Moving from Python 2 to Python 3

Introduction This document is aimed at Python 2 programmers wishing to start developing using Python 3. The document lists those objects and idioms that have changed

Removals and Replacements

An operator, an exception, a constant, some types, several global functions, several dictionary methods, and some `itertools` functions are gone

Python 2	*Python 3.1*
`if a <> b:`	`if a != b:`
`apply(fn, args)`	`fn(*args)`
`apply(fn, args, kwargs)`	`fn(*args, **kwargs)`
`if isinstance(x, basestring):`	`if isinstance(x, str):`
`x = buffer(y)`	`x = memoryview(y)` `# this is similar`
`if callable(x):`	`if hasattr(x, "__call__"):`
`fh = file(fname, mode)`	`fh = open(fname, mode)`
`if d.has_key(k):`	`if k in d:`
`for k, v in \ d.iteritems():`	`for k, v in d.items():`
`for k in d.iterkeys():`	`for k in d.keys():`
	`for k in d:`
`for v in \ d.itervalues():`	`for v in d.values():`
`for line in \ file.xreadlines():`	`for line in file:`
`x = input(msg)`	`x = eval(input(msg))`
`intern(s)`	`sys.intern(s)`
`f = itertools.ifilter( fn, seq)`	`f = filter(fn, seq)`
`m = itertools.imap( fn, seq)`	`m = map(fn, seq)`
`z = itertools.izip( seq1, seq2)`	`z = zip(seq1, seq2)`
`dir = os.getcwdu()`	`dir = os.getcwd()`
`s = raw_input(msg)`	`s = input(msg)`
`r = reduce(fn, seq)`	`r = functools.reduce( fn, seq)`
`reload(module)`	`imp.reload(module)`
`class MyErr( StandardError):`	`class MyErr( Exception):`
`sys.maxint`	`sys.maxsize`
`for i in xrange(n):`	`for i in range(n):`

Renamed Attributes and Methods

Implement `__bool__()` instead of `__nonzero__()` to return a custom class's truth value

Python 2	*Python 3.1*
`fn.func_closure`	`fn.__closure__`
`fn.func_code`	`fn.__code__`
`fn.func_defaults`	`fn.__defaults__`
`fn.func_dict`	`fn.__dict__`
`fn.func_doc`	`fn.__doc__`
`fn.func_globals`	`fn.__globals__`
`fn.func_name`	`fn.__name__`
`obj.method.im_func`	`obj.method.__func__`
`obj.method.im_self`	`obj.method.__self__`
`obj.method.im_class`	`obj.method.__class__`
`string.letters`	`string.ascii_letters`
`string.lowercase`	`string.ascii_lowercase`
`string.uppercase`	`string.ascii_uppercase`
`threading.Lock. \ acquire_lock()`	`threading.Lock. \ acquire()`
`threading.Lock. \ release_lock()`	`threading.Lock. \ release()`

`class Thing:`
`  def __init__(`
`  self, x):`
`  self.x = x`
`  def __nonzero__(`
`  self):`
`  return \`
`  bool(self.x)`
	`class Thing:`
	`  def __init__(`
	`  self, x):`
	`  self.x = x`
	`  def __bool__(self):`
	`  return bool(self.x)`

Exceptions

Catching exception objects requires the `as` keyword; raising exceptions with arguments requires parentheses; strings cannot be used as exceptions

Python 2	*Python 3.1*

`try:`
`  process()`
`except ValueError, \`
`  err:`
`  print err`
	`try:`
	`  process()`
	`except ValueError \`
	`  as err:`
	`  print(err)`

`try:`
`  process()`
`except (MyErr1,`
`  MyErr2), err:`
`  print err`
	`try:`
	`  process()`
	`except (MyErr1,`
	`  MyErr2) as err:`
	`  print(err)`

`raise MyErr, msg`	`raise MyErr(msg)`
`raise MyErr, msg, tb`	`raise MyErr(msg). \ with_traceback(tb)`
`raise "Error"`	`raise Exception( "Error")`
`generator.throw( MyErr, msg)`	`generator.throw( MyErr(msg))`
`generator.throw( "Error")`	`generator.throw( Exception("Error"))`

Renamed Modules

Data read from a URL, e.g., using `urllib.request.urlopen()` is returned as a `bytes` object; use `bytes.decode(encoding)` to convert it to a string. The `bsddb` (Berkeley DB library) is gone—but is avaliable from `pypi.python.org/pypi/bsddb3`. See PEP 3108 (`www.python.org/dev/peps/pep-3108`) for module renaming details

Python 2	*Python 3.1*
`import anydbm`	`import dbm`
`import whichdb`	
`import BaseHTTPServer`	`import http.server`
`import \` `  SimpleHTTPServer`	
`import CGIHTTPServer`	
`import __builtin__`	`import builtins`
`import commands`	`import subprocess`
`import ConfigParser`	`import configparser`
`import Cookie`	`import http.cookies`
`import cookielib`	`import http.cookiejar`
`import copy_reg`	`import copyreg`
`import dbm`	`import dbm.ndbm`
`import DocXMLRPCServer`	`import xmlrpc.server`
`import \` `  SimpleXMLRPCServer`	
`import dumbdbm`	`import dbm.dumb`
`import gdbm`	`import dbm.gnu`
`import httplib`	`import http.client`
`import Queue`	`import queue`
`import repr`	`import reprlib`
`import robotparser`	`import \` `  urllib.robotparser`
`import SocketServer`	`import socketserver`
`import \` `  test.test_support`	`import test.support`
`import Tkinter`	`import tkinter`

`import urllib`	`import \`
	`  urllib.request, \`
	`  urllib.parse, \`
	`  urllib.error`

`import urllib2`	`import \`
	`  urllib.request, \`
	`  urllib.error`

`import urlparse`	`import urllib.parse`
`import xmlrpclib`	`import xmlrpc.client`

Python 3.1 Idioms

Tuples need parentheses in comprehensions; metaclasses are set with the `metaclass` keyword; import the pickle and string I/O modules directly; `lambda` doesn't unpack tuples; set literals are supported (the empty set is `set()`; `{}` is an empty `dict`); sorting is fastest using a key function; `super()` is better; type-testing is more robust with `isinstance()`; use `True` and `False` rather than `1` and `0`

Python 2	*Python 3.1*
`L = [x for x in 3, 6]`	`L = [x for x in (3, 6)]`

`class A:`
`  __metaclass__ = \`
`  MyMeta`
`class B(MyBase):`
`  __metaclass__ = \`
`  MyMeta`
	`class A(`
	`  metaclass=MyMeta):`
	`  pass`
	`class B(MyBase,`
	`  metaclass=MyMeta):`
	`  pass`

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

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

`fn = lambda (a,): \` `  abs(a)`	`fn = lambda t: \` `  abs(t[0])`
	`fn = lambda a: abs(a)`
`fn = lambda (a, b): \` `  a + b`	`fn = lambda t: \` `  t[0] + t[1]`
	`fn = lambda a, b: a + b`
`S = set((2, 4, 6))`	`S = {2, 4, 6}`
`S = set([2, 4, 6])`	
`L = list(seq)` `L.sort()`	`L = sorted(seq)`

`words.sort(`
`  lambda x, y:`
`  cmp(x.lower(),`
`  y.lower()))`
	`words.sort(`
	`  key=lambda x:`
	`  x.lower())`

`class B(A):`
`  def __init__(self):`
`  super(B, self). \`
`  __init__()`
	`class B(A):`
	`  def __init__(self):`
	`  super(). \`
	`  __init__()`

`if type(x) == X:`	`if isinstance(x, X):`
`if type(x) is X:`	
`while 1:` `  process()`	`while True:` `  process()`

New in Python 3.1

Dictionary and set comprehensions; * unpacking; binary literals; bytes and bytearray types; bz2.BZ2File and gzip.GzipFile are context managers; collections.Counter dictionary type; collections.OrderedDict insertion-ordered dictionary type; decimal.Decimals can be created from floats

Python 2	*Python 3.1*

`d = {}`
`for x in range(5):`
`  d[x] = x**3`
	`d = {x: x**3`
	`  for x in range(5)}`

`S = set(` `  [x for x in seq])`	`S = {x for x in seq}`

Python 3.1

```
a, *b = (1, 2, 3)       # a==1; b==[2, 3]
*a, b = (1, 2, 3)       # a==[1, 2]; b==3
a, *b, c = (1, 2, 3, 4) # a==1; b==[2, 3]; c==4
x = 0b1001001           # x==73
s = bin(97)             # s=='0b1100001'
y = int(s, 2)           # y==97
u = "The €" # or: u = "The \u20ac"
            # or: u = "The \N{euro sign}"
v = u.encode("utf8") # v==b'The \xe2\x82\xac'
w = v.decode("utf8") # w=='The €'
x = bytes.fromhex("54 68 65 20 E2 82 AC")
    # x==b'The \xe2\x82\xac'
y = x.decode("utf8") # y=='The €'
z = bytearray(y)
z[-3:] = b"$"        # z==bytearray(b'The $')
with bz2.BZ2File(filename) as fh:
  data = fh.read()
counts = collections.Counter("alphabetical")
# counts.most_common(2)==[('a', 3), ('l', 2)]
d = collections.OrderedDict(
  (("x", 1), ("k", 2), ("q", 3)))
# list(d.keys())==['x', 'k', 'q']
dec = decimal.Decimal.from_float(3.75)
# dec==Decimal('3.75')
```

Special Methods

The slice methods (__delslice()__, __getslice()__, __setslice__) are gone; instead __delitem()__, __getitem()__, and __setitem__ are called with a slice object.

The methods __hex__() and __oct__() are gone; use hex() and oct(). To provide an integer, implement __index__().

General Notes

Python 3 often returns iterables where Python 2 returned lists. This is usually fine, but if a list is really needed, use the list() factory function. For example, given dictionary, d, list(d.keys()) returns its keys as a list. Affected functions and methods include dict.items(), dict.keys(), dict.values(), filter(), map(), range(), and zip().

Most of the types module's types (such as types.LongType) have gone. Use the factory function instead. For example, replace if isinstance(x, types.IntType) with if isinstance(x, int).

Comparisons are strict—x < y will work for compatible types (e.g., x and y are both numbers or both strings); otherwise raises a TypeError.

Some doctests involving floating point numbers might break because Python 3.1 uses David Gay's algorithm to show the shortest representation that preserves the number's value. For example, 1.1 in Python 2 and 3.0 is displayed as 1.1000000000000001, and in Python 3.1 as 1.1.

String Format Specifications

str.format() strings have one or more replacement fields of form: {*Name*!*Conv*:*Spec*}. *Name* identifies the object to format. Optional !*Conv* is: !a (ASCII repr() format), !r (repr() format), or !s (string format). Optional :*Spec* is of form:

: *Fill Align Sign* # 0 *Width* , .*Prec Type*

Fill is any character except }. *Align* is: < (left), > (right), ^ (center), or = (pad between sign and number). *Sign* is: + (force), – (- if needed), or " " (space or -). # prefixes ints with 0b, 0o, or 0x. 0 means 0-pad numbers. *Width* is the minimum width. The , means use grouping commas. .*Prec* is the maximum width for strs and number of decimal places for floats. *Type* is: % (percent), b (binary), d (decimal), e or E (exponential), f (float) g or G (general float) n (localized) o (octal), x or X (hex). Everything is optional, except that *Fill* requires *Align*.

```
"{:*=+10.1f}".format(12345.67) # '+**12345.7'
"{:*>+10.1f}".format(12345.67) # '**+12345.7'
"{:+010.1f}".format(12345.67)  # '+0012345.7'
"{:,.2f}".format(12345.678)    # '12,345.68'
```

informIT
An informIT.com publication by Mark Summerfield.