学习Python 2还是Python 3？

罗振宇在今年的跨年演讲，《时间的朋友》中有个观点，大意是说，人们都有一种受虐情节，有时候希望别人对他粗暴一点。为此，他还举了两个例子，分别是“乔布斯对待消费者的态度”和“和菜头不尊重他的饮食需求”，末了还很享受的来一句：我爱死他了，对我再粗暴一点好不好！

看到很多新同学在学习Python的过程中，犹豫学习Python 2还是学习Python 3而迟迟不行动，白白地浪费了大把时间，错过了升职加薪的机会，我真心觉得非常遗憾。所以，我忍不住想对大家粗暴一次，给大家一个粗暴而又正确的答案：

> 应该学习Python 2还是Python 3？都要学！

这个答案可能很出乎意料，也很容易反驳，例如：

- Python 3 才是Python的未来
- Python 官方都建议指直接学习Python 3
- Python 2 只维护到2020年

真的是这样吗？作为一个在一线互联网公司奋斗的工程师，也是一个多年的Python老手，大家不妨来看看我这么说的理由。

为什么还要学习Python 2

Python 2只维护到2020年不应该成为你拒绝Python 2的理由

所有纠结学习Python 2还是Python 3的朋友都知道，按照Python官方的计划，Python 2只支持到2020年。可能大家接触Python的时间还不长，不知道Python官方曾经还说过，Python 2只支持到2015年(https://github.com/python/peps/blob/master/pep-0373.txt)。所以，大家可以看到，跳票不是中国人的特权，Python官方也是会跳票的。

如果大家关注科技新闻的话，会注意到，就在前几天，微软刚宣布将在2020年对Win 7停止任何技术支持，之后即使遇到BUG和安全问题，他们也不会再修复，就像现在的XP一样。但是，大家看看我们周围的同事、朋友、亲戚，到底是用Win 7的多还是用Win 10的多？这些用Win 7的人有吵着说我要升级Windows的吗？用Win 10的人有吵着让用Win

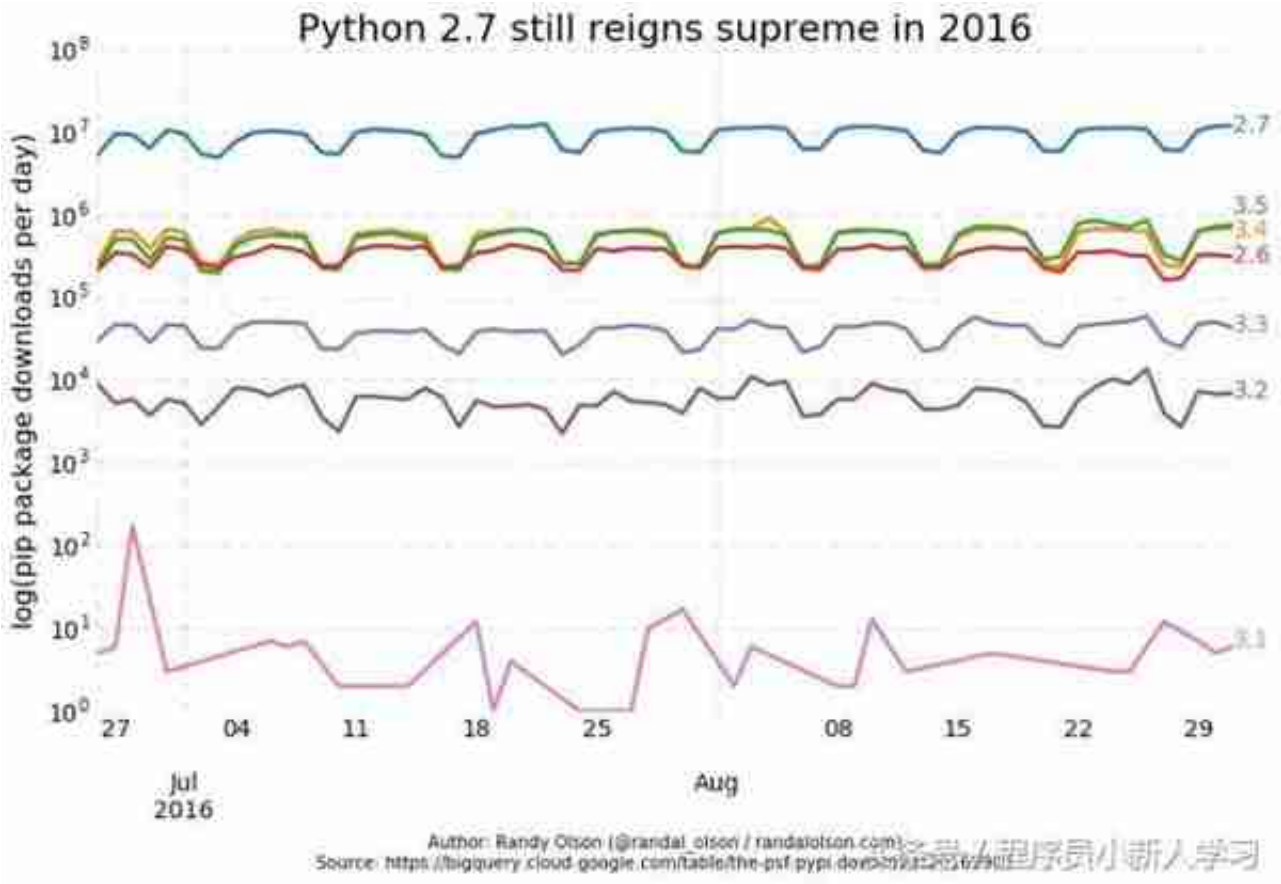
7的人升级吗？

但是，在Python这个圈子，就是有很多人吵着要让别人升级Python 3。很多时候用户并不关心自己用的是Python 2还是Python 3，只要能用就行。所以，用Python 2的人并没有什么动力去升级到Python 3。

如果你觉得，Python 3才是Python的未来，不希望接触Python 2的项目。那么，问题来了，女神跟你说，晚上来我家给我修下电脑呗，但是我的电脑是Windows XP的，你是去还是不去？

Python官方建议学习Python 3只是一种一厢情愿的行为

我们来看一下Python 2和Python 3的下载统计数据(Python 2.7 still reigns supreme in pip installs)：



Python 2的使用量远远超过Python 3。而且，大家注意竖轴的单位，是指数！简单换算一下就知道，仅从下载量来说，Python 2.7的下载量是总下载量的90%！所以，学习Python，想直接抛弃Python 2学习Python 3，几乎是不可能的事情。

上面的数据是全球范围的统计数据，我们来看看中国互联网的情况。为了写这篇文章，我专门在同学群里面发了红包，邀请了来自百度、阿里、腾讯、网易、美团、华为、招行、建行、eBay、美图、Oracle等公司的一线技术专家，统计了各大公司使用Python的情况。

统计数据如下：

- 10% 使用 Python 3
- 20% 既使用Python 2也使用Python 3，Python 2用的更多
- 70% 使用Python 2

统计数据基本与pypi的全球范围的统计数据吻合。所以，如果你说，我一开始学的就是Python 3，Python 3也是Python的未来，我不想去了解和学习Python 2。那么，你可能要和大半个中国互联网失之交臂了。或许你也不在乎，但是，如果有人拿钱砸你让你维护Python 2的代码呢？

Python 2还会存在很长一段时间

不知道大家有没有想过，为什么Python官方极力让大家使用Python 3，而Python 2依然处于统治地位呢？

其实答案很简单也很粗暴：因为绝大多数人，你给他什么，他就用什么。据我说知，尽管Python 3在2008年12月就已经发布了，但是，目前Python 3仍然不是任何操作系统的默认Python解释器，这是Python 3使用不广泛的主要原因。

我们都知道，在任何一家公司，升级服务器的操作系统版本都是一个很慎重事情。所以，我们有理由相信，Python 2还会存在很长一段时间。很长是多长呢？至少比2020年还要长。

这个世界并不是非黑即白的，Python也不是

有了前面的数据做支撑，我们不是应该学习Python 2吗，为什么Python 2和Python 3都要学呢？

首先，这个世界并不是非黑即白的，Python也不是。在学习Python 2和学习Python 3中间，其实有一个很好的平衡，那就是同时兼容Python 2和Python 3。为了做到同时兼容Python 2和Python 3，需要深用到Python的__future__库。__future__库里面包含了不少从Python 3

backport 到Python 2的特性，充分使用__future__库，可以很好的兼容Python 2和Python 3。

其次，Python 2和Python 3确实有一些差异，但是，并没有大家想象的那么大，Python 2和Python 3之间的差异不到Python语法的10%，我们可以快速地了解哪些Python 2里面的语法在Python 3中已经被弃用，在我们写代码的过程中，规避掉这一部分语法即可。在Python的最佳实践中，Python 3里弃用的Python语法，在Python 2里面也不推荐使用，不然也不会被弃用了。如果你知道并坚持Python的最佳实践，那么，对你来说，Python 2和Python 3的差异就更小了。

最后，我们可以参考优秀的开源软件的做法，如OpenStack，努力做到代码同时兼容Python 2和Python 3(Python3 - OpenStack)，也可以借助一些开软的库（如six）来同时兼容。如果能够做到同时兼容Python 2和Python 3，我们的使用者将更广泛，我们的代码也将更有价值。

拿Windows 来说，当 Windows 7 发布的时候（我就不说 Vista 了），很多人依然抱着 XP 不放，当你问他们为什么时，他们会一本正经地说，“新系统好卡啊”，或是“好多软件都不兼容啊”，或是“改变很大啊，好不习惯啊”，甚至是“ XP 已经是很好的系统了，微软出个新系统就是为了坑钱”。

于是乎，春去春又来，送走了 Win 7 ，我们又迎来了 Win 8 ，但是这些人的想法依然没有改变（我相信中国人中这种情况多一些）。如果这种人很多而且这种情况持续下去的话，最终的结局只会是微软的状况越来越差，最终人们毫无选择，投降了Linux的怀抱（咦？怎么有种心花怒放的感觉）。

当我在脑子里把上面的 Win XP 换成 Python 2 、Win 7 换成 Python 3 甚至 Python 4 时，不禁感到一阵恐惧，我差点就和其他人合谋把 Python 给害死！试想一下，多年以后，Ruby 、Go 等语言都有了很多新的特性，虽然最新的 Python 也十分优秀，但因为一些人，不愿改变，坚守着老版本，抛出一些可笑的理由，最终 Python 因为用户习惯而没落了， Guido 和整个 Python 社区的努力都被这些人的习惯给无视了。

让我们来看看这些可笑的理由（关于详细的解释，可以看一下知乎上的徐酿泉的答案，我在这简单总结一下）：

什么？支持 Python 3 的库太少？醒醒吧，这都6年了，最新都3.4.1了，现在还不支持 Python 3

的库大多是常年无人维护的东西了。

什么？新版本和旧版本兼容性差？放心吧，以后的版本会越来越不兼容，除非你打算死守 Python 2 一辈子。况且，为了新的特性，改变一下有那么难吗？

最后，那些还在坚守旧版本的人，你们的一堆理由和批评，真的不是在为自己的问题作辩护吗？

stop talking, just do it

前面说了我对学习Python 2还是Python 3的一些观点，希望能够帮助大家少走弯路，另外，关于Python的版本问题，我这里还有一些良心建议：

- 学习Python前，先了解在Python 3里面已经弃用的Python 2语法，对这些部分简单带过不要花太多时间
- 使用Python 2，不要使用Python 2.7以前的版本
- 使用Python 3，不要使用Python 3.4以前的版本
- 多了解Python 2的__future__库
- 对同一份代码，不要为Python 2和Python 3分别维护分支，努力在一套代码中兼容Python 2和Python 3

这篇文章详细的说明了为什么要同时学习Python 2和Python 3，如何在Python 2和Python 3中找到一个平衡。但是，重要的不是纠结学习Python 3还是Python 2，而是“stop talking, just do it！”。

Python2与Python3的具体区别

除了引入import from future，了解一下两者的区别也是很必要的

print函数:（Python3中print为一个函数，必须用括号括起来；Python2中print为class）

Python 2 的 print 声明已经被 print()
函数取代了，这意味着我们必须包装我们想打印在小括号中的对象。

Python 2

1

2

3

4

print 'Python', python_version()

print 'Hello, World!'

print('Hello, World!')

print "text", ; print 'print more text on the same line'

run result:

Python 2.7.6

Hello, World!

Hello, World!

text print more text on the same line

Python 3

1

2

3

4

print('Python', python_version())

print('Hello, World!')

print("some text,", end="")

print(' print more text on the same line')

run result:

Python 3.4.1

Hello, World!

some text, print more text on the same line

通过input()解析用户的输入：（Python3中input得到的为str；Python2的input的到的为int型，Python2的raw_input得到的为str类型）统一一下：Python3中用input，Python2中用row_input，都输入为str

幸运的是，在 Python 3 中已经解决了把用户的输入存储为一个 str
对象的问题。为了避免在 Python 2
中的读取非字符串类型的危险行为，我们不得不使用 raw_input() 代替。

Python 2

Python 2.7.6

[GCC 4.0.1 (Apple Inc. build 5493)] on darwin

Type “help”, “copyright”, “credits” or “license” for more information.

```
>>> my_input = input('enter a number: ')enter a number: 123
>>> type(my_input)<type 'int'> >>> my_input = raw_input('ent
er a number: ')enter a number: 123 >>> type(my_input)<type '
str'>
```

Python 3

Python 3.4.1

[GCC 4.2.1 (Apple Inc. build 5577)] on darwin

Type “help”, “copyright”, “credits” or “license” for more information.

```
>>> my_input = input('enter a number: ')enter a number: 123
>>> type(my_input)<class 'str'>
```

整除:(没有太大影响)（Python3中/表示真除，%表示取余，//表示地板除（结果取整）；Python2中/表示根据除数被除数小数点位得到结果，//同样表示地板除）统一一下：Python3中/表示真除，%表示取余，//结果取整；Python2中带上小数点/表示真除，%表示取余，//结果取整

Python 2

1

2

3

4

5

print 'Python', python_version()

print '3 / 2 =', 3 / 2

print '3 // 2 =', 3 // 2

print '3 / 2.0 =', 3 / 2.0

print '3 // 2.0 =', 3 // 2.0

run result:

Python 2.7.6

3 / 2 = 1

3 // 2 = 1

3 / 2.0 = 1.5

3 // 2.0 = 1.0

Python 3

1

2

3

4

5

print('Python', python_version())

print('3 / 2 =', 3 / 2)

print('3 // 2 =', 3 // 2)

print('3 / 2.0 =', 3 / 2.0)

print('3 // 2.0 =', 3 // 2.0)

run result:

Python 3.4.1

3 / 2 = 1.5

3 // 2 = 1

3 / 2.0 = 1.5

3 // 2.0 = 1.0

xrange模块：

在 Python 3 中，range() 是像 xrange() 那样实现以至于一个专门的 xrange() 函数都不再存在（在 Python 3 中xrange() 会抛出命名异常）。

在 Python 2 中 xrange() 创建迭代对象的用法是非常流行的。比如： for 循环或者是列表/集合/字典推导式。

这个表现十分像生成器（比如。“惰性求值”）。但是这个 xrange-iterable 是无穷的，意味着你可以无限遍历。

由于它的惰性求值，如果你不得仅仅不遍历它一次，xrange() 函数 比 range() 更快（比如 for 循环）。尽管如此，对比迭代一次，不建议你重复迭代多次，因为生成器每次都从头开始。

python 2.4 与 python 3.0 的比较

一、 print 从语句变为函数

原: print 1, 2+3

改为: print (1, 2+3)

二、range 与 xrange

原 : range(0, 4) 结果 是 列表 [0,1,2,3]

改为：list(range(0,4))

原 : xrange(0, 4) 适用于 for 循环的变量控制

改为：range(0,4)

三、字符串

原: 字符串以 8-bit 字符串存储

改为: 字符串以 16-bit Unicode 字符串存储

四、try except 语句的变化

原: try:

.....

except Exception, e :

.....

改为

try:

.....

except Exception as e :

.....

五、打开文件

原： file(.....)

或 open(.....)

改为：

只能用 open(.....)

六、从键盘录入一个字符串

原: raw_input("提示信息")

改为: input("提示信息")

七、bytes 数据类型

A bytes object is an immutable array. The items are 8-bit bytes, represented by integers in the range 0 <= x < 256.

bytes 可以看成是“字节数组”对象，每个元素是 8-bit 的字节，取值范围 0~255。

由于在 python 3.0中字符串以 unicode 编码存储，当写入二进制文件时，字符串无法直接写入（或读取），必须以某种方式的编码为字节序列后，方可写入。

（一）字符串编码(encode) 为 bytes

例: s = "张三abc12"

b = s.encode(编码方式)

b 就是 bytes 类型的数据

常用的编码方式为 ： "uft-16" , "utf-8", "gbk", "gb2312", "ascii" , "latin1" 等

注 ： 当字符串不能编码为指定的“编码方式”时，会引发异常

(二) bytes 解码(decode)为字符串

s = "张三abc12"

b = s.encode("gbk") # 字符串 s 编码为 gbk 格式的字节序列

s1 = b.decode("gbk") # 将字节序列 b以gbk格式 解码为字符串

说明，当字节序列不能以指定的编码格式解码时会引发异常

(三）使用方法举例

```
#coding=gbk

f = open("c:\\1234.txt", "wb")
```

```
s = "张三李四abcd1234"
# ------------------------------
# 在 python2.4 中我们可以这样写：
# f.write( s )
# 但在 python 3.0中会引发异常
# ------------------------------
b = s.encode("gbk")
f.write( b )
f.close()
input("?")
```

读取该文件的例子:

```
#coding=gbk
f = open("c:\\1234.txt", "rb")
f.seek(0,2) #定位至文件尾
n = f.tell() #读取文件的字节数
f.seek(0,0) #重新定位至文件开始处
b = f.read( n )
# ------------------------------
# 在 python 2.4 中 b 是字符串类型
# 要 python 3.0 中 b 是 bytes 类型
```

```
# 因此需要按指定的编码方式确码
# ----------------------------
s = b.decode("gbk")
print ( s )
# ----------------------------
# 在 python 2.4 中 可以写作 print s 或 print ( s )
# 要 python 3.0 中 必须写作 print ( s )
# ----------------------------
f.close()
input("?")
```

运行后应显示:

张三李四abcd1234

(四) bytes序列，一但形成，其内容是不可变的

例：

```
s="ABCD"
b=s.encode("gbk")
print b[0] # 显示 65
b[0] = 66
# 执行该句，出现异常: 'bytes' object does not support item assignment
```

八、 chr(K) 与 ord(c)

python 2.4.2以前

chr(K) 将编码K 转为字符，K的范围是 0 ~ 255

ord(c) 取单个字符的编码, 返回值的范围: 0 ~ 255

python 3.0

chr(K) 将编码K 转为字符，K的范围是 0 ~ 65535

ord(c) 取单个字符的编码, 返回值的范围: 0 ~ 65535

九、 除法运算符

python 2.4.2以前

10/3 结果为 3

python 3.0

10 / 3 结果为 3.3333333333333335

10 // 3 结果为 3

十、字节数组对象 --- 新增

(一） 初始化

a = bytearray(10)

a 是一个由十个字节组成的数组，其每个元素是一个字节，类型借用 int

此时，每个元素初始值为 0

（二） 字节数组 是可变的

a = bytearray(10)

a[0] = 25

可以用赋值语句更改其元素，但所赋的值必须在 0 ~ 255 之间

(三) 字节数组的切片仍是字节数组

(四) 字符串转化为字节数组

```
#coding=gbk
s ="你好"
b = s.encode( "gbk") # 先将字符串按某种“GBK”编码方式转化为 bytes
c = bytearray( b ) #再将 bytes 转化为 字节数组
```

也可以写作

```
c = bytearray( "你好", "gbk")
```

(五) 字节数组转化为字符串

```
c = bytearray( 4 )
c[0] = 65 ; c[1]=66; c[2]= 67; c[3]= 68
s = c.decode( "gbk" )
print ( s )
# 应显示： ABCD
```

(六) 字节数组可用于写入文本文件

```
#coding=gbk
f = open("c:\\1234.txt", "wb")
s = "张三李四abcd1234"
# ------------------------------
```

```
# 在 python2.4 中我们可以这样写：

# f.write( s )

# 但在 python 3.0中会引发异常

# ------------------------------

b = s.encode("gbk")

f.write( b )

c=bytearray( "王五","gbk")

f.write( c )

f.close()

input("?")
```