Python 2 and 3 Compatibility

In a single code base

About Me

- Software Engineer
- FOSS contributor
- Aeronautical Eng. Student
- @Captain_Joannah

`

OUTREACHY

LABO REM US*

Ceph

fintech
transformative technology

Rails Girls Summer of Code

Qutebrowser

One more thing

I love Python

Recent excitements around pypy, static typing and other stuff.

Also

Javascript, Not that Much

Python 2 and 3 Compatibility

In a single code base

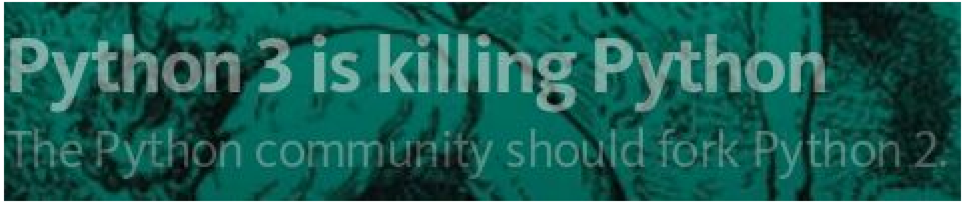
Python 2.x

- In October 2000 , python 2.0 was released.
- With many prime features and everything.
- Later became public and community backed.
- Evolved to many versions aka 2.x.

Python 3.x

- In December 2008 python 3 (3k , 3000) was released.
- It was very backwards incompatible.
- Improved major design shortcomings in the language.
- Some feature improvements necessitated a major version number for the language (According to the core team).

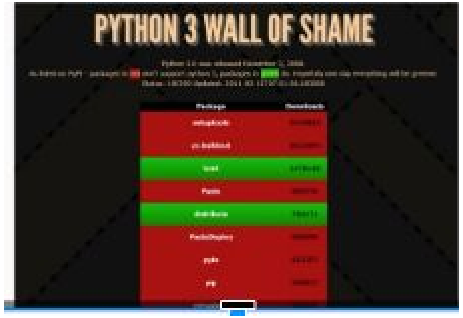
Some of the Reactions!!!!!!



Python 3 is killing Python

The Python community should fork Python 2.

With time ....



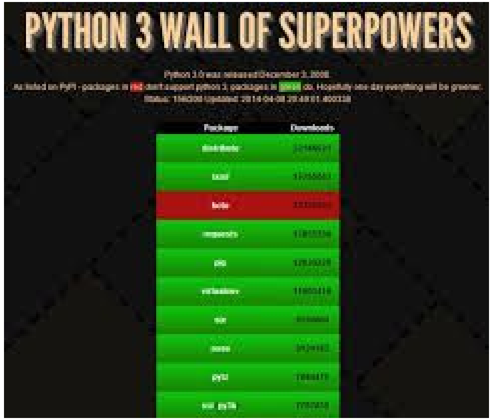
PYTHON 3 WALL OF SHAME

Python 3.0 was released December 3, 2008

As listed on PyPI - packages in 🟥 don't support python 3, packages in 🟩 do. Hopefully one day everything will be greener.

Status: [illegible] Updated: 2011-[illegible]

Package	Downloads
[illegible]	[illegible]
zc.buildout	[illegible]
[illegible]	[illegible]
Paste	[illegible]
distribute	[illegible]
PasteDeploy	[illegible]
pytz	[illegible]
PIL	[illegible]



PYTHON 3 WALL OF SUPERPOWERS

Python 3.0 was released December 3, 2008.

As listed on PyPI - packages in 🟥 don't support python 3, packages in 🟩 do. Hopefully one day everything will be greener.

Status: 156/200 Updated: 2014-04-08 20:48:01.400338

Package	Downloads
distribute	[illegible]
[illegible]	[illegible]
boto	[illegible]
requests	[illegible]
pip	[illegible]
virtualenv	[illegible]
six	[illegible]
nose	[illegible]
pytz	[illegible]
[illegible]	[illegible]

The future of python 2

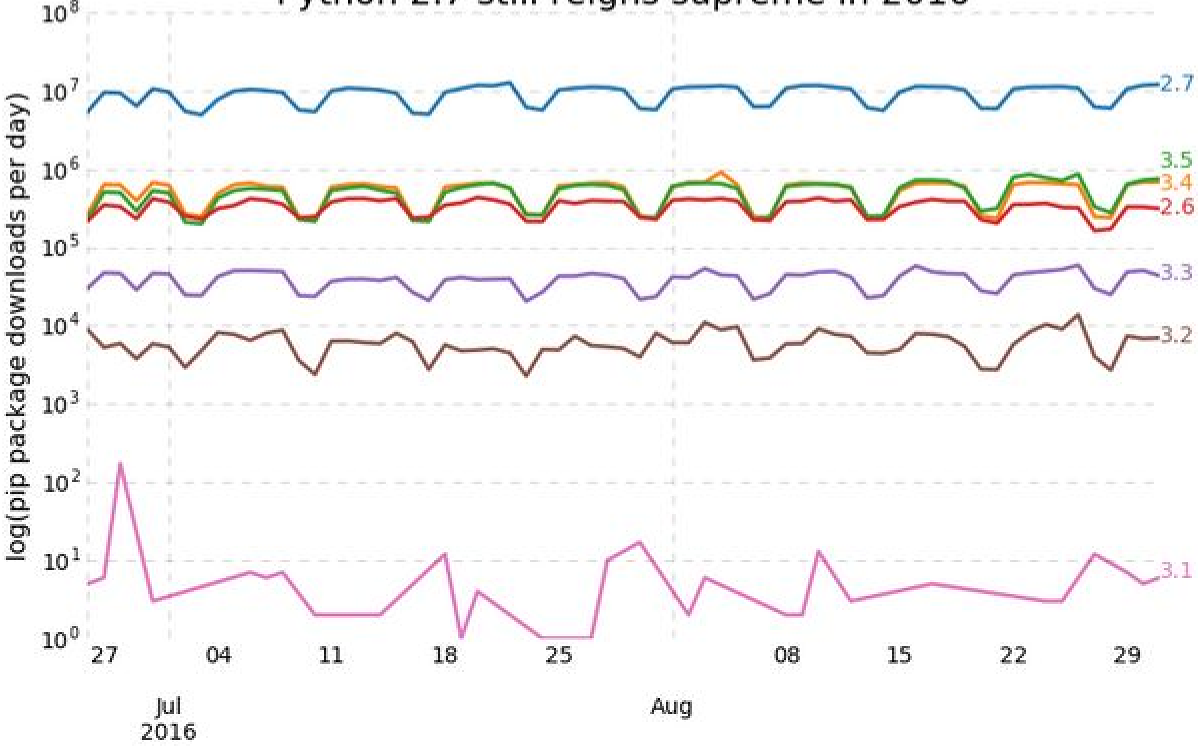
Python 2.x is legacy, Python 3.x is the present and future of the language.

PSF Stand

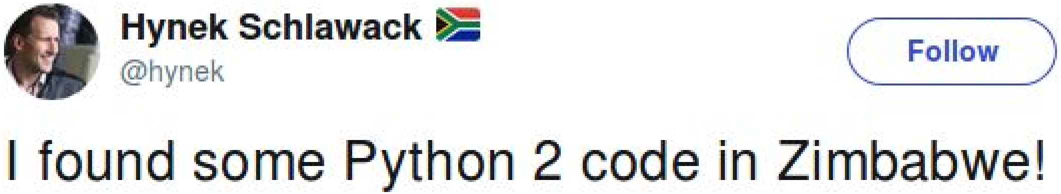
The Dilemma

Python 3 is already the future, but python 2 is still in use and will continue to be used for sometime

Python 2.7 still reigns supreme in 2016

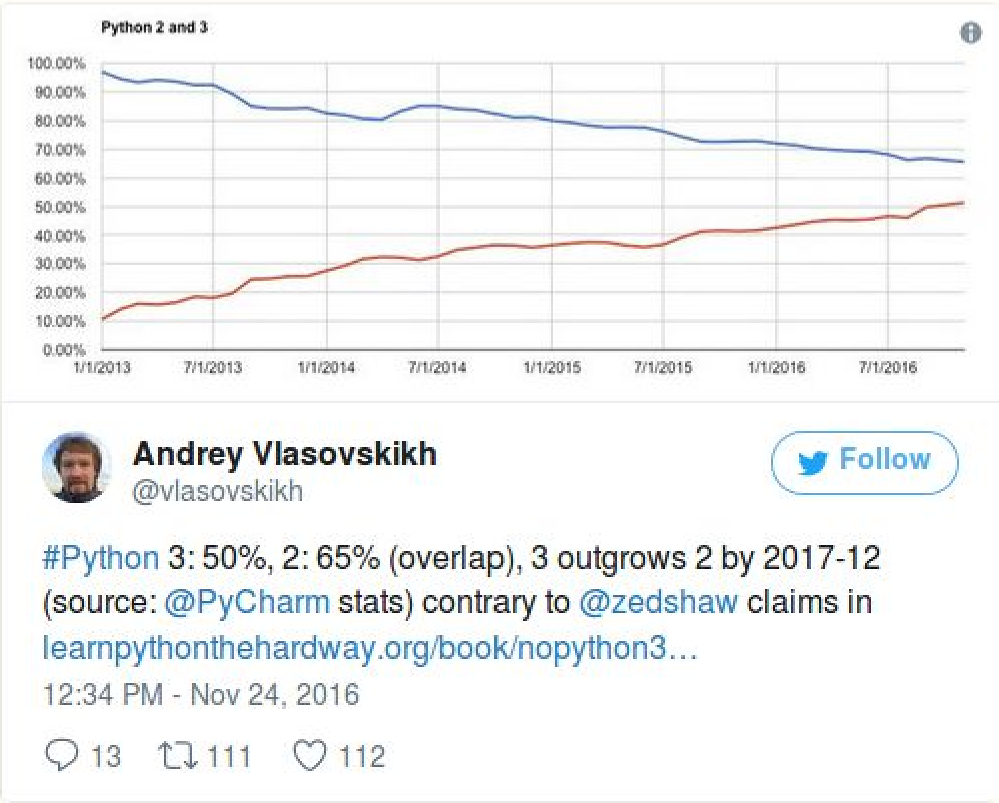


Author: Randy Olson (@randal_olson / randalolson.com)
Source: https://bigquery.cloud.google.com/table/the-psf:pypi.downloads20160903



Hynek Schlawack 🇿🇦
@hynek

I found some Python 2 code in Zimbabwe!



Andrey Vlasovskikh
@vlasovskikh

#Python 3: 50%, 2: 65% (overlap), 3 outgrows 2 by 2017-12 (source: @PyCharm stats) contrary to @zedshaw claims in learnpythonthehardway.org/book/nopython3…

12:34 PM - Nov 24, 2016

13 111 112

The Dilemma

Assuming that all your existing python 2 users will Instantly switch to python 3 is unrealistic.

In view of this

Support both python 2 and 3 for legacy systems and libraries.

To the Rescue

- Python-future
    - `pip install future`
- Six
    - `pip install six`

Print

Print

Python 2:

```
Import sys

print >> sys.stderr, 'echo Lima golf'
print 'say again'
print 'I say again', 'echo Lima golf'
print 'Roger',
```

Python 3:

```
import sys

print ('echo lima golf', file=sys.stderr)
print ('say again')
print ('I say again', 'echo lima golf')
print ( 'Roger', end='')
```

Use __future__ import

```
from __future__ import print_function
import sys


print ('echo lima golf', file=sys.stderr)
print ('say again')
print ('I say again', 'echo lima golf')
print ( 'Roger', end='')
```

Use six.print_

```
import six
import sys


six.print_('echo lima golf', file=sys.stderr)
six.print_('say again')
six.print_('I say again', 'echo lima golf')
six.print_('Roger', file=sys.stdout, end='')
```

The __future__ import is special and must be imported before anything else in the module/file.

Numbers

Numbers : Integer Inspection

Python 2:

```
y = 3

if isinstance (y, long):
        print (“y is a long Integer”)
else:
        print (“y is not a long integer”)
```

Python 3:

```
y = 3
if isinstance (y, int):
        print (“y is an Integer”)
else:
        print (“y is not an integer”)
```

Use int from future's builtins module

```
from builtins import int

y = 3

if isinstance (y, int):
        print (“y is an Integer”)
else:
        print (“y is not an integer”)
```

Six : integer_types constant

```
import six

y = 3

if isinstance(y, six.integer_types):
        print (“y is an Integer”)
else:
        print (“y is not an integer”)
```

Numbers :True Division

Python 2:

```
x, y = 5.0, 2
result = x / y
assert result == 2.5
```

Python 3:

```
x, y = 5, 2
result = x / y
assert result == 2.5
```

__future__ : division

```
from __future__ import division

x, y = 5, 2
result = x / y
assert result == 2.5
```

Exceptions

Raising Exceptions

Python 2:

```
def func(value):
        traceback = sys.exc_info()[2]
        raise ValueError, “funny value”, traceback
```

Python 3:

```
def func(value):
        traceback = sys.exc_info()[2]
        raise ValueError (“funny value”).with_traceback(traceback)
```

Python-future : raise_

```
from future.utils import raise_


def func(value):
    traceback = sys.exc_info()[2]
    raise_ (ValueError, “funny value”, traceback)
```

Six : raise_

```
from six import raise_
```

```
def func(value):
    traceback = sys.exc_info()[2]
    raise_ (ValueError, “funny value”, traceback)
```

Catching Exceptions

Python 2:

```
(x,y) = (5,0)

try:
        z = x/y
except ZeroDivisionError, e:
        print e
```

Python 3:

```
(x,y) = (5,0)
try:
z = x/y
except ZeroDivisionError as e:
z = e
print z
```

For compatibility

Use the as keyword instead of a comma.

```
(x,y) = (5,0)

try:
        z = x/y

except ZeroDivisionError as e:
        z = e
print z
```

Package Imports

Renamed Modules : use optional imports

```
try:
    from http.client import responses
except ImportError:
    from httplib import responses
```

Relative Imports

Python 2:

```
import constants
from cop import SomeCop
```

Python 3

```
from . import constants
from . cop import SomeCop
```

For compatibility : turn off implicit relative imports

```
from __future__ import absolute_import

from . import constants
from . cop import SomeCop
```

Setting Metaclasses

Setting Metaclasses

Python 2:

```
class MyBase (object):
        pass

class MyMeta (type):
        pass

class MyClass (MyBase):
        __metaclass__ = MyMeta
        pass
```

Setting Metaclasses …

Python 3

```
class MyBase (object):
        pass

class MyMeta (type):
        pass

class MyClass (MyBase, metaclass=MyMeta):
        pass
```

Python-future : with_metaclass

```
from future.utils import with_metaclass

class MyBase (object):
	pass

class MyMeta (type):
	pass

class MyClass (with_metaclass(MyMeta), MyBase)):
	pass
```

Six : with_metaclass

```
from six import with_metaclass

class MyMeta(type):
pass

class MyBase(object):
pass

class MyClass(with_metaclass(MyMeta, MyBase)):
pass
```

Six : @add_metaclass()

```
import six

class MyMeta(type):
    pass

@add_metaclass(MyMeta)
class Klass(object):
    pass
```

Strings and Bytes

Strings and bytes

Python 2:

```
Name = ’Captain’
```

Python 3:

```
Name = u’Captain'
Name = b’Captain'
```

Use the prefixes for compatibility

```
Name = u’Captain'
Name = b’Captain'
```

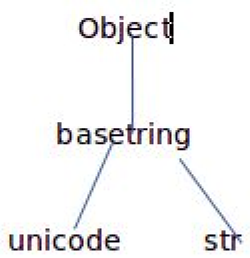
Six : u and b

```
import six

Name = six.u(’Captain')
Name = six.b(’Captain')
```

BaseString

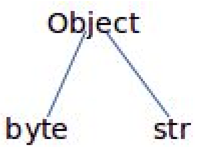
Python 2



```
string1 = “echo”
string2 = u “lima”
isinstance(string1, str)       #True
isinstance(string2, str)       #False
isinstance(string1, unicode)   #False
isinstance(string2, unicode)   #True
isinstance(string1, basestring)#True
isinstance(string2, basestring)#True
```

BaseString ..

Python 3



python-future

```
from past.builtins import basestring
```

```
string1 = “echo”
string2 = u “lima”
isinstance(string1, basestring)#True
isinstance(string2, basestring)#True
```

six

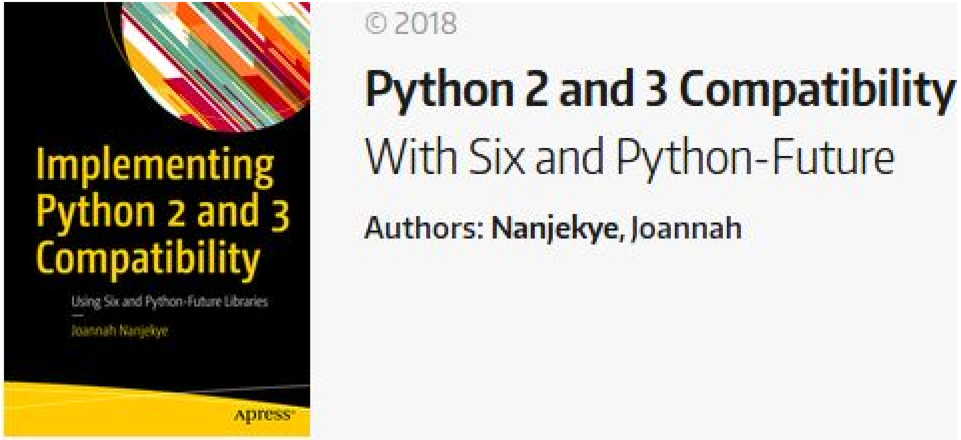
```
import six

string1 = “echo”
string2 = u “lima”
isinstance(string1, six.string_types)
isinstance(string2, six.string_bytes)
```

Conclusion

- Python 3 was not a mistake. New projects should use as default.
- Python 2 is still in use and will still be even after 2020.
- Libraries should be hybrid to wider reach.

The book



Implementing Python 2 and 3 Compatibility

Using Six and Python-Future Libraries

Joannah Nanjekye

Apress

© 2018

Python 2 and 3 Compatibility

With Six and Python-Future

Authors: **Nanjekye, Joannah**

Thank You

Resources

Ed Schofield , Cheat Sheet: Writing Python 2-3 compatible code

Benjamin Peterson, Six: Python 2 and 3 Compatibility Library

http://www.randalolson.com/2016/09/03/python-2-7-still-reigns-supreme-in-pip-installs/