changes 2to3 cannot handle automatically.

Runtime warnings

Demo

Backported features

Many features of Python 3 are available in 2.6

- New I/O library
- Unicode and bytes literals

```
from __future__ import unicode_literals
```

- Future built-in functions

```
from future_builtins import map, zip, hex
```

- New syntax for catching and raising exceptions
- ABCs, new `ast` module, advanced string formatting, rich comparaisons, etc

Backported features

Demo

2to3 source code translator

- Convert files or directories
- Generate a unified diff-formatted patch

```
2to3 project/ > python3.patch
```

- Can also fix doctests

```
2to3 -d tests.py
```

- Fixers can be run individually

2to3 source code translator

Limitations

- Handle only syntactic transformations—i.e, there is no type inference.
- Cannot fix things like:

```
m = d.has_key
if m(key):
    ...
```

```
attr = "has_key"
if getattr(d, attr)(key):
    ...
```

```
eval("d.has_key(key)")
```

2to3 source code translator

Demo

Upcoming changes

- %-style formatting may become deprecated.
- Performance improvements.
- New `importlib` module.

Conclusion

- Python 3 has lot of new features.
- There are many tools available to help you during the transition.
- Send bug reports to http://bugs.python.org/
- Subscribe to the **python-porting** mailing list for additional help.