# 完全用python工作

## 前言

曾几何时，我是 C/C++的坚实拥护者，花了近乎5000小时在上面，里面的奇技淫巧多如牛毛，令人无所适从，尤其是虚函数、虚继承、指针、垃圾回收等一系列背后原理的东西，总是感觉智商捉急。Python就完全没有了这些困扰，但坦白的讲，我学python是走了很多弯路的，linux系统一般是自带python的，而windows由于微软对自家C#的袒护，不仅没有自带，而且在windows下使用python也远非linux下那般顺畅。那我为什么还要费这么大的功夫转向python呢？在[鱼是最后一个看到水](http://blog.csdn.net/pongba/article/details/2025830)的中也提到了这点，C++越来越不能满足现有开发的需求，庞大、臃肿、学习成本高，收益却不怎么丰厚，远不及新出的语言来的划算，最致命的是设计范式上的缺陷，把所有的工作交给程序员，不断的重复造轮子，它是给机器看的，不是给人看的，如果这种现象不改变的话，不久的将来，C++只能和汇编一样被扔进历史的垃圾堆。

随着C++11和14的引进，C++正在python化，参看[C++逐渐python化](http://www.oschina.net/translate/cpp-has-become-more-pythonic)一文，学习python是大势所趋。看看[C、C++、python、Java、php、C#六种流行语言大PK](http://www.jizhuomi.com/software/376.html)，就能一眼看出python的简洁优雅，随着大数据时代的来临，[我为什么说Python是大数据全栈式开发语言](http://www.thebigdata.cn/JieJueFangAn/14928.html)，python的地位只会更加稳固。我主要用到数据处理的部分，也就是机器学习的部分，所以后面讲到搭建python环境时请注意自己的用途。

还是再补一刀吧：函数式编程被证明能有效的提高开发效率（全局变量、代码、作用域的问题参考 lua upvalue），面向对象编程坑了一代人。

## 环境搭建

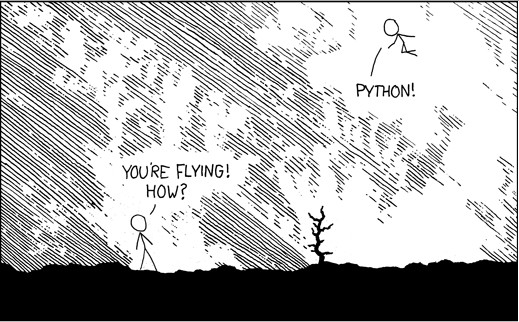
以前不知道，装了python(x,y)2.7.9，结果装不上深度学习的库theano，就半途而废了，后来才知道，这么干是不行的。Windows下装python是有门道在里面的。直接上我的方案好了：

IDE选Anconda2.1.0，不要选后面的版本，它们取消了对MinGW的支持，不然你是编译不过theano的。

其他库的安装一般是pip install \*\*\*，或者下载原文件后执行python setup.py install，相当的方便。

## 为什么非python不可

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）

作为机器学习的实践者, 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.

## [Philosophy in Python](http://www.cnblogs.com/stewarttzy/archive/2012/12/31/2840502.html)

在python命令行输入import this就能出现python的设计哲学：

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. 即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）

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. 虽然这并不容易,因为你不是 Python 之父（这里的Dutch是指Guido）

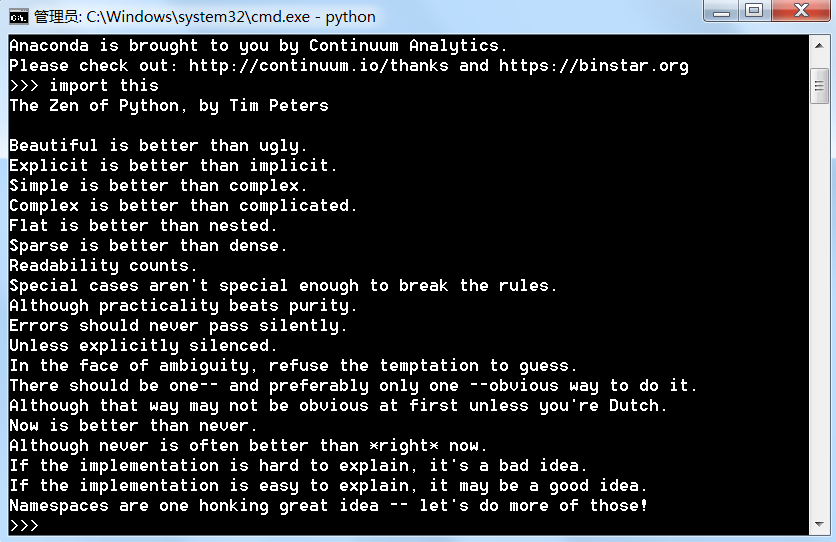
Now is better than never. 今胜于无

Although never is often better than \*right\* now. 无胜于促；做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）

If the implementation is hard to explain, it's a bad idea. 难述其施，谬法也；如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）

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



## 特性

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

print\_whatevercrapsthefuckinguserinputs(“I",”dont", "give", "a", "fuck“), ，比如

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！

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。

## 相关教程

[简明 Python 教程](http://old.sebug.net/paper/python/)

[Python教程](http://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000)廖雪峰

[python基础知识](http://blog.csdn.net/google19890102/article/category/3046675)

[Python快速教程](http://www.cnblogs.com/vamei/archive/2012/09/13/2682778.html)

[Python函数式编程指南](http://www.cnblogs.com/huxi/category/251137.html)

[Python3.x和Python2.x的区别](http://www.cnblogs.com/codingmylife/archive/2010/06/06/1752807.html)

[python计算机视觉项目实践](http://blog.csdn.net/ikerpeng/article/details/25027567)

[用Python和OpenCV创建一个图片搜索引擎的完整指南](http://blog.csdn.net/kezunhai/article/details/46417041)

[用Python开始机器学习](http://blog.csdn.net/lsldd/article/category/2709209)

[theano学习指南](http://www.cnblogs.com/xueliangliu/archive/2013/04/03/2997437.html)

## 参考

王垠：[完全用Linux工作](http://www.cnblogs.com/skyseraph/archive/2010/10/30/1865280.html)

石雨浓：[强大的Python--完全用Python工作](http://blog.renren.com/share/100366304/15032720056)