To Py2 or to Py3 - That is the Question!

by
Kaustubh Vaghmare

(c) Copyright, KVaghmare

if jan2014:

```
allPackagesAvailable = False
performanceAvailablePackages < ideal
usePython2()
```

if jan2017:

```
startTalk()
```

(c) Copyright, KVaghmare

The Plan of the Talk

- A 4 Part

Part I
Why make Python 3?

(c) Copyright, KVaghmare

Why a new version for any S/W?

- Fix bugs.
- Fix issues.
- Add new features.

Python 3 was initially all about cleaning up the language and taking it closer to its Zen!

The Zen of Python

```
>>> import this
```

(c) Copyright, KVaghmare

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Few Examples of Bad Stuff in Python 2!

(c) Copyright, KVaghmare

print

```
>>> for i in range(3):
        print i
0
1
2


>>> for i in range(3):
        print i,

0 1 2
```

Implicit, not explicit!
Not readable!
Not obvious!

The /:

```
>>> a = 5
>>> b = 2
>>> a/b
2
```

Behavior of a statically typed language in a dynamically typed language!

(c) Copyright, KVaghmare

Silly Comparisons

```
>>> aList = []
>>> b = 3
>>> aList > b
True
>>> c = 5498709375
>>> aList > c
True
```

(c) Copyright, KVaghmare

Too many ranges!

```
   # I need to execute a loop 10,00,000 times.
>>> for i in range(1000000):
        ...
OR
>>> for i in xrange(1000000):
        ...
```

range = Makes a list in the memory

xrange = Generates ‘i’ values on demand.

(c) Copyright, KVaghmare

And much more...

- Two types of classes.
- Two types of comparison overload operators
- Confusing and dangerous raw_input and input functions
- l.sort() but sorted(l)!!
- Confusing package imports!

(c) Copyright, KVaghmare

And a major one – str

```
>>> a = ‘Hello’
>>> type(a)
str
# This is an 8-bit string.
>>> ... some raw binary data stream code ...
>>> ... captured in a reference ‘b’
>>> type(b)
str
```

Raw binary streams and human readable text – one and the same!

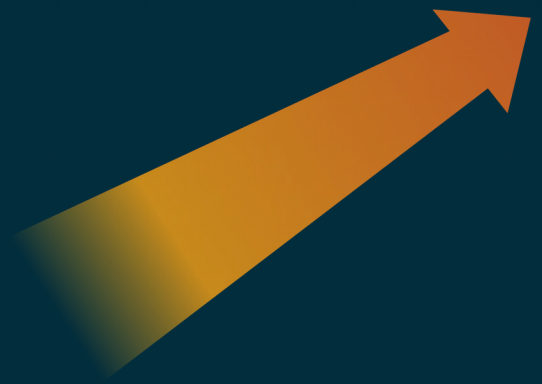
(c) Copyright, KVaghmare

So, let’s fix this situation!

(c) Copyright, KVaghmare

Two ways!

- Deprecation Approach
  - Recommend one coherent approach.
  - Issue warnings everytime user uses older approach.
- Fork it to something new!
  - Introduce a coherent new approach.
  - Break backward compatibility.



(c) Copyright, KVaghmare

So if Python 3 was to be a better, cleaner and a more consistent language – why is Python 2 still used?

(c) Copyright, KVaghmare

Reason - 1

- Python’s real strength
  - 3rd party libraries
- What Python 3 offered?
  - Syntactical coherence - Yes
  - Performce – No
- No real incentive or hurry to migrate.

(c) Copyright, KVaghmare

Reason - 2

- Very kind core Python team!
  - Long term support and bug fixes to Python 2.

(c) Copyright, KVaghmare

Reason - 3

- Backports!

```
>>> from __future__ import XYZ
```

(c) Copyright, KVaghmare

Reason - 4

- Python 3.0 was released with haste
  - Average 30% slower.
  - I/O could be > 10 times slower!

(c) Copyright, KVaghmare

Part 2
So what's changed, doc?

https://pythonclock.org/

(c) Copyright, KVaghmare

Astropy

- V2.0 LTS
  - Last Python 2 release!
- V3.0
  - Will work only on Python 3

Jupyter

- V5.0 (current)
  - Last version for Python 2.

JAN 03, 2013

Will Scientists Ever Move to Python 3?

March 2016: Please note the date on this post. Given the developments in the last three years, I would no longer agree with much of what I've written here. In particular, I substantially underestimated the ability of tools like six and python-future to enable single-codebase Python 2/3 support, and virtually all scientific packages now use such tools to support both. Short version: just use Python 3. There's almost no reason not to any more.

(c) Copyright, KVaghmare

Part III
So, what's new in Python 3?
(Differences between Python 2 and Python 3)

(c) Copyright, KVaghmare

‘print’ is a function

```
>>> for i in range(3):
        print(i, end=’ ‘)
0 1 2
>>> print(‘Hello’,’World’, sep=’|‘)
Hello|World

>>> f = open(‘newFile’, ‘w’)
>>> print(‘Something’, file=f)
```

(c) Copyright, KVaghmare

/ and //

```
>>> a = 5
>>> b = 2
>>> a/b
2.5
>>> a//b
2
```

(c) Copyright, KVaghmare

One Consistent Integer

```
>>> a = 23
>>> a**20
1716155831334586342923895201

# In Python 2, you would get
# 1716155831334586342923895201L
```

One single range

```
>>> for i in range(1000000):
        ...
```

Will never create a huge list.

i.e. range() is a ‘generator’ function.

If you need a list, you need to say,

```
>>> aList = list(range(1000000))
```

Same is true for map() and zip()

(c) Copyright, KVaghmare

Strings!

```
>>> a = ‘Hello’
>>> type(a)
str
>>> b = u’World’
>>> type(b)
str
```

All strings are Unicode strings.

```
>>> rawStream = b’somechars’
>>> type(rawStream)
bytes
>>> rawStream + ‘more’
TypeError
```

Classes – in Python 2...

```
>>> class A:
        pass
>>> type(A)
<type 'classobj'>
```

```
>>> class A(object):
        pass
>>> type(A)
<type ‘type’>
```

```
>>> class A(list):
        pass
>>>> type(A)
<type ‘type’>
```

(c) Copyright, KVaghmare

Classes – in Python 3 ...

All the same!

Exception Raising

```
try:
    ...
except OSError as V:
    If V.errno == 2:
        ...
```

```
try:
    ...
except FileNotFound:
    ...
```

There are many more error classes in Python 3 which reduce error catching code.

```
try:
    ...
except Something, V:
    ...
```

```
try:
    ...
except Something as V:
    ...
```

Just one input()

```
>>> a = input(‘something’)
>>> type(a)
str
```

Python 2’s distinct raw_input() and input() are gone!

(c) Copyright, KVaghmare

No more weird comparisons!

```
>>> [] > 45435435
TypeError
```

(c) Copyright, KVaghmare

Now, some new stuff you can only do in Python 3!

(c) Copyright, KVaghmare

Tuple Unpacking

```
>>> a, b, *theRest = range(10)
>>> a
0
>>> b
1
>>> theRest
[2, 3, 4, 5, 6, 7, 8, 9]
```

(c) Copyright, KVaghmare

Tuple Unpacking (contd)

```
>>> a, *theMiddle, b = range(10)
>>> a
0
>>> theMiddle
[1, 2, 3, 4, 5, 6, 7, 8]
>>> b
9
```

(c) Copyright, KVaghmare

Tuple Unpacking in Function Defs

```
>>> def sum(a, b, danger=False):
        if danger:
            shutil.rmtree(‘/’)
>>> sum(2, 3)
# FINE!
>>> sum(2, 3, 4)
# BAD! danger = 4 is Boolean True
>>> def sum(a, b, *, danger=False):
# Now, there is only one way to set danger to True
>>> sum(2, 3, danger=True)
```

(c) Copyright, KVaghmare

Function Annotations

```
def histogram(x: ‘Samples’, n: ‘Number of Bins’) -> ‘Binned Sample’:
    ...
```

There is no direct application. But automated code documentation tools can benefit from such annotations. Even otherwise, it makes your function code easier to understand.

(c) Copyright, KVaghmare

And Much More ...

- @ operator
  - Only Python 3.5 onwards
  - Multiplies 2 matrices
- ‘raise from’
  - Better exception chaining
- F-strings (Python 3.6 & later)

  eg. a = f“hello {x}”
- ‘yield from’

(c) Copyright, KVaghmare

- A better standard library
  - More modules
  - Duplicates merged
- Better asynchronous I/O support
  - @coroutine feature
  - Direct async syntax
- Clearer package import syntax.

(c) Copyright, KVaghmare

- Variable names can be Unicode based (to an extent)
- Pathlib objects
  - fileLocation / ‘subFolder’ / ‘new.txt’
  - No need to os.path.join()
- And expanding.

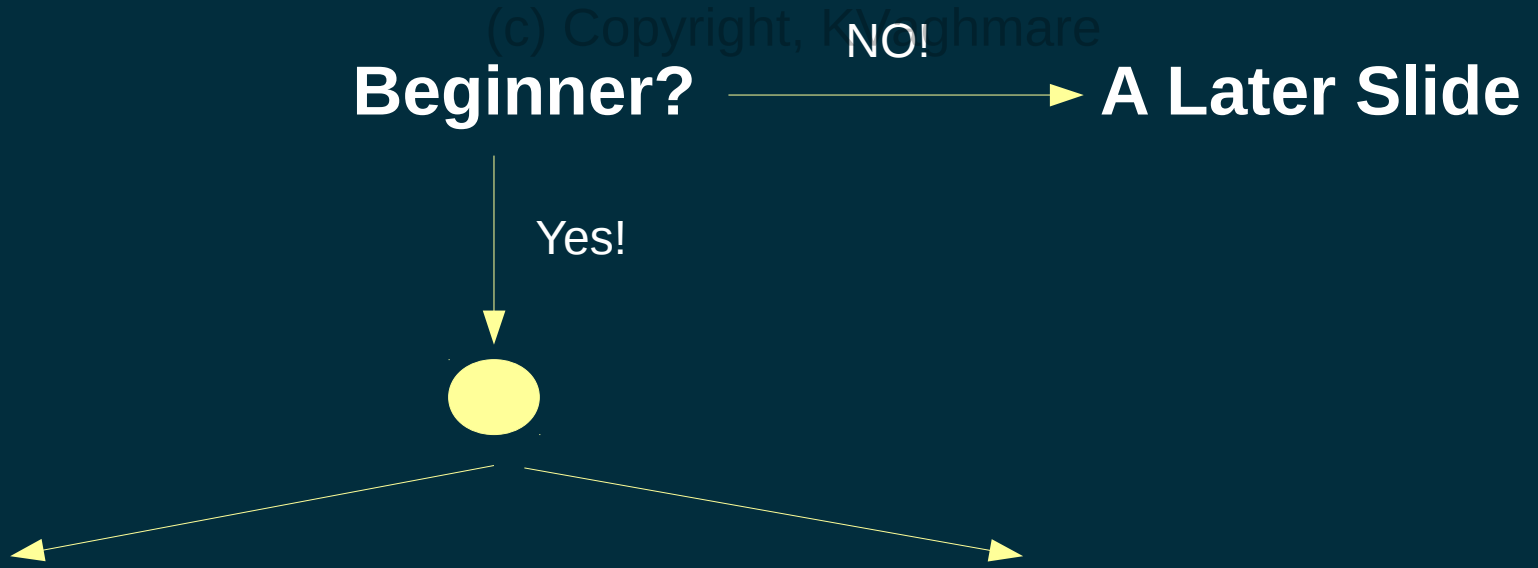
(c) Copyright, KVaghmare

Part IV
Decision Making & Migration Strategies

(c) Copyright, KVaghmare

What version should I use?

(c) Copyright, KVaghmare



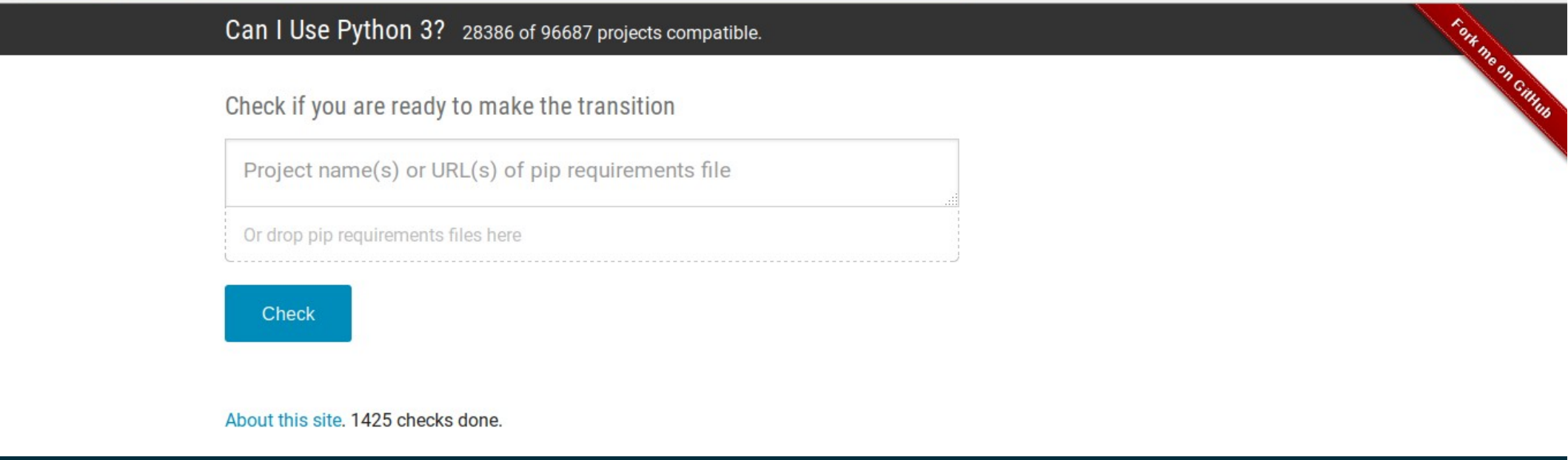
Use a lot of personal code from collaborators?

- Written in Python 2.
- Collaborators won't move to Python 3.
- Then stick to Python 2 BUT
- Code in as version neutral a manner as possible to make future transition easier.

Will code afresh using third party libraries!

- Python 3 may be the best choice for you.
- If you can assemble a list of of packages you'll need, you can also check their Python 3 compliance from: https://caniusepython3.com/

(c) Copyright, KVaghmare



(c) Copyright, KVaghmare

A Caveat...

File	Type	Py Version
esutil-0.6.2rc3.tar.gz (md5)	Source	

Author: Erin Scott Sheldon
Home Page: http://code.google.com/p/esutil/
License: GPL
Categories

- Development Status :: 5 - Production/Stable
- Intended Audience :: Science/Research
- License :: OSI Approved :: GNU General Public License (GPL)
- Programming Language :: Python
- Programming Language :: Python :: 2
- Programming Language :: Python :: 2.6
- Programming Language :: Python :: 2.7
- Programming Language :: Python :: 3
- Programming Language :: Python :: 3.3
- Programming Language :: Python :: 3.4
- Programming Language :: Python :: 3.5
- Programming Language :: Python :: 3.6
- Topic :: Scientific/Engineering :: Astronomy

Package Index Owner: esheldon
DOAP record: esutil-0.6.2rc2.xml

(c) Copyright, KVaghmare

Not a Beginner?

- Coding for yourself?
- Don’t give a [whatever] about Python 3’s motives?
- Not making a worthwhile contribution with your code anyway?
- Will never move to Python 3?

Suit yourself!

(c) Copyright, KVaghmare

Not a Beginner?

- And want to remain updated?
- Most of the ‘Beginner’ decision flow applies.

(c) Copyright, KVaghmare

Parallel Support for 2 and 3

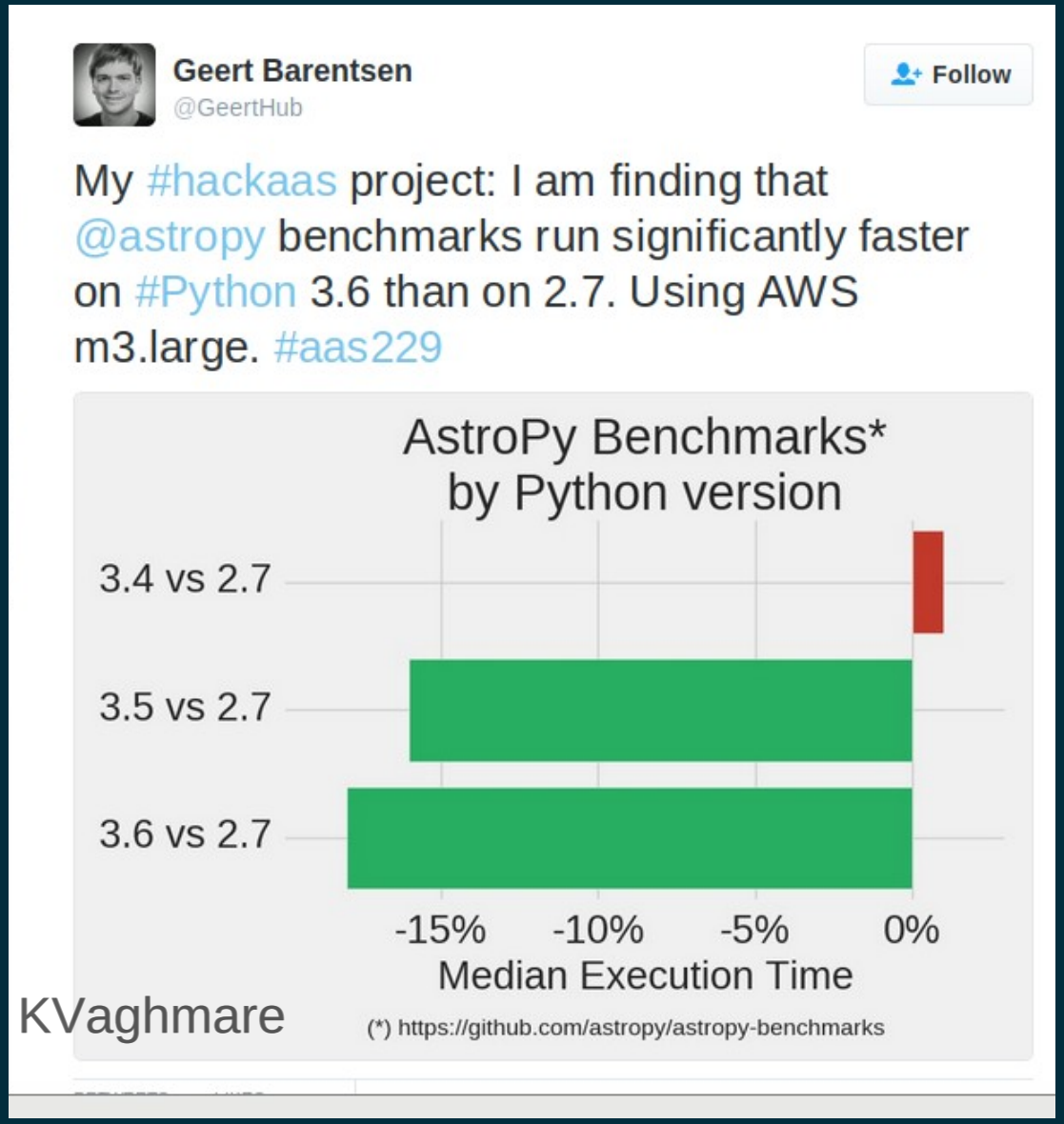
- You need ‘six’ (2*3)
- You also need to understand Futurize/Modernize which help write setup scripts in a parallel manner.

(c) Copyright, KVaghmare

Support only Python 3.

- Make sure your code is atleast Python 2.7 compatible.
- Ensure you have your tests in place.
- Run 2to3 on your codes one by one and review recommended changes.
- When confident ‘2to3 -w’ to replace existing files (backup files are created for ‘diff’ing)
- Run your tests.
- Live happily ever after.

And from a recent mail...



(c) Copyright, KVaghmare

Conclusions

- Python 3 is better.
  - Syntactically
  - Feature wise
  - And now even performance wise.
- Only motive for Python 2 is a (rare) dependency issue (and personal inertia).
- Python 2 support is dropping.
- A good time to start jumping ship.
- I am going to!

(c) Copyright, KVaghmare

```
...
...
except EOFError:
    sayThankYou()
    sys.exit(0)
```

(c) Copyright, KVaghmare