Python 3 へのコード変換について

2015.9.10

SCSK㈱　R＆Dセンター
OSS戦略企画室　OSS技術第二課

OpenStack では今 Python 3 対応の作業が進められています。Python の開発サイトでもトップページで Python 3 の特徴を述べて Python 3 の使用を奨励している様です。

https://www.python.org/

以降では Python 3 への移行の理由と、具体的に Python2.7 から Python 3.4 へのコード変換に必要となる情報について述べます。

Python 3 への置き換えの事情

- Python 2.7 のサポート期限

  サポートは2020年まで(当初より延長された)

  https://www.python.org/dev/peps/pep-0373/

- サポート内容

  サポートされるのは重大なセキュリティバグぐらいで、
  それ以外のバグは基本的に Python 3 で修正。

  Python 3 からのバックポートも余り期待できない模様。

  https://www.python.org/dev/peps/pep-0466/

Python 3 への置き換えの事情

- Python 3 の機能

  全体的に言語体系が整理されたと言う印象。

  文字型の扱いが明確になった。また文字コードがunicodeベースになった。

  直接リストを返さなくなった。似た機能が統一された。

  新しい機能として ファイル記述子の非継承、asyncio

  モジュールの追加等。

  (本ドキュメントは移植に関して説明するもので新機能については言及しない)

  http://docs.python.jp/3.4/whatsnew/

Python 3 への置き換えの事情

ディストリビューションの状況

- RedHat
  - RHEL 6: Python 2.6
  - RHEL 7: Python 2.7
- Ubuntu
  - Ubuntu 12.04: Python 2.7, (Python 3.2)
  - Ubuntu 14.04: Python 2.7, (Python 3.4)
  - Ubuntu 16.04: Python 3.4 がデフォルト?
- OpenSUSE
  - OpenSUSE 11.4: Python 2.7
  - OpenSUSE 12.3: Python 2.7, (Python 3.3)
  - OpenSUSE 13.2: Python 2.7, (Python 3.4)

Python 3 への置き換えの事情

Python の開発状況

- 2.8 のリリースは無し

  https://www.python.org/dev/peps/pep-0404/
- 最新リリース 3.4.3

  https://www.python.org/dev/peps/pep-0429/
- 3.5　9月リリース予定

  https://www.python.org/dev/peps/pep-0478/
- 3.6 開発バージョン

  https://www.python.org/dev/peps/pep-0494/

  https://docs.python.org/dev/index.html

リプレースの状況

Python 3 としては Python 3.0 が 2008/12/3 にリリースされて以来、時間が立つが移植作業はそれ程活発では無いように見える。

代表的な Python ベースの下記のソフトを見るとサポートは半々ぐらい。

Anaconda:	Python2
Django:	Python 2, 3
Mercurial:	Python 2
OpenStack:	Python 3対応中
Trac:	Python 2.7
Zope:	Python 2, 3

pypi に登録されているパッケージで Python 3 対応のものはおよそ 18 % 程度(2015/9/4 時点)。

リプレースの状況

移植が進まない要因

- Python 3 との互換性の問題

  Python 2 は後方互換性に配慮していたが 、Python 3 では互換性を捨て、よりクリーンなコードを目指した。その為置き換えが結構大変。

  未サポートとなったOSもある(Windows 2000, OS/2, VMS)

- 特に python 2.7 でも困らない?
- Python 2.7 のサポート期限の延期

  当初 2015年までのサポートだったが、現在 2020年までに延期されしばらくは安泰。

移植の必要な処理

Python 3 で変わった主なコード

- Python 2 で削除されたメソッド
- 戻り値の変更
- 文字コード
- その他

  削除された print 文

  除算の演算子の扱い

  BaseException.message 属性の削除

移植の必要な処理

- Python 2 で削除されたメソッド
  - dict

    dict.iteritems() --> dict.items()

    dict.viewitems()

    dict.iterkeys() --> dict.keys()

    dict.viewkeys()

    dict.itervalues() --> dict.values()

    dict.viewvalues()

移植の必要な処理

- Python 2 で削除されたメソッド
  - xrange() --> range()
  - cmp() --> 無し
  - unicode() --> str()
  - iterator.next() --> next()
  - contextlib.nested --> 無し
  - itertools
    - itertools.izip() --> zip()
    - itertools.ifilter() --> filter()
    - itertools.imap() --> map()

移植の必要な処理

●戻り値の変更

リストからイテレータ

- dict

  dict.items()

  dict.keys()

  dict.values()

  * dictionary の場合はビューオブジェクトが返る
- map()
- range()
- zip()

移植の必要な処理

str型とbytes型

- struct の s 書式型
- subprocess.Popen
- hmac

文字型に関連する変更はこの他、多数ありそうだが
個々にドキュメントで確認するしか無さそう。

移植の必要な処理

- 文字コード

  python 2 の文字は str 型と unicode 型がありstr型は ascii ベース。

  str 型と unicode 型を連結してもエラーにならない等、型としては曖昧な部分がある。

  python 3 での文字は str 型と新たに bytes 型ができ

  unicode ベース。

  str型 と bytes型は明確に区別され連結や、encode, decode も型が合わなければエラーとなる。

  文字列を返すメソッドの in/out は要注意。

  バイトデータのインデックス参照の結果は Python 3 では文字ではなく数字となる。

移植の必要な処理

- その他
  - 削除された print 文

    print 文は削除され、print() 関数のみとなった。print は予約語からも外された。

    関数である為 print には括弧が必要。

    print 文の後ろの式の部分は引数として渡す。
  - 除算の扱い

    '/' の結果は Python2 では整数だが Python 3 では浮動小数点が返る。

    整数を返して欲しい場合は '//' を使用する。
  - BaseException.message 属性の削除

    代わりに BaseException.args attribute が使用できる

移植に使えるユーティリティ

移植ライブラリ

- __future__ (python 標準のモジュール)
- six

変換ツール

- 2to3(python 付属のツール)
- sixer
- context_unnester
- python-modernize
- 2to6
- python-future

移植に使えるユーティリティ

- __future__

  Python 2 上で Pyrhon 3 の機能を利用できる様にする。

  ‘from __future__ import feature-name’

  上記の様に feature を import する事で利用する。

  http://docs.python.jp/3.4/library/__future__.html?highlight=__future__#__future__

  2.7 では次の feature を使用できる。

移植に使えるユーティリティ

- division

  ‘//’ の結果を浮動小数点で返す。

- absolute_import

  明示的な相対インポートのサポート。

- print_function

  print() 関数を使用可能にする。

- unicode_literals

  プレフィックスの無い文字列リテラルを unicode とみなす。

移植に使えるユーティリティ

- six

  Python 2 / 3 の両方で動作するラッパーメソッドを提供するライブラリ。

  命名はPython 2 / 3 をサポートする事から 2x3=6 に由来するらしい。

  http://pythonhosted.org/six/

six は 800行ほどの Python スクリプト。下記の様に単に Python のバージョンを判定して処理の分岐を行っている部分もある。

six.py からの抜粋

```
…
# Useful for very coarse version differentiation.
PY2 = sys.version_info[0] == 2
PY3 = sys.version_info[0] == 3

if PY3:
    string_types = str,
    integer_types = int,
    class_types = type,
    text_type = str
    binary_type = bytes

    MAXSIZE = sys.maxsize
else:
    string_types = basestring,
    integer_types = (int, long)
    class_types = (type, types.ClassType)
    text_type = unicode
    binary_type = str
…
```

移植に使えるユーティリティ

- 2to3

  python に付属するコード変換ツール。

  実体は lib2to3 ライブラリ。

  Python 3 への変換ツールなので変換後のコードは
  Pyrhon 2 では動作しない。

  lib2to3 を使って変換ツールを自作する事も可能だが、
  このAPIは安定しておらず劇的な変更の覚悟が必要。

  http://docs.python.jp/3.4/library/2to3.html

移植に使えるユーティリティ

変換パターンに対応した変換名を指定する事で特定の変換のみを行う事ができる。

“2to3 -l“ 変換プログラムの一覧表示

“2to3 -f fix1 -f fix2 script-name“ 特定の変換を適用

“2to3 -x fix1 -x fix2 script-name“ 特定の変換を適用外

移植に使えるユーティリティ

- sixer

  six で変換できる箇所のみを修正する。ツールは Python 3 上でのみ動作する。

  OpenStack 向けに、OpenStack のメンバが作成したもの。

  https://pypi.python.org/pypi/sixer

- context_unnester

  contextlib.nested の変換のみに特化したツール。

  これも OpenStack 向けに、OpenStack のメンバが作成したもの。

  https://github.com/adrienverge/context_unnester

移植に使えるユーティリティ

- python-modernize

  lib2to3 を利用し 2to3 相当の変換と six を使用した変換機能をサポートする。

  2to3 同様変換パターンを指定できる。また'—no-six' を指定すると six を使った変換は行わない。

  http://python-modernize.readthedocs.org/en/latest/

- 2to6

  python-modernize と凡そ同じ機能。2013/7/28 のコミット以来更新が無い。

  pypi には未登録。

  https://github.com/limodou/2to6

移植に使えるユーティリティ

- python-future

  下記の変換を行う。

  Python 2 --> Python 2/3

  Python 3 --> Python 2/3

  python-modernize 同様 lib2to3 を利用し 2to3 相当の変換と six 機能を使用した変換を行う。

  それに加え独自のライブラリが実装されている。

  2to3 同様変換パターンを指定できる。

  python-future に依存したコードが挿入される事もある為、変換プログラム単独では動作しないかも知れない。

  http://python-future.org/

移植に使えるユーティリティ

単純に Python 3 用に変換する場合は 2to3 で変換後、足りない部分を修正。

Python 2, 3 双方で動作させたい場合は six を使用する事になる。ツールはpython-modernaize が全体的に機能を網羅しており、ツール依存の部分も無い為、無難だが 2to3 に比べ変換したりない部分がある。

2to3 適用後 sixer や python-modernaize で six 部分の変換を行うと良いかも知れない。

変換例を以下に示す。

➢ 削除されたメソッドのケース(python-modernize 使用)

```
$ cat samp_remove.py
import itertools

d1 = {'a': '1', 'b': '2'}

b = b'abcd'
u = unicode(b)
if u == b:
   print "equal binary u=%s" % u
else:
   print "not binary u=%s" % u

for k, v in d1.iteritems():
   print "iteritems() k=%s v=%s" % (k, v)

ivals = d1.itervalues()
while True:
   n = ivals.next()
   print "iterator.next() n=%s" % n
   if n == '2':
      break

for i in xrange(2):
   print "xrange() i=%d" % i

it = itertools.imap(str, [1, 2])
```

```
for s in it:
   print "itertools.imap() s=%s" % s

a = 1
b = 2
if cmp(a, b) < 0:
   print "a < b"
$ python-modernize -w samp_remove.py
root: Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
…
RefactoringTool: Refactored samp_remove.py
--- samp_remove.py      (original)
+++ samp_remove.py      (refactored)
@@ -1,32 +1,36 @@
+from __future__ import absolute_import
+from __future__ import print_function                                  -- 1)
 import itertools
+import six
+from six.moves import range

 d1 = {'a': '1', 'b': '2'}

 b = b'abcd'
-u = unicode(b)
+u = six.text_type(b)                                                   -- 2)
 if u == b:
-    print "equal binary u=%s" % u
+    print("equal binary u=%s" % u)                                     -- 3)
 else:
```

```
-    print "not binary u=%s" % u
+    print("not binary u=%s" % u)

-for k, v in d1.iteritems():
-    print "iteritems() k=%s v=%s" % (k, v)
+for k, v in six.iteritems(d1):                                  -- 4)
+    print("iteritems() k=%s v=%s" % (k, v))

-ivals = d1.itervalues()
+ivals = six.itervalues(d1)
 while True:
-    n = ivals.next()
-    print "iterator.next() n=%s" % n
+    n = next(ivals)
+    print("iterator.next() n=%s" % n)
     if n == '2':
         break

-for i in xrange(2):
-    print "xrange() i=%d" % i
+for i in range(2):
+    print("xrange() i=%d" % i)

 it = itertools.imap(str, [1, 2])                               -- 5)
 for s in it:
-    print "itertools.imap() s=%s" % s
+    print("itertools.imap() s=%s" % s)

 a = 1
```

```
b = 2
 if cmp(a, b) < 0:                                                    -- 5)
-    print "a < b"
+    print("a < b")
RefactoringTool: Files that were modified:
RefactoringTool: samp_remove.py
$
編集
$ diff -u samp_remove.py.bak samp_remove.py
--- samp_remove.py.bak  2015-09-03 06:29:19.142521670 +0000
+++ samp_remove.py      2015-09-03 06:29:51.166738014 +0000
@@ -26,11 +26,11 @@
 for i in range(2):
     print("xrange() i=%d" % i)

-it = itertools.imap(str, [1, 2])                                     -- 6)
+it = map(str, [1, 2])
 for s in it:
     print("itertools.imap() s=%s" % s)

 a = 1
 b = 2
-if cmp(a, b) < 0:                                                    -- 6)
+if a < b:
     print("a < b")
$
```

１）Python 2 で print() 関数を使える様に追加
２）Python 2 では unicode()、Python 3 では str() に展開
３）print 関数に変換
４）Python 2 では iter(dict.iteritems())、Python 3 では iter(dict.items()) に展開
dict_items オブジェクトはイテレータとは属性が異なる為イテレータに変換
５）変換が必要だが未変換
６）手動で変更

```
$ python2 samp_remove.py
equal binary u=abcd
iteritems() k=a v=1
iteritems() k=b v=2
iterator.next() n=1
iterator.next() n=2
xrange() i=0
xrange() i=1
itertools.imap() s=1
itertools.imap() s=2
a < b
$ python3 samp_remove.py
not binary u=b'abcd'
iteritems() k=a v=1
iteritems() k=b v=2
iterator.next() n=1
iterator.next() n=2
xrange() i=0
xrange() i=1
itertools.imap() s=1
itertools.imap() s=2
a < b
$
```

➢ 削除されたメソッドのケース(2to3 使用)

```
$ 2to3 -w samp_remove.py
root: Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
…
RefactoringTool: Refactored samp_remove.py
--- samp_remove.py      (original)
+++ samp_remove.py      (refactored)
@@ -3,30 +3,30 @@
 d1 = {'a': '1', 'b': '2'}

 b = b'abcd'
-u = unicode(b)
+u = str(b)                                                    -- 1)
 if u == b:
-    print "equal binary u=%s" % u
+    print("equal binary u=%s" % u)
 else:
-    print "not binary u=%s" % u
+    print("not binary u=%s" % u)

-for k, v in d1.iteritems():
-    print "iteritems() k=%s v=%s" % (k, v)
+for k, v in d1.items():                                       -- 2)
+    print("iteritems() k=%s v=%s" % (k, v))

-ivals = d1.itervalues()                                       -- 3)
+ivals = iter(d1.values())
```

```
while True:
-    n = ivals.next()
-    print "iterator.next() n=%s" % n
+    n = next(ivals)
+    print("iterator.next() n=%s" % n)
     if n == '2':
         break

-for i in xrange(2):
-    print "xrange() i=%d" % i
+for i in range(2):
+    print("xrange() i=%d" % i)

-it = itertools.imap(str, [1, 2])
+it = map(str, [1, 2])                                          -- 4)
 for s in it:
-    print "itertools.imap() s=%s" % s
+    print("itertools.imap() s=%s" % s)

 a = 1
 b = 2
 if cmp(a, b) < 0:
-    print "a < b"
+    print("a < b")
$
```

１）str() に変換
２）dict.items() に変換
３）iter(d1.values()) に変換
４）map() に変換(python-modernize() では変換されていなかった)

➢ 戻り値がリストからイテレータに変わったケース

```
$ cat samp_iterator.py
d1 = {'a': '1', 'b': '2', 'c': '3'}
for k, v in d1.items():
    print "items() k=%s" % k

s = map(str, [1, 2, 3])
print "map=%s" % s

s = map(str, [1, 2, 3])[1]
print "map=%s" % s
$ python-modernize -w samp_iterator.py
root: Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
…
RefactoringTool: Refactored samp_iterator.py
--- samp_iterator.py    (original)
+++ samp_iterator.py    (refactored)
@@ -1,9 +1,12 @@
+from __future__ import absolute_import
+from __future__ import print_function
+from six.moves import map
 d1 = {'a': '1', 'b': '2', 'c': '3'}
-for k, v in d1.items():
-    print "items() k=%s" % k
+for k, v in list(d1.items()):
+    print("items() k=%s" % k)

-s = map(str, [1, 2, 3])
```

```
-print "map=%s" % s
+s = list(map(str, [1, 2, 3]))
+print("map=%s" % s)

 s = map(str, [1, 2, 3])[1]
-print "map=%s" % s
+print("map=%s" % s)
RefactoringTool: Files that were modified:
RefactoringTool: samp_iterator.py
$
編集
$ diff -u samp_iterator.py.bak samp_iterator.py
--- samp_iterator.py.bak        2015-09-02 10:36:48.880141958 +0000
+++ samp_iterator.py    2015-09-02 10:43:56.027294096 +0000
@@ -1,9 +1,12 @@
+from __future__ import absolute_import
+from __future__ import print_function
+from six.moves import map
 d1 = {'a': '1', 'b': '2', 'c': '3'}
-for k, v in d1.items():
-    print "items() k=%s" % k
+for k, v in list(d1.items()):                                   -- 1)
+    print("items() k=%s" % k)

-s = map(str, [1, 2, 3])
-print "map=%s" % s
+s = list(map(str, [1, 2, 3]))
+print("map=%s" % s)
```

```
-s = map(str, [1, 2, 3])[1]
-print "map=%s" % s
+s = list(map(str, [1, 2, 3]))[1]
+print("map=%s" % s)
$ python2 samp_iterator.py
items() k=a
items() k=c
items() k=b
map=['1', '2', '3']
map=2
$ python3 samp_iterator.py
items() k=b
items() k=c
items() k=a
map=['1', '2', '3']
map=2
$
```

1）ここでは list() で変換しているが、変換の必要はない

➢ str 型と bytes 型

```
$ cat samp_str.py
import struct

p = struct.pack('2sisi', 'aa', 3, 'b', 100)
print "struct.pack()=", struct.unpack('2sisi', p)

s = b'abcde'
b_cmp = 'c'
if s[2] == b_cmp:
   print "s[2] equal %s:" % b_cmp, s[2]
else:
   print "s[2] not equal %s:" % b_cmp, s[2]
$ python-modernize -w samp_str.py
…
$
編集
$ diff -u samp_str.py.bak samp_str.py
--- samp_str.py.bak     2015-09-03 23:48:54.612342691 +0000
+++ samp_str.py 2015-09-03 23:51:39.613462446 +0000
@@ -1,11 +1,13 @@
+from __future__ import absolute_import
+from __future__ import print_function
 import struct

-p = struct.pack('2sisi', 'aa', 3, 'b', 100)
-print "struct.pack()=", struct.unpack('2sisi', p)
```

```
+p = struct.pack('2sisi', b'aa', 3, b'b', 100)                    -- 1)
+print("struct.pack()=", struct.unpack('2sisi', p))

 s = b'abcde'
-b_cmp = 'c'
+b_cmp = 99                                                       - - 2)
 if s[2] == b_cmp:
-    print "s[2] equal %s:" % b_cmp, s[2]
+    print("s[2] equal %s:" % b_cmp, s[2])
 else:
-    print "s[2] not equal %s:" % b_cmp, s[2]
+    print("s[2] not equal %s:" % b_cmp, s[2])
$ python2 samp_str.py
struct.pack()= ('aa', 3, 'b', 100)
s[2] not equal 99: c
$ python3 samp_str.py
struct.pack()= (b'aa', 3, b'b', 100)
s[2] equal 99: 99
$
```

1）書式 s の引数を bytes 型に変更
2）Python 3 では bytes のインデックスでの参照は数字になる為変更

編集

```
$ diff -u samp_str.py.bak samp_str.py
--- samp_str.py.bak3    2015-09-03 23:51:39.613462446 +0000
+++ samp_str.py 2015-09-03 23:53:17.390125938 +0000
@@ -1,12 +1,16 @@
 from __future__ import absolute_import
 from __future__ import print_function
+import six
 import struct

 p = struct.pack('2sisi', b'aa', 3, b'b', 100)
 print("struct.pack()=", struct.unpack('2sisi', p))

 s = b'abcde'
-b_cmp = 99
+if six.PY3:
+    b_cmp = 99
+else:
+    b_cmp = 'c'
 if s[2] == b_cmp:
     print("s[2] equal %s:" % b_cmp, s[2])
 else:
$ python2 samp_str.py
struct.pack()= ('aa', 3, 'b', 100)
s[2] equal c: c
$ python3 samp_str.py
struct.pack()= (b'aa', 3, b'b', 100)
s[2] equal 99: 99
$
```

➢ contextlib.nested

```
$ cat samp_nested.py
import contextlib
import mock

with contextlib.nested(
   mock.patch.object(self.ovs, 'set_protocols'),
   mock.patch.object(self.ovs, 'set_controller'),
) as (mock_set_protocols, mock_set_controller):
   self.assertEqual(mock_set_protocols.call_count, 1)
   self.assertEqual(mock_set_controller.call_count, 1)
$ cp -p samp_nested.py samp_nested.py.orig
$ context_unnester/context_unnester.py samp_nested.py
samp_nested.py
$ context_unnester.py samp_nested.py
samp_nested.py
$ cat samp_nested.py
import mock

with mock.patch.object(self.ovs, 'set_protocols') as mock_set_protocols,¥
       mock.patch.object(self.ovs, 'set_controller') as mock_set_controller:
   self.assertEqual(mock_set_protocols.call_count, 1)
   self.assertEqual(mock_set_controller.call_count, 1)
$
```

➢ その他

```
$ cat samp_misc.py
import sys
i = 5 / 2
print >> sys.stderr, "i=",
print >> sys.stderr, i
i = 5 // 2
print >> sys.stderr, "i=", i

try:
   raise IOError, 'some errors occured'
except IOError, ex:
   print "ex=%s" % ex.message
$ python-modernize -w samp_misc.py
root: Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
…
$
編集
$ diff -u samp_misc.py.bak samp_misc.py
--- samp_misc.py.bak    2015-09-02 04:25:08.438598822 +0000
+++ samp_misc.py       2015-09-02 04:27:31.175616793 +0000
@@ -1,11 +1,13 @@
+from __future__ import absolute_import
+from __future__ import print_function
 import sys
 i = 5 / 2
-print >> sys.stderr, "i=",
```

```
-print >> sys.stderr, i                                    - 1)
+print("i=", end=' ', file=sys.stderr)
+print(i, file=sys.stderr)
 i = 5 // 2
-print >> sys.stderr, "i=", i
+print("i=", i, file=sys.stderr)

 try:
-    raise IOError, 'some errors occured'
-except IOError, ex:                                        -- 2)
-    print "ex=%s" % ex.message
+    raise IOError('some errors occured')
+except IOError as ex:
+    print("ex=%s" % ex.args[0])
$ python2 samp_misc.py
i= 2
i= 2
ex=some errors occurred
$ python3 samp_misc.py
i= 2.5                                                     -- 3)
i= 2
ex=some errors occured
$
```

1）末尾の出力を end 引数に、出力先を file 引数に変換

2）ex を as ex に。message 属性は削除されたので args に変換

3）Python 3 では '/' の結果は浮動小数点になる

Openstack の現状

Openstack の Python 3 対応状況は下記のサイトで参照できる。

https://wiki.openstack.org/wiki/Python3

主なライブラリ関係は作業終了となっているが、主要コンポーネントは作業中のものが多い。

終了済みのものも unit test でエラーが出ない程度で、実際に OpenStack として起動して確認していないものと思われる。

(以下のイメージは 2015/8/26 現在のもの)

Openstack の現状

Common Libraries (Oslo Projects)

For the list of Common Libraries, see http://git.openstack.org/cgit/openstack/governance/tree/reference/programs.yaml#n160

Project	Python 3 compatibility	Comment
cliff	Yes	
oslo.concurrency	Yes	
oslo-incubator	Yes	py34 gate is voting
oslo.config	Yes	
oslo.context	Yes	
oslo.db	Yes	PyMySQL driver is now used by default for MySQL. setup.cfg contains the Python 3 classifier.
oslo.i18n	Yes	
oslo.log	Yes	
oslo.messaging	Yes, except of Qpid & AMQP drivers. py34 is not more voting because of ubuntu still uses 3.4.0 which has a severe bug	oslo.messaging 1.11.0 fully works on Python 3, except of Qpid and AMQP 1.0 transports which are Python 2 only. The py34 gate is voting.
oslo.middleware	Yes	
oslo.rootwrap	Yes	
oslo.serialization	Yes	
oslosphinx	?	The project only contains two short .py files, it looks to be Python 3 compatible. Is Sphinx Python 3 compatible?

Openstack の現状

OpenStack applications

Project	Python 3 compatibility	py34 gate	Comments
aodh	Yes	voting	Patch: Port remaining tests to Python 3.
ceilometer	Yes	voting	Victor Stinner is working on porting Ceilometer to Python 3. Patches: • Patches with py3 topic
gnocchi	Yes	Yes	
ironic	Yes	Yes	Python 3.4 unit tests are now being run for openstack/ironic. The unit tests are a voting job. Thanks to Victor Sergeyev for all of his work to update the Ironic code to make it pass the unit tests using Python 3.4: Run tests in py34 environment
Rally	Yes	voting	A big thank to Andrey Kurilin for a lot of work in this direction
cinder	work in progress	voting	cinder-python3 blueprint (written by Victor Stinner) has been accepted for Liberty. Patches: bp/cinder-python3 topic.
glance	work in progress	voting	Victor Stinner is working on porting Glance to Python3. Patches: Patches for glance (topic: py3). test_migrations is blocked by a bug in oslo.db (Fix test_migrations on Python 3) / sqlalchemy-migrate (Add VerNum.__index__() for Python 3 support).

Openstack の現状

heat	work in progress	voting	Python34 Support spec (by Sirushti Murugesan) accepted for Liberty. Patches: bp/heat-python34-support topic.
keystone	work in progress	voting	The spec Add spec for python-3 compatibility (by Morgan Fainberg) was accepted for Liberty. See also the blueprint python3 started by Dolph Mathews on 2014-07-22; David Stanek is working on it. Patches: bp/python3 topic. pysaml2 3.0 now supports Python 3. Aslo blocked by ldappool and python-ldap. pyldap is a fork of python-ldap which adds Python 3 support. python-ldap may be replaced with ldap3, but it requires more work.
neutron	work in progress	voting	Porting to Python 3 spec (by Cyril Roelandt) accepted for Liberty, port in progress. Patches: • Patches of the blueprint neutron-python3
nova	work in progress	voting	Adding Python 3.4 support to Nova spec (by Victor Stinner) accepted for Liberty. Port in progress. Patches: • master (bp/nova-python3). See also Py3.4 Compatibility in Liberty nova priorities tracking.
sahara	work in progress	voting	

サポート

Python 3 に関する FAQ が下記にまとめられている。

http://python-notes.curiousefficiency.org/en/latest/python3/questions_and_answers.html

ポーティングに関するトラブルや質問は下記 ML で尋ねる事が出来る。

https://mail.python.org/mailman/listinfo/python-porting

参考資料

http://docs.python.jp/3.4/whatsnew/

http://docs.python.jp/3.4/howto/pyporting.html

http://getpython3.com/