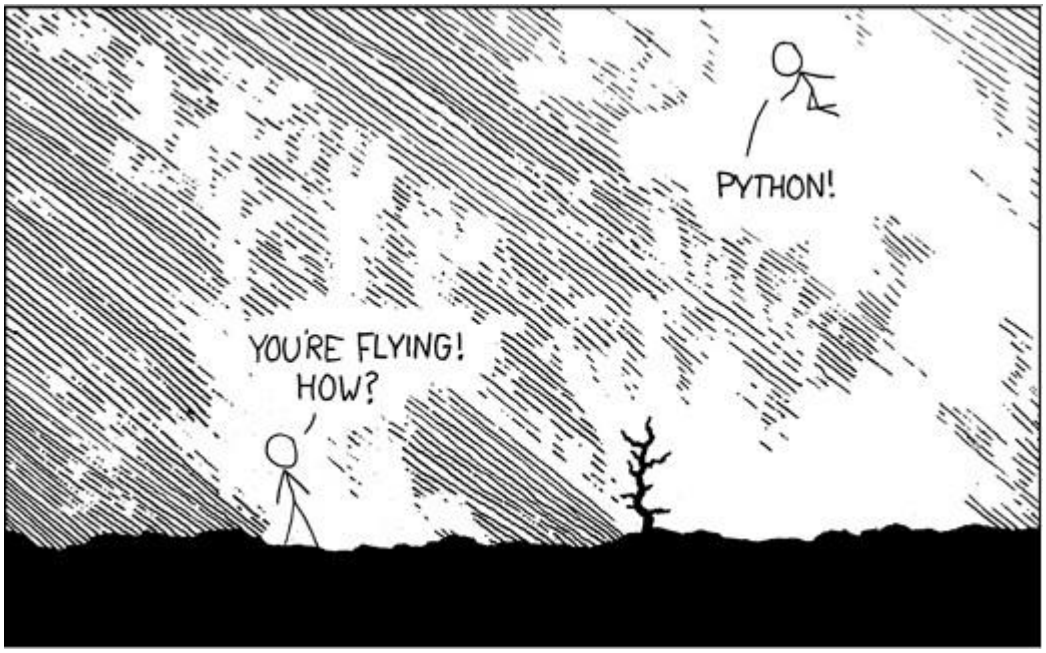
完全用 Python 工作---Harness the power of Python

作者： 石雨浓

第一天，太初有道，神谕，import light，于是便有光.



（Quick fact: 在 python 解释器里输入 import antigravity 有彩蛋）

作为一个业余物理工作者以及入门计算机使用者，选择一门称手的编程语言是非常重要的事. 从计算能带，处理数据，编写脚本到画图，写个 http 服务器分享文件（看上去很专业，实际在 Python 里只有一行），做个网页，几乎全部需要计算机完成. 但是为了这其中每个不同的目的单独去学一门语言成本简直过于高，于是需要一个一般用途（general-purpose）的语言,处理所有的事是非常自然的事情.

编程语言的两极是 Assembly 和 Haskell，一个接近硬件的本质，一个接近计算的本质. 一个是地狱，处理着最繁琐最耗神的事情：内存分配，系统调度，硬件架构，各种寄存器 A1, B2... 一个是天堂，优美的写着递归，高阶函数，lambda 表达式，优美的并行计算(完全不用考虑 race condition). 然而我们生活在人间，所以大规模应用的语言不可能如此纯粹. 两端中间游离着很多 general-purpose 的语言，C, C++, Java, Ruby，几乎都能达到我们所有日常的要求. 只不过，这些语言能做的 Python 都能做，而且 Python 做得更好。接下来我说明为什么。但是要说明本文的读者不包括写嵌入式，写 javascript 以及写大型项目对性能要求极高的人（即使是大型项目也可以80%用 python，20%用 C），当然还有就是 java 和 C++的重度患者。(完全使用 XX 工作意思不是"所有人都完全使用 XX 工作"！显然只是部分人。更多的是，非专业编程但是想提高效率的人。比如之前有篇<完全使用*nix 工作>，C#，ios 开发的人显然就一下也不能用。对于我，linux 再好我也只能装在老电脑上交交 CS225的作业。当我把 mint, opensuse, archlinux 装遍了，下一步就是 gentoo 了的时候，否决它只有一条理由，我笔记本电池不经用，而桌面 linux 的电源管理..... 感谢我的 cpu 风扇~!）

首先，我想说的是，为什么不用下面这些大部分人很熟悉的语言：

1. C: 你难道指针扎得不疼么？每天收垃圾很舒服？键盘上 P 右边两个键是不是已经按坏了？

2. C++: 学 C++三年以内请不要说你会 C++; 学了三年以上的人，恭喜你们，你过去几年浪费的时间我可以拿着香飘飘环绕地球一圈了.

3. Java: 不好意思，Java 的面向对象对我来说是原子弹打原子。 而且 Java7才引进 Lambda 表达式实在是太晚了，即使 java 以后会跟 python 越来越像，至于支持真正的函数式编程？我希望下个末日之前可以实现. 而且有时候我确实需要单行执行的解释器而 Java 并没有。

4. Ruby: Ruby 很好，但是你为什么不直接说你只是为了用 RoR?

5. Lisp: 如果你用 lisp，你平时肯定会用 python 或者 perl 写脚本。 而且你会 Lisp 不去拯救世界还来看这篇文章干什么?! 抽象语法树什么的最讨厌了....

6. Perl: 我第一次看 Perl 的代码就感觉像用脚写的. "为什么满屏的正则表达式？"!

7. C#, php, javascript：呵呵。

8. Shell: 这算语言么？

9. Matlab: 第一，我穷酸学生没钱每年买你的正版，看到激活码就想吐。 第二，我不想心血来潮画两个心形函数的时候用1mb 的窄带花两天下个5.03Gb 的文件在我128Gb 的固态硬盘里装，然后用完两个小时就删，如此循环。 第三，我会 python 了不想再花时间学你的 sb 语法，熟悉你的.m 文件。第四，所有对 windows 的垄断的血泪控诉都直接对 mathwork 转过来吧～什么对开源，对自由，对的打击信仰～绝对适用～ 第五，python 大部分时候如果不比你好用至少跟你一样好用，而这只是它不到10%的功能，几个程序员业余时间写出来的库。真心请 matlab 你这个没事发邮件“培训一个星期2000刀打折700刀”的大公司滚粗。

10. Haskell: 每次想静下心来学 haskell 都会情不自禁从范畴论看起....

对于单纯程序语言的使用者来说，用途（内在逻辑）大于一切不必要的语言细节。比如我就想建个数组放东西，为什么我要懂内存回收？！

所以在易用性方面，Python 相对于他们作了很大改进的部分。

好吧，你会说 Python 没有缺点么。确实有，而且很严重，那就是运行慢。而且是慢出风格，慢出自信。（Python 3比 Python 2 慢 15%以上， 这是一种什么风格！）相同的程序 Python 比 C 慢几百倍很正常。这让 Python 的发展受到很多限制。但是对于个人使用来说这个缺点完全不属于缺点。第一，这个年代谁没有奔腾酷睿2什么的。你手机的运行能力都可以几毫秒内把你在厕所拍的几千张自拍液化，磨皮，磨骨好几遍了。而且你觉得0.01秒和0.5秒的区别真的那么大么？12秒也不是很久啊。第二，很大程度上程序的慢更关乎于算法，比起 O（n）和 O（n^2)的区别， 语言间的差异就显得很小了，第三，请注意，如果你使用过 Python 而且真实的觉得 Python 慢，那么情看下这个列表：

1. Google 创立前的第一个网络爬虫。

2. Quora，美国最大在线知识问答平台，开复哥总是在上面拽文的。

3. Dropbox。

4. Youtube

5. BT。

6. 知乎，中国的 Quora。

7. 豆瓣，开创社交工具绿色系代表 yp 的先河。

你知道我要说什么了。.....恩～他们有一个共同点~ ---------- 都是 Python 写的！如果 tmd 的 Dropbox 没有觉得 Python 慢，请你也有足够的信心不要觉得 Python 慢。另外八卦一下，现在 Python 之父前两天从 google 去 Dropbox 了，这是很值得自豪的事， 值得 Dropbox 为之自豪。

Python 是荷兰人 van Rossum1991年开发完成的脚本解释语言。起这个脑缺的名字是因为他是一个叫做 Monty Python 的脑缺喜剧团体的脑残粉（BTW，Monty Python 出演的巨蟒与圣杯是英国电影史上跟大话西游同样地位的喜剧，其中亚瑟王被黑成了炭，里面圆桌骑士们拿着块石头敲来敲去各处蹦达着，看影评我才知道这是表示他们在骑马%&……×（). 于是人们知道以这么脑残的名字取的语言不是像 brainfuck 语言一样是 brainfucker，那么就会像莫里盖尔曼以乔伊斯“芬尼根的守夜人”中虚构名词来命名的夸克一样，成为一个一个不朽的新创造。Python 显然属于后者。

接下来，说正题，为什么 Python 如此先进（对于初学者）。

代码简洁性和可读性

写过 hello world，hello android， hello **的人都知道，学语言最好的途径就是写和读（即使是学书面的自然语言）。所以代码的可读性是选择学一门语言的关键因素，因为你代以后会花很多时间读别人的代码。可读性带来的影响是非常深远的。有种说法，说在遥远的古代阿拉伯数字传入之前欧洲之前，其数学发展几乎为0，而造成这种缓慢的原因就是因为复杂的罗马数字的广泛使用。这表明很多时候即使我们不愿意承认，往往是形式决定的内容．比如罗马数字没有0， 自然很多数学概念就难以发展．没有流形也不可能发展广义相对论一样．所以.....如果想以后从此过上幸福的生活，请不要选用 perl． 如果不幸选择了 perl，那么就君就 一入侯门深似海，从此萧郎是路人 了。当以后你两行清泪的看着自己十天前写的不过10几行的楔形文字时，你就会明白．

而 Python 的可读性是我见过最好的:

1. Python 的代码格式使用缩进而不是括号。 首先节省了很多行数，变得而为紧凑，而美观．相传的俄罗斯人偷美国 NASA 的 C 代码那个段子满屏括号的情况是不可能出现 Python 版本的．第二，逻辑相当清晰．循环的结束与开始一目了然． 第三，屏幕右方得到充分利用．比如使用24寸屏幕的人是不是感觉自己总是望着左边编程.....和17寸等高的屏幕区别不大，很费右边的电．

比如，这是某个 C 用来图像采样的算法的代码:

好吧，很带萌感，画风也很不错.

但是 Python 也不是写不出混乱的代码，或者说只要有一定自由性的语言就可以写出。比如没事写个(Pseudo Code): boolean AlwaysTrue = False; 或者 int MagicNumber =42; 什么的。

这个是 Python 版，完全不符合 Python 哲学。

Python 的思想

上面一点是语言形式的，C 也可以改成缩进， 所以此项不是核心优势. 而使用一门语言是使用它的思想. 于是要谈到 Python 的哲学，The Zen of Python. 在 python 命令行里输入 import this 可以看到，:

Beautiful is better than ugly.美优于丑

Explicit is better than implicit.晰胜于浑

Simple is better than complex. 简胜于繁

Complex is better than complicated. 繁胜于杂

Flat is better than nested. 平胜于嵌

Sparse is better than dense. 稀胜于稠

Readability counts. 可读至上

Special cases aren't special enough to break the rules. 殊例不足违训

Although practicality beats purity. 虽实用大于纯粹

Errors should never pass silently. 谬不可疏

Unless explicitly silenced. 除明示

In the face of ambiguity, refuse the temptation to guess. 晦不存疑

There should be one-- and preferably only one --obvious way to do it. 一法万用

Although that way may not be obvious at first unless you're Dutch. 若非尼德兰红毛，法难定 （.....科学没有国籍）

Now is better than never. 今胜于无

Although never is often better than *right* now. 无胜于促

If the implementation is hard to explain, it's a bad idea. 难述其施，谬法也

If the implementation is easy to explain, it may be a good idea. 其施可述，或可行

Namespaces are one honking great idea -- let's do more of those!命名空间，多多益善

如果觉得俺翻译得太烂可以看英文. 但是你看到一个物品/工具/器件/用品的设计理念是如上时, 除开认为设计者装逼之外, 必定会觉得这个工具是十分可靠, 清爽的. 首先看到这个蹩脚的诗我就想到的是 Unix 伟大的亲嘴原则（Keep It Simple and Stupid), 而 Python 的确如此。（部分）解读如下：

美优于丑。我只想举一个例子，你们感受一下（哈哈哈哈）。学 C 语言写辗转相除的 novice 们先 include 你的 stdio 去吧:

```
---
def gcd(x, y):
    while y:
        x, y = y, x%y
        return x
----
```

明晰胜于浑晦:

Python 逻辑与或用 and 和 or 而不是 && 和 ||

简胜于繁，繁胜于杂，平胜于嵌，稀胜于稠 可读至上:

分别用 C 和 Python 写个 hello world 就能明白。

特例不足违反规则:

这句话最好的例子是 Python 没有 char 类型。因为大伙儿认为 char 只是一位的字符串。

实用大于纯粹:

于是 Python 里有 chr().

谬不可疏 除明示:

有时候 exception 还是重要的

晦不存疑，一法万用:

1. 与其用 iterator, for 循环, 指针移动，数组下标移动， 不如一个 for...in...直接解决问题。

2. list，dict 和 tuple 的数据结构也是希望能适用于绝大部分场合。这三位可以在必要的时候变成：列表，栈，队列，哈希表，七七八八各种东西....

3. 函数的参数也勉强可以算进来。利用*和**, python 的函数参数灵活性无以复加。

若非尼德兰红毛，法难定:

第一次看到荷兰人写的 xx= if yy>0 then 0 else 1 是不是想哭...

中间几句太哲学了.....:

略过....

命名空间，多多益善：

面向对象...

要不是科比最近战绩太差...这些特性简直可以形容为黑曼巴...灵活，准确，快速，强力。

Python 语法的优美之处数不胜数，难以名状，深入人心，犬牙交错，人神共愤，不随意肌。语法是思想的延伸，有人说你学一门新语言而不学习新的思想则。还是新评装旧酒。由于我不是写 tutorial，就不一一讲述所有的细节了。 只列举下最好和简单的特性, 而像 decorator（面向切面），generator，多线程，itertools，一次肯定也讲不完。

1. Python 中的整数相当于 C 中的长整型(long), 32 位的机器上整型取值范围为 -2147483648 至 2147483647, 64 位机器上为 -9223372036854775808 到 9223372036854775807。Python 的长整型是无限制的，只要内存允许。很相似的是 Python 里的无限 list。一个很著名的例子是使用生成器(generator)，就可以生成一个无限长的

Fibonacci 数列;

```
def fib():
    a=b=1
    while True:
        yield a a, b= b, a+b
```

这个数列号称无限长，其实是需要运算哪一位时才计算。这就是著名的惰性求值。Python 中的长整型和无限 list 的概念均来自于 Haskell。

对于 C 和 C++要处理大数据要使用高精度算法，用一个 struct 表示一个大数，使用一个 array 储存它，然后自定义运算函数(加减乘除)。

2. List comprehension，切片等操作

使用 list comprehension 可以杜绝掉50%以上的 for 循环，后者的效率极其低下(可以看看 C 源码的实现)，而且不够紧凑。我之前上面举的第一个粒子就是 list comprehension 的很好的运用。随便举个 python 官方文档的粒子：

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x !=y]
```

运行结果得到，

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

反转一个数列的例子，是个 one-liner ：

```
lis[::-1]
```

3. 正则表达式

Python 正则表达是内置的。一个粒子是我在实验室测试 Josephson Junctions 时碰到的情况，简化的说就是：我有几百个文件夹，每个文件夹有几百个文件，每个文件有几万条数据，每几个数据我要处理完然后存在另外一个文件夹的另外一个文件里。我要做的工作有：

-用正则表达式找到我要的文件夹和文件，剔除不需要的。

-逐行读出 txt 文件里面的数据

-每几个数据设个参数平均，最小二乘处理，剔除几个极端情况，画出图

-保存 这里 Python 就起到了一个脚本语言应尽的责任了。

整个从打开文件到保存 不超过150行，还加上我罗哩叭嗦的注释。如果用 C 写... 呵呵呵呵呵呵。当然这个用 Shell 也不会太差，不过实验室用的是 windows，用 windows 脚本我还不如去死。自从我写完这个脚本后，从此我们实验室的 testing 就过上了幸福的生活。（可能么？）

4.reduce, lambda, filter 和 map

这些全来自于函数式编程。比如找到 prime 之内的质数：

```
filter(lambda prime: all(prime%num for num in range(2, prime)), range(2,prime))
```

如果作为中文读出来则是（prime 是之前给定的一个数）：在2到 prime 之间，过滤出那些所有不被2到自己整除的数。难道还能更简单么!如果用的 C，呵呵。这里出现了 Zen of Python 没有提到但是是 Python 里非常重要的一点，对“数”的操纵。毕达哥拉斯信奉一切皆数，程序语言更应算更是如此，只有对“数”和“类型”的完全掌控，才能如鱼得水。

5. 语言的动态性 Python 是动态语言，这是非常重要的一点，一直忘了说。这一点可以直接完爆 C++自己一向自豪的泛型编程，模板编程。且看一个 strangeness 为0的粒子：

```
def build(type, value):
    return type(value)
build(int, 0)
```

所以稀饭们请看过来， 你们家 C 可以三行写出这种东西么？!! 没完，接着：

```
def impose(func, value):
    return func(value)
def anyfunc(value):
    return value*value
print impose(anyfunc, value) #此处是 python2.7的语法
```

一看就知道是函数式编程。请问 C 可以么？！

当然还有之前说的函数参数的灵活性：

如果定义一个函数

```
def print_whatevercrapsthefuckinguserinputs(**params):
    print params
```

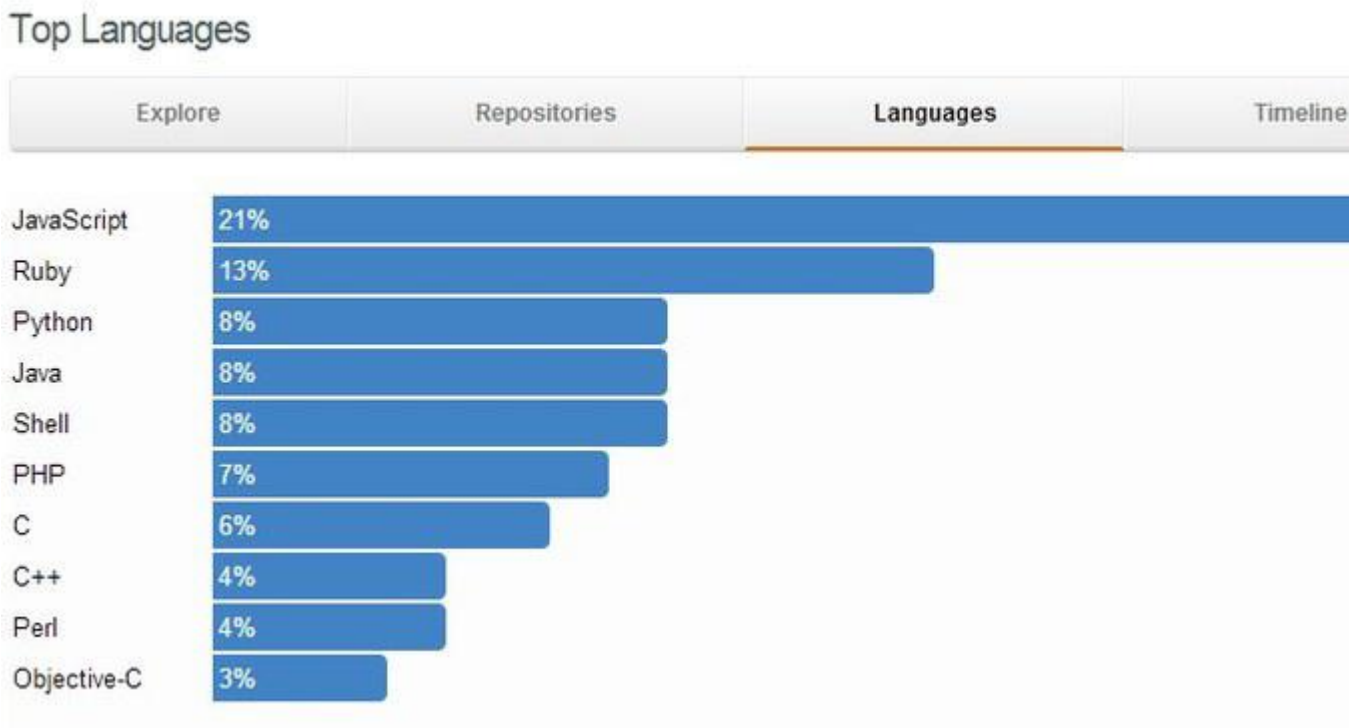
你就可以想输入多少参数就输入多少了，比如 print_whatevercrapsthefuckinguserinputs(“I",”dont", "give", "a", "fuck“),

只要有 print 函数的接口（这又涉及了鸭子类型和类似 haskell 的 typeclass 的性质，呱....）。其实际作用是 比如你想在数据库里输入大批量用户信息，mi amigo，对于这样一个蛋疼的函数名字，调用一次就够了。

在 Python 里类型，函数，全部都是可操作的对象。这可以改变一切。第一个例子显示了对数据类型的操作，第二个是对函数的操作,第三个是对参数的控制。我不想想象用 C 写这个例子了，因为 C 根本写不出来。 以上所有这些 Python 的特性，你可能说我用 C 实现一个一样的就好了。诚然，你可以在 C 里一个个写出来自己喜欢的特性，但是你写到后面你会发现你只是再次发明了 Python，然后拿 C 重新写了一个解释器， 而且实现得更烂而已。Please! Don't re-invent the wheel.

Python 的类库齐全

对于我们普通人来说一个语言最重要的还是库函数的齐全程度，Java 在此方面已经登峰造极了。而现在的 Python 不输于他。一个语言的火热程度可以从类库看出， 下面是 github 上 语 言 的 排 名 ：



由于 Javscript 的特殊性和 Ruby 的 RoR 我们就不去管它了，Python 占8%,C 占6%. 貌似差得不远, 恩. 但是如果你仔细看 C 的 project，随便翻十页可能发现会有三页的项目其实是在写 python 的类库. 真是母亲为孩子显出一切.

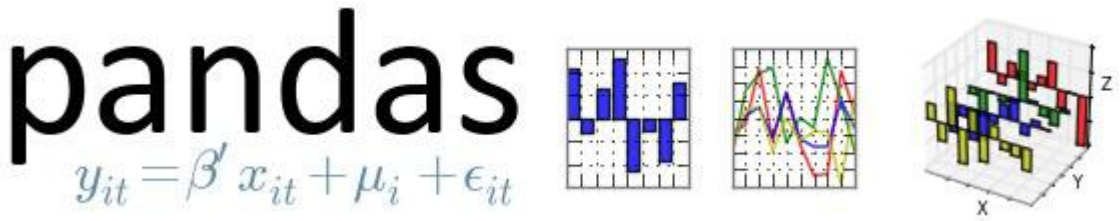
另外一个说明 Python 类库齐全的例子是我这个学期被某教授压着要算一个固体模型的 Berry's Phase. 正值 final 之前, 如何有心思写这厮. 一筹莫展之际竟然发现 python 有个固体算能带和巴里态的库!! nm 这也能有!? 仔细观摩了下源代码, 1000来行,干净整洁, 速度用之，皆大欢喜. 这种小众库都有我已经不能想象你有什么变态要求 Python 不能满足了. 以下是一些常用的类库.

1.如果你想写网络应用，轻量级：web.py （web.py 的作者最近自杀去世，RIP）中量级：Django,Pylon 重量级：twist。其实很多人诟病 Python 的网络框架过于多，不能集中起来，我倒觉得无所谓。我用过 Django， 写个小小的博客程序，1000多行，这是用 java 不可想象的。 Youtube 上有个半个小时的用 Django 写博客的演示：http://www.youtube.com/watch?v=srHZoj3ASmk 。 半个小时!! 一边写一边讲！一个博客程序!! 还带后台功能！What the F*! 是编程么！？我脱稿写个平衡树都不只半个小时啊！

2.如果你想做科学计算，NumPy&SciPy 可以取代90%matlab 的常用功能，然后让我想一想，他们好像只有几十 mb!!加上 Pythonxy 也才700mb 多!但是你没有更多的新学语法的成本，也不用付给 mathwork 血汗钱。

3.网络爬虫。Python 的超强项，beautifulsoup 的网页解析，scrapy 等等，不一而足。Twitter，微博等好像也有 python 的 API。

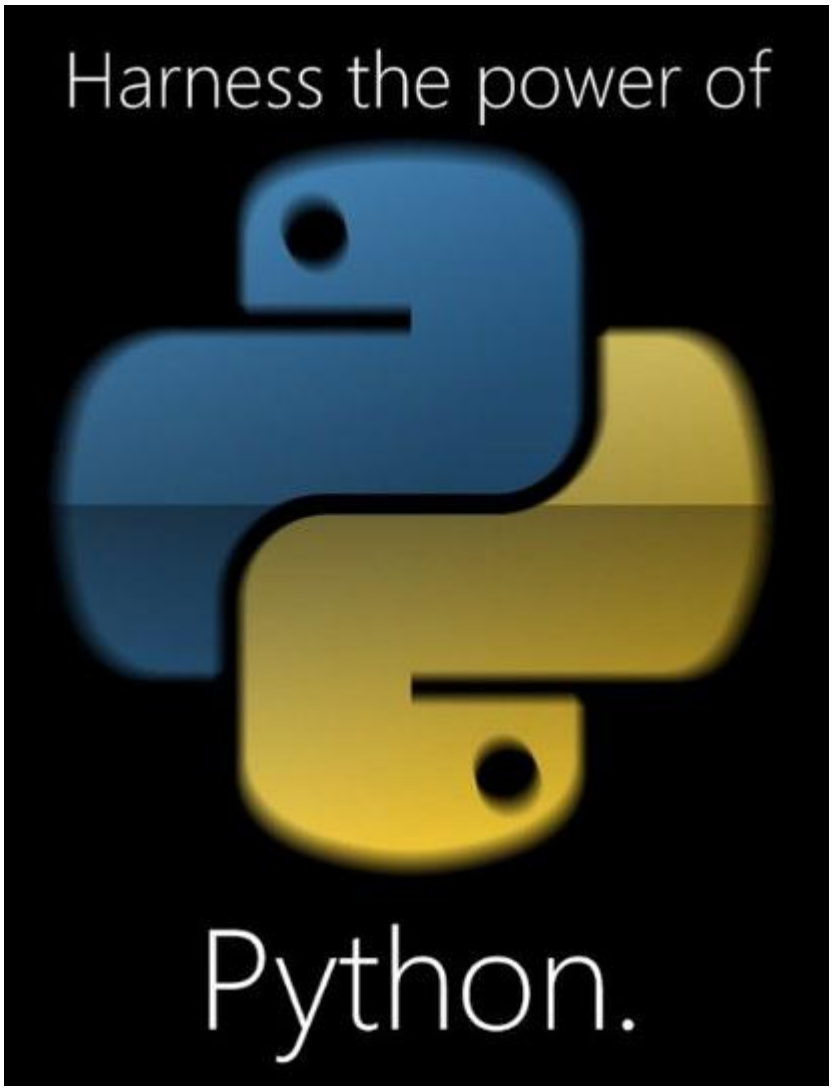
4.Machine Learning，数据分析和 Natural Language Processing。 请去 Kaggle 上看看多少参赛者是用的 Python。 著名的库有 Pandas。



5.写游戏。Pygame。 试过几个别人写出来的游戏，非常顺畅。

6.桌面应用。 pyqt。Dropbox 貌似就是使用的 pyqt 写的。

最后我想讲下 Python Challenge。Python 作为一个强劲的编程语言有着极为活跃的社区，文档丰富，教程齐全。当然就有很好的网上解谜过关类的教程。Python Challenge 是在各个类型的解谜过关性的我看到过的最好的一个。难度适中，而且可以从中学到很多。解法不局限于 python，可以用 perl， shell，C 甚至 Erlang！官方解答往往有10多种解法，精妙至极。但是你会一步步从中发现 Python 的优势。一共33关，在充满乐趣的智力挑战和极大的满足感后，你可以学到 PIL 库的使用，pickle 的使用，正则表达式，完成后你会发现思维方式的改变。 什么？不知道网址在哪里？Let me google that for you。 我做了半年多已经完成得还行了，我会把答案和分析帖在这里。



Last words

神爱众人，于是带来 Python。

----------PIRATICUS 13:7