Python教程

v3.0

# 前言

Python3.x可用于文本处理、数值及符号运算、绘图等，文档基于Win10、Python 3.5.2 |Anaconda 4.2.0 (64-bit)。

更新：<http://www.misaraty.com/21>。

misaraty

misaraty@163.com

2016.12.15

# 目录

[前言 1](#_Toc468046921)

[目录 2](#_Toc468046922)

[安装 3](#_Toc468046923)

[字符串 4](#_Toc468046924)

[for 6](#_Toc468046925)

[while 9](#_Toc468046926)

[if 11](#_Toc468046927)

[列表 12](#_Toc468046928)

[字典 14](#_Toc468046929)

[正则表达式 17](#_Toc468046930)

[函数 20](#_Toc468046931)

[模块 28](#_Toc468046932)

[C扩展 31](#_Toc468046933)

[格式化 32](#_Toc468046934)

[文件 34](#_Toc468046935)

[性能优化 35](#_Toc468046936)

[参考 38](#_Toc468046937)

[主要参考 38](#_Toc468046938)

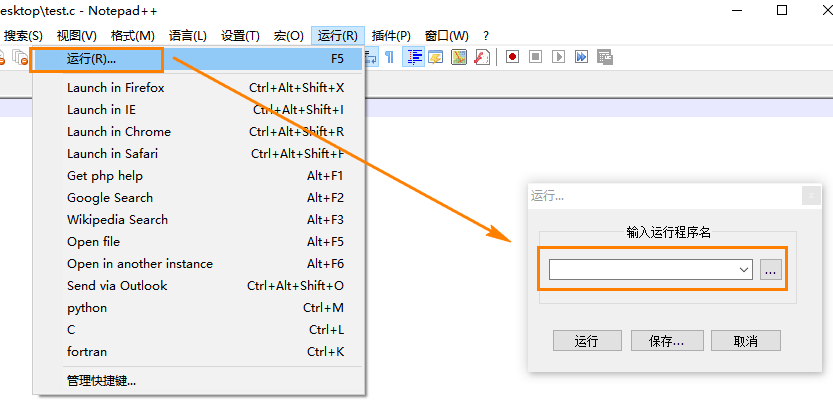
[次要参考 39](#_Toc468046939)

# 安装

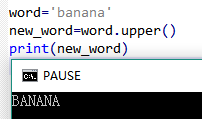
下载[Anaconda](https://www.continuum.io/downloads)，64位，默认加入path。

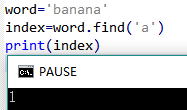
配置notepad++，

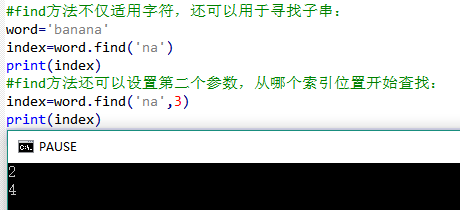
输入cmd /k python "$(FULL\_CURRENT\_PATH)" & ECHO. & PAUSE & EXIT

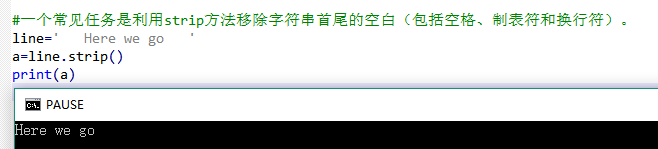


# 字符串







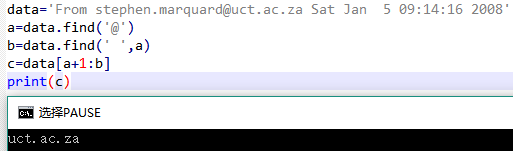


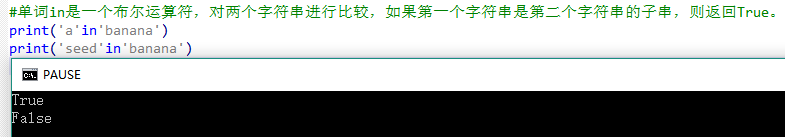
通常，我们想要在一个字符串中寻找它的子串。如下是一行结构化的字符串：

From stephen.marquard@ uct.ac.za Sat Jan 5 09:14:16 2008

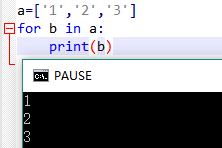
我们只想抽出电子邮件的第二部分（即uct.ac.za），可以通过find方法和字符串切片来实现。

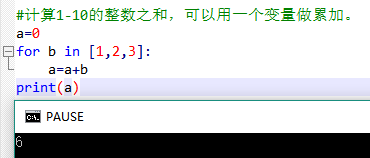
首先，在字符串中找到@符号的位置。其次，找到@符号之后第一个空格所在的位置。最后，再用字符串切片来提取字符串中我们需要的部分。

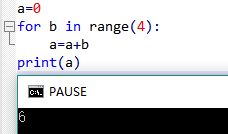


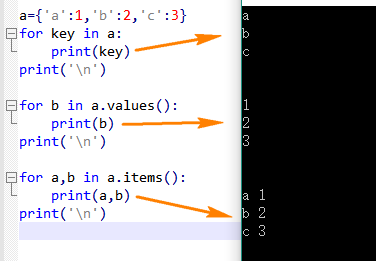


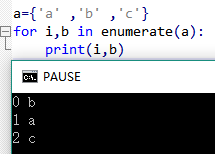
# for

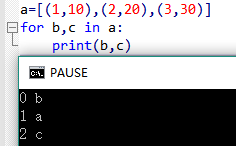


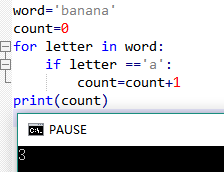


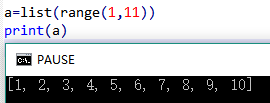


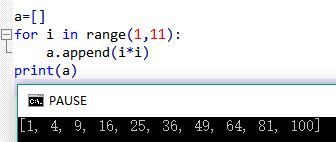


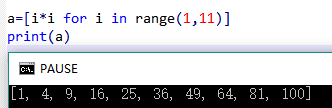


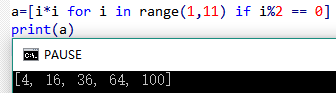


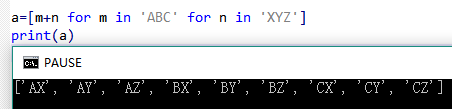




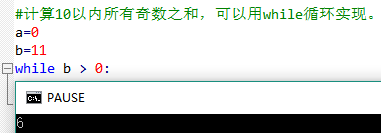


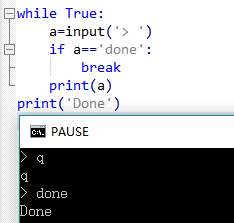


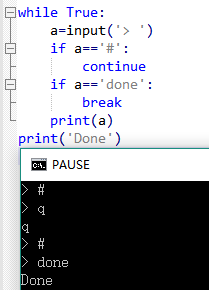


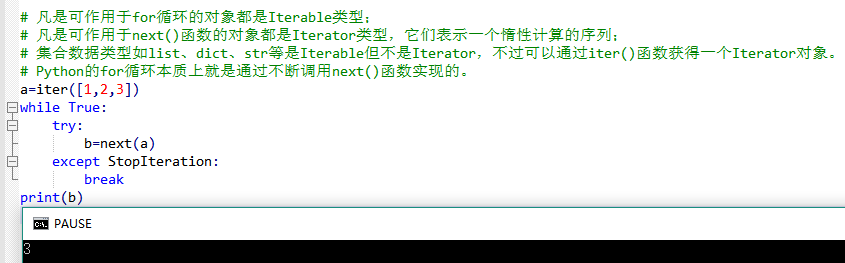


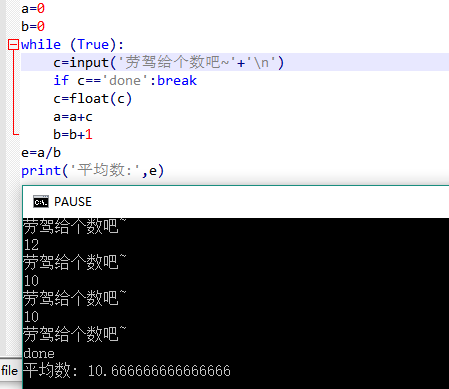
# while

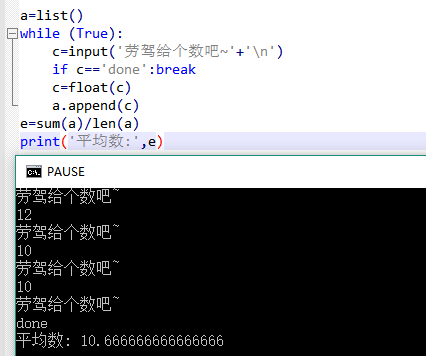




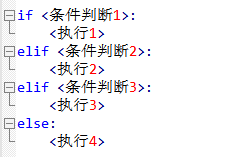


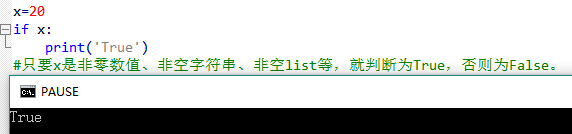






# if





if x == y:

print 'x and y are equal'

else:

if x < y:

print 'x is less than y'

else:

print 'x is greater than y'

inp = raw\_input('Enter Fahrenheit Temperature:')

try:

fahr = float(inp)

cel = (fahr - 32.0) \* 5.0 / 9.0

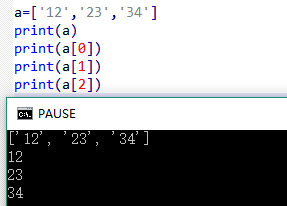
print cel

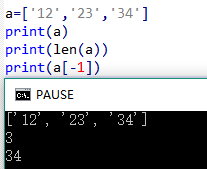
except:

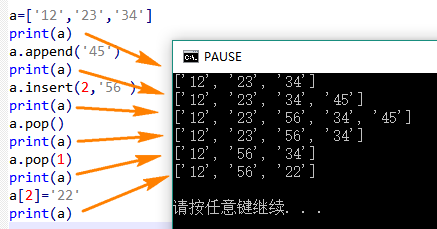
print 'Please enter a number'

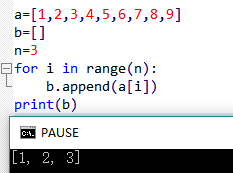
Python首先执行try语句块。如果一切顺利，它就会跳过except语句块。如果在try语句块里发生意外，Python就会跳出try语句块，执行except语句块。

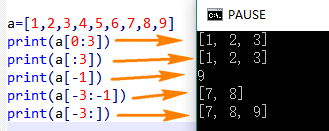
# 列表

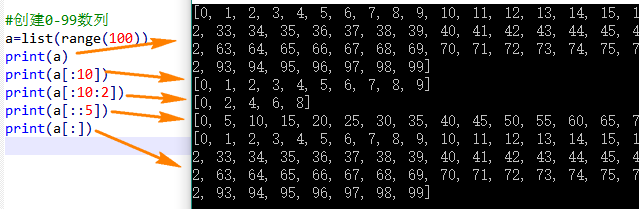


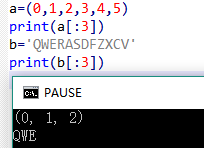




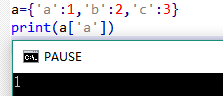


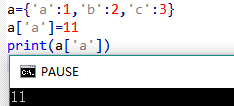


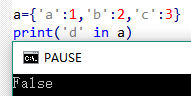


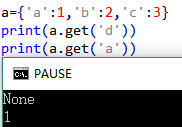


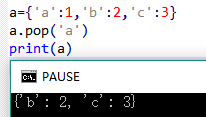
# 字典

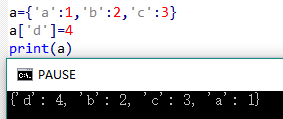


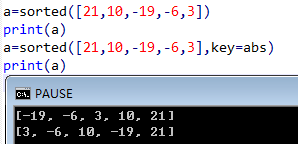


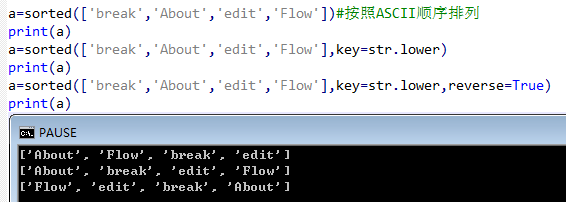




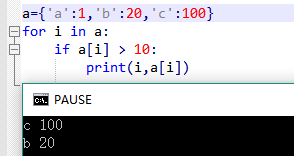




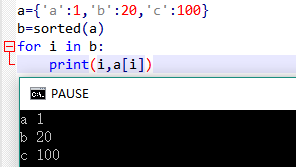




如果想要找到字典中所有值大于10的记录，程序代码如下：



如果按照字母顺序输出字典中的键，首先使用字典对象的keys方法，将所有的键放入一个列表。然后，对这个列表进行排序，对排序后的列表进行迭代。依照字母顺序，通过每个键输出对应的键值对：



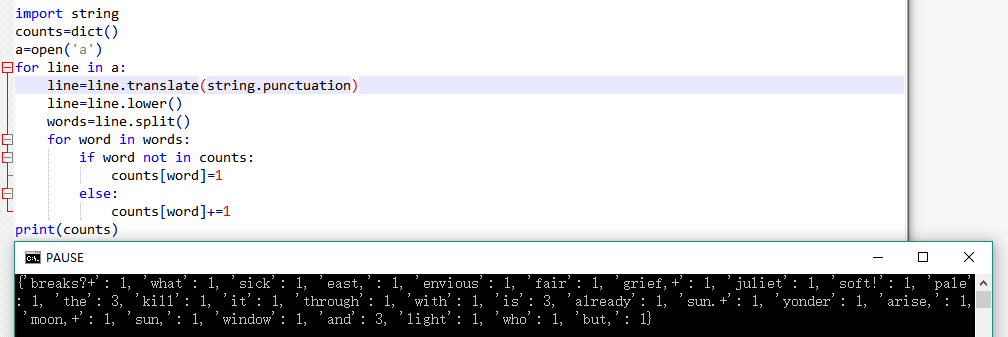
a文件内容，

But, soft! what light through yonder window breaks?

It is the east, and Juliet is the sun.

Arise, fair sun, and kill the envious moon,

Who is already sick and pale with grief,



# 正则表达式

a文件，

From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

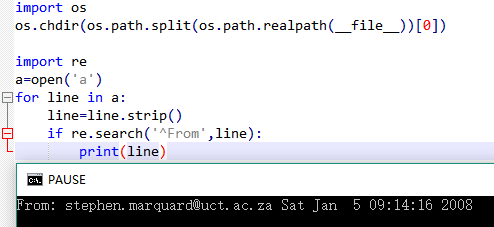
Return-Path: <postmaster@collab.sakaiproject.org>

for <source@collab.sakaiproject.org>;

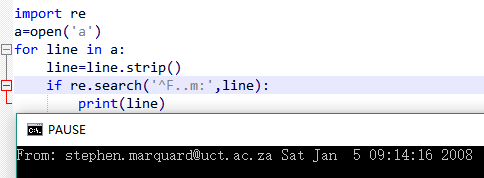
Received: (from apache@localhost)

Author: stephen.marquard@uct.ac.za

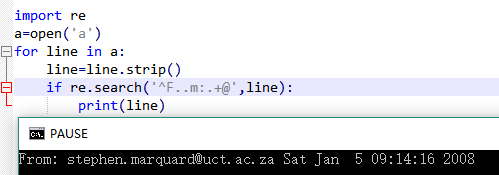
仅匹配“From”开头的文本行，



正则表达式"F..m:"会匹配配“From:”、"Fxxm:"、“F12m:”或"Fl@m:"。正则表达式的句点可以匹配任意字符。

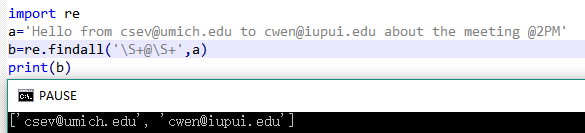


“\*”和“+”表示一个字符可以重复任意次数，在构造正则表达式时结合这种能力特别有用。这些特定字符用来代替单个字符，星号匹配零或多个字符，加号匹配一个或多个字符。

进一步减少代码，以下示例使用重复的通配符：

我们不想为每个文本行类型编写代码，每个文本行都分割和切片一次。以下程序使用findall()找到文本中的电子邮件地址，从每一行抽取一个或多个电子邮件地址。

findall()函数搜索第二个参数的字符串，返回一个包含形如电子邮件地址字符串的列表。我们使用两字符序列来匹配非空字符(\S)。



一些电子地址的开头或结尾包含了不正确的字符，如“<”或“;”。这里声明一下，仅需要以字母或数字开头和结尾的字符串部分。

要做到这一点，我们使用正则表达式的另一个功能，使用方括号罗列多个可接受的匹配字符。在某种意义上，“\S”匹配的是非空字符的集合。现在，我们更清楚一些字符匹配的本质了。

下面是新的正则表达式：

[a-zA-Z0-9]\S\*@\S\*[a-zA-Z]

如果我们想要找到以“X-”开头的文本行，如下所示：

X-DSPAM-Confidence: 0.8475

X-DSPAM-Probability: 0.0000

我们不仅需要文本行中的浮点数，还需要统计符合以上语法的文本行数。

使用下面的正则表达式来挑选出符合要求的文本行：

^X-.\*: [0-9.]+

解释一下，文本以“X-”开头，之后是零个或多个字符“.\*”，然后是一个冒号和一个空格。空格之后是一个或多个字符，可以是一个数字(0-9)或一个句点“[0-9.]+”。需要注意的是，方括号中的句点实际匹配的是句点本身，也就是说，它在方括号内不是通配符。

^ 匹配文本行的开头。

$ 匹配文本行的结尾。

. 匹配任一字符（一个通配符）。

\s 匹配一个空白字符。

\S 匹配一个非空字符（与\s相反）。

\* 应用于前接字符，表示前接字符的零个或多个匹配。

\*? 应用于前接字符，以非贪婪模式，表示前接字符的零个或多个匹配。

+ 应用于前接字符，表示前接字符的一个或多个匹配。

+? 应用于前接字符，以非贪婪模式，表示前接字符的一个或多个匹配。

[aeiou] 匹配指定字符集中的一个字符。这里只能是“a”、“e”、 “i”、 “o”或 “u”，不接受其他字符。

[a-z0-9] 使用减号指定字符区间。这里表示一个字符，必须是小写字母或数字。

[^A-Za-z] 第一个字符是^，它表示反向逻辑。这里匹配除了大小写字符之外的其他任意字符。

( )在正则表达式中添加括号，括号内容会丧失匹配功能，但在findall()中可以用于抽取特定部分的字符串，而不是整个字符串。

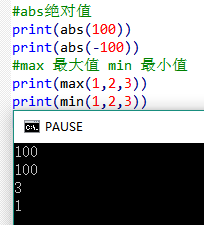
\b 匹配空字符串，仅用于单词的首尾。

\B 匹配空字符串，但不能用于单词的首尾。

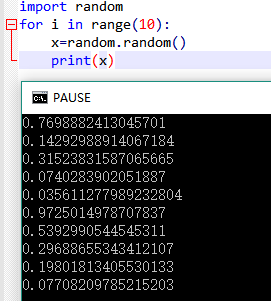
\d 匹配任意十进制数字，等价于[0-9]。

\D 匹配任意非数字字符，等价于0-9。

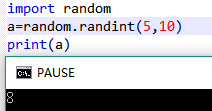
# 函数



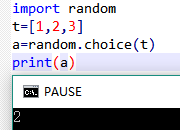
random函数返回一个介于0.0-1.0的随机浮点数（包括0.0，但不包括1.0）。每次调用时，你会得到一长串的数字。



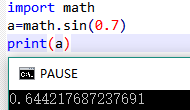
random函数只是众多随机数函数中的一个。randint函数传入两个整数参数，并返回介于这两个参数之间的整数（包括这两个参数在内）。

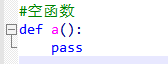


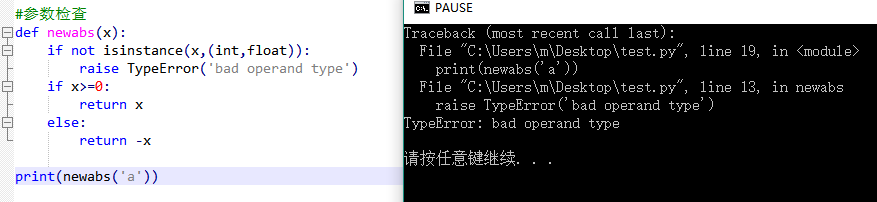
为取得一组值中的随机元素，可以使用下面的程序：

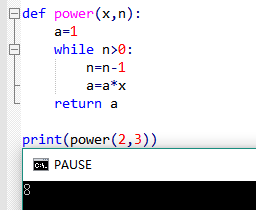


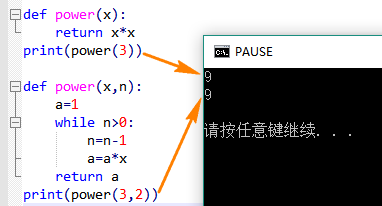
Python的math模块提供了很多常见的数学函数。在使用之前要导入相应的包。为了使用其中的函数，你必须指定模块名和函数名，用圆点隔开，也就是英文的句号。这种格式叫做点标记（dot notation）。

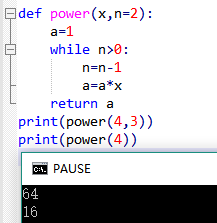


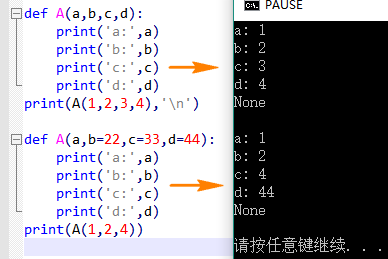


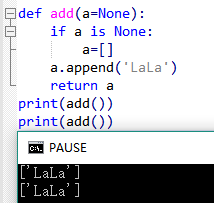


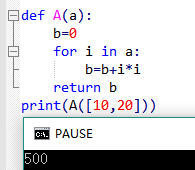


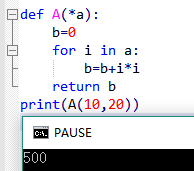


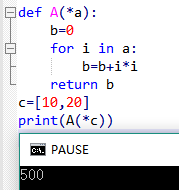


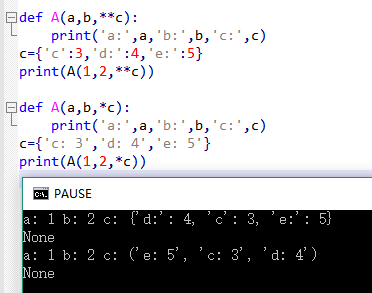


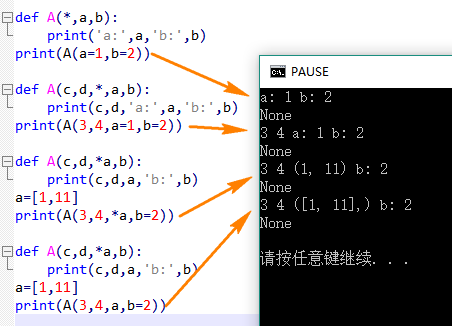


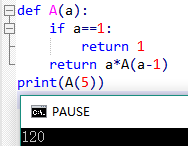


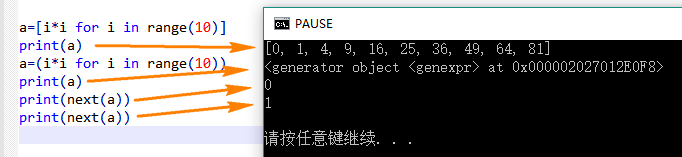


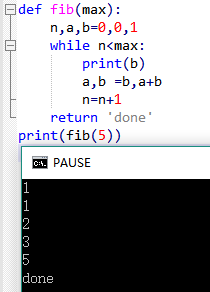


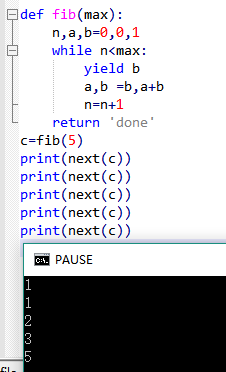


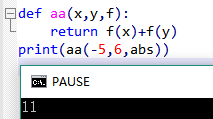


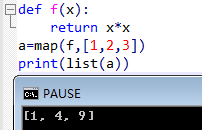


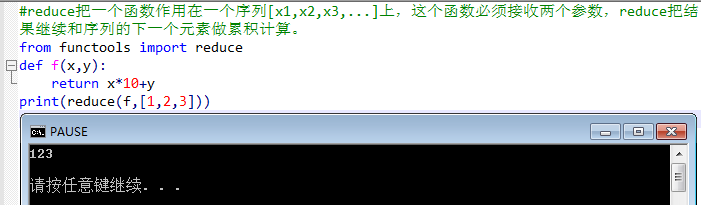


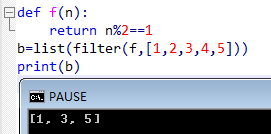


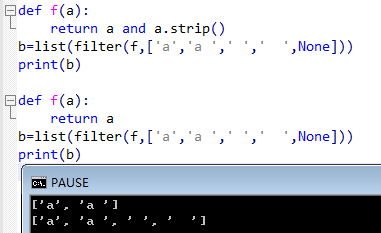


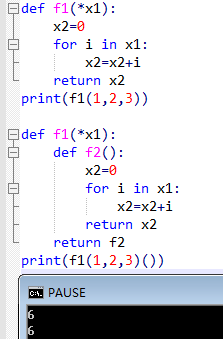


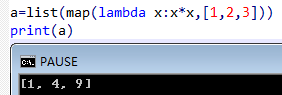




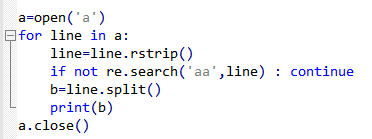








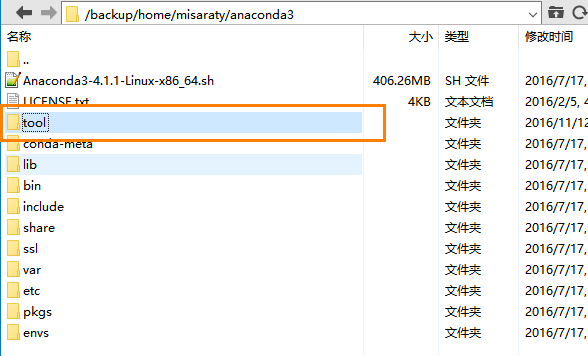
行间解析。split方法处理此类问题非常有效。先编写一个小程序，找到以“aa”开头的句子，然后把这些句子分解，最后输出句子中的第三个单词：



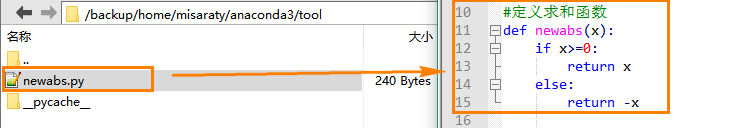
# 模块

以anaconda64位+linux为例。

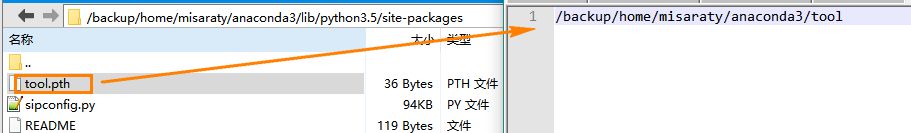
1 在集群anaconda安装目录下新建tool文件夹（位置自拟，只要与后面与之对应上就好）。



2 在tool目录下，新建newabs文件，文件内即为newabs函数。



3 打开~/anaconda3/lib/python3.5/site-packages，新建tool.pth(名称任意)。



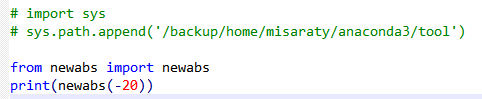
4 在linux目录下新建a.py，

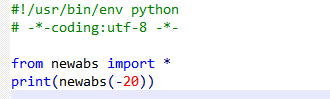


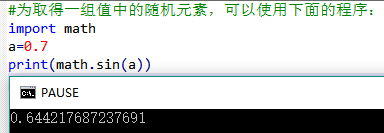
python a.py。

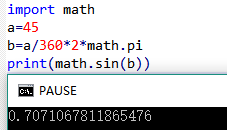
或者，

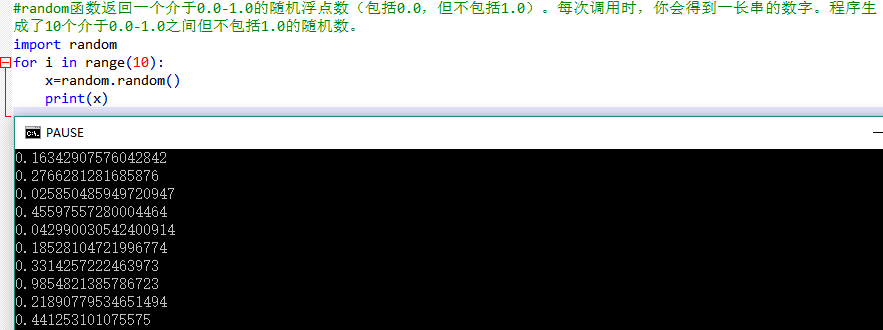
不按照1 2 3 4步骤（不推荐），加路径，

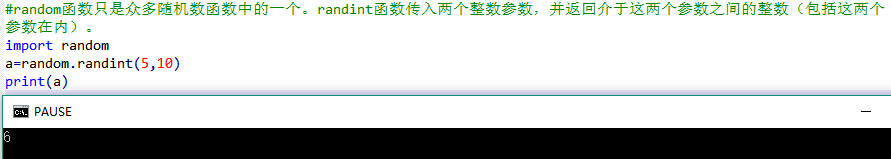


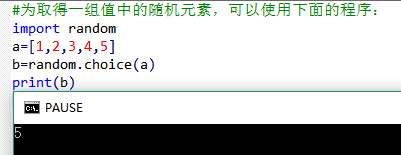






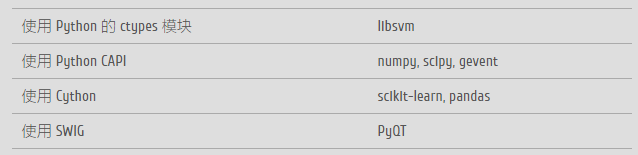




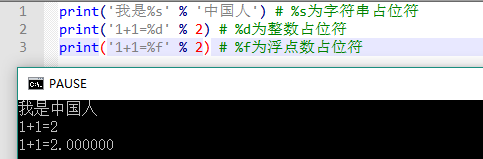


# C扩展

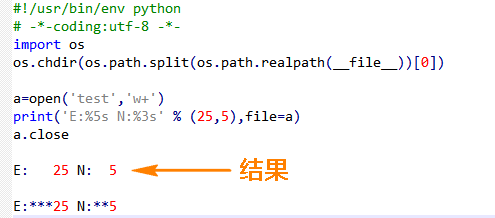
Python调用C，首选Cython，次选C API。



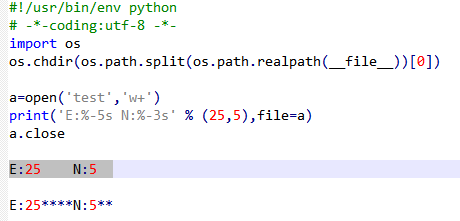
# 格式化



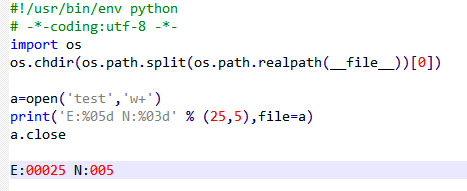
指定占位宽度，



左对齐占位宽度，

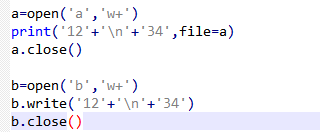


用0填充占位，



# 文件

写入文件，write是原样写入，print会在最后加一空行。







# 性能优化

**优化算法时间复杂度**

算法的时间复杂度对程序的执行效率影响最大，在Python中可以通过选择合适的数据结构来优化时间复杂度，如list和set查找某一个元素的时间复杂度分别是O(n)和O(1)。不同的场景有不同的优化方式，总得来说，一般有分治，分支界限，贪心，动态规划等思想。

**减少冗余数据**

如用上三角或下三角的方式去保存一个大的对称矩阵。在0元素占大多数的矩阵里使用稀疏矩阵表示。

**合理使用copy与deepcopy**

对于dict和list等数据结构的对象，直接赋值使用的是引用的方式。而有些情况下需要复制整个对象，这时可以使用copy包里的copy和deepcopy，这两个函数的不同之处在于后者是递归复制的。后者慢一个数量级。

**使用dict或set查找元素**

python dict和set都是使用hash表来实现(类似c++11标准库中unordered\_map)，查找元素的时间复杂度是O(1)。

**优化循环**

循环之外能做的事不要放在循环内，比如下面的优化可以快一倍。

**优化包含多个判断表达式的顺序**

对于and，应该把满足条件少的放在前面，对于or，把满足条件多的放在前面。

**使用join合并迭代器中的字符串**

**不借助中间变量交换两个变量的值**

**使用if is**

使用if is True比if==True将近快一倍。

**使用级联比较x < y < z**

**while 1比while True更快**

**使用\*\*而不是pow**

**使用C扩展(Extension)**

目前主要有CPython(python最常见的实现的方式)原生API, ctypes,Cython，cffi三种方式，它们的作用是使得Python程序可以调用由C编译成的动态链接库，其特点分别是：

CPython原生API: 通过引入Python.h头文件，对应的C程序中可以直接使用Python的数据结构。实现过程相对繁琐，但是有比较大的适用范围。

ctypes: 通常用于封装(wrap)C程序，让纯Python程序调用动态链接库（Windows中的dll或Unix中的so文件）中的函数。如果想要在python中使用已经有C类库，使用ctypes是很好的选择，有一些基准测试下，python2+ctypes是性能最好的方式。

**Cython**: Cython是CPython的超集，用于简化编写C扩展的过程。Cython的优点是语法简洁，可以很好地兼容numpy等包含大量C扩展的库。Cython的使得场景一般是针对项目中某个算法或过程的优化。在某些测试中，可以有几百倍的性能提升。

cffi: cffi的就是ctypes在pypy（详见下文）中的实现，同进也兼容CPython。cffi提供了在python使用C类库的方式，可以直接在python代码中编写C代码，同时支持链接到已有的C类库。

使用这些优化方式一般是针对已有项目性能瓶颈模块的优化，可以在少量改动原有项目的情况下大幅度地提高整个程序的运行效率。

**并行编程**

因为GIL的存在，Python很难充分利用多核CPU的优势。但是，可以通过内置的模块multiprocessing实现下面几种并行模式：

**多进程**：对于CPU密集型的程序，可以使用multiprocessing的Process,Pool等封装好的类，通过多进程的方式实现并行计算。但是因为进程中的通信成本比较大，对于进程之间需要大量数据交互的程序效率未必有大的提高。

多线程：对于IO密集型的程序，multiprocessing.dummy模块使用multiprocessing的接口封装threading，使得多线程编程也变得非常轻松(比如可以使用Pool的map接口，简洁高效)。

分布式：multiprocessing中的Managers类提供了可以在不同进程之共享数据的方式，可以在此基础上开发出分布式的程序。

不同的业务场景可以选择其中的一种或几种的组合实现程序性能的优化。

**使用性能分析工具**

除了上面在ipython使用到的timeit模块，还有cProfile。

代码优化能够让程序运行更快，它是在不改变程序运行结果的情况下使得程序的运行效率更高，根据 80/20 原则，实现程序的重构、优化、扩展以及文档相关的事情通常需要消耗 80% 的工作量。优化通常包含两方面的内容：减小代码的体积，提高代码的运行效率。

改进算法，选择合适的数据结构

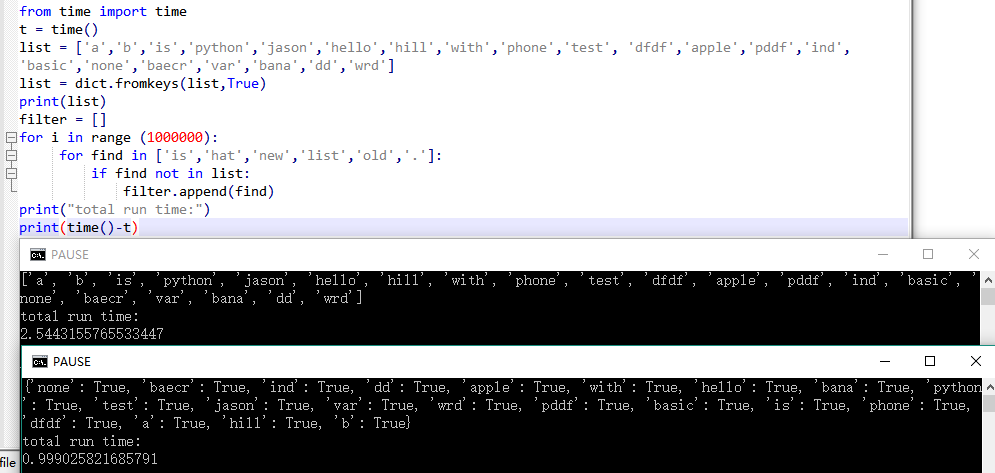
一个良好的算法能够对性能起到关键作用，因此性能改进的首要点是对算法的改进。在算法的时间复杂度排序上依次是：

O(1) -> O(lg n) -> O(n lg n) -> O(n^2) -> O(n^3) -> O(n^k) -> O(k^n) -> O(n!)

因此如果能够在时间复杂度上对算法进行一定的改进，对性能的提高不言而喻。但对具体算法的改进不属于本文讨论的范围，读者可以自行参考这方面资料。下面的内容将集中讨论数据结构的选择。

字典 (dictionary) 与列表 (list)

Python 字典中使用了 hash table，因此查找操作的复杂度为 O(1)，而 list 实际是个数组，在 list 中，查找需要遍历整个 list，其复杂度为 O(n)，因此对成员的查找访问等操作字典要比 list 更快。



如果去掉行 #list = dict.fromkeys(list,True) 的注释，将 list 转换为字典之后再运行，效率大概提高很多。因此在需要多数据成员进行频繁的查找或者访问的时候，使用 dict 而不是 list 是一个较好的选择。

对循环的优化

对循环的优化所遵循的原则是尽量减少循环过程中的计算量，有多重循环的尽量将内层的计算提到上一层。 下面通过实例来对比循环优化后所带来的性能的提高。

字符串的优化

python 中的字符串对象是不可改变的，因此对任何字符串的操作如拼接，修改等都将产生一个新的字符串对象，而不是基于原字符串，因此这种持续的 copy 会在一定程度上影响 python 的性能。对字符串的优化也是改善性能的一个重要的方面，特别是在处理文本较多的情况下。字符串的优化主要集中在以下几个方面：

在字符串连接的使用尽量使用 join() 而不是 +：在代码清单 7 中使用 + 进行字符串连接大概需要 0.125 s，而使用 join 缩短为 0.016s。因此在字符的操作上 join 比 + 要快，因此要尽量使用 join 而不是 +。

当对字符串可以使用正则表达式或者内置函数来处理的时候，选择内置函数。如 str.isalpha()，str.isdigit()，str.startswith(('x', 'yz'))，str.endswith(('x', 'yz'))

对字符进行格式化比直接串联读取要快，因此要使用，

out = "<html>%s%s%s%s</html>" % (head, prologue, query, tail)

而避免，

out = "<html>" + head + prologue + query + tail + "</html>"

如果需要交换两个变量的值使用 a,b=b,a 而不是借助中间变量 t=a;a=b;b=t。

# 参考

## 主要参考

[廖雪峰Python教程](http://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000)；

[信息管理专业Python教程](https://fanwscu.gitbooks.io/py4inf-zh-cn/content/)；

[Python教程](http://www.runoob.com/python/python-tutorial.html)；

[Python3教程](http://www.runoob.com/python3/python3-tutorial.html)；

[Python3.5.2教程](http://docs.pythontab.com/python/python3.5/)；

[Pyqt5教程1](http://zetcode.com/gui/pyqt5/)；

[Pyqt5教程2](https://my.oschina.net/wisedream/blog/536052)；

## 次要参考

[Python格式化输出](http://www.cnblogs.com/plwang1990/p/3757549.html)；

[Python中使用代码：以Numpy为例](https://segmentfault.com/a/1190000000479951)；

[与C进行交互](http://www.kancloud.cn/wizardforcel/scipy-lecture-notes/129876)；

[Python性能优化技巧](http://blog.konghy.cn/2016/09/26/python-optimize)；

[加速你的Python代码](http://uwebs.tk/archives/2013/03/25/speeding-up-your-python-code.html)；

[用C/C++拓展Python](https://blog.brickgao.com/2015/08/14/extending-Python-with-C-and-C-Plus-Plus/)；

[Python2与Python3的差异对比](https://my.oschina.net/leejun2005/blog/173553)；

[Python性能优化的20条建议](https://segmentfault.com/a/1190000000666603)；

[Python代码性能优化技巧](https://www.ibm.com/developerworks/cn/linux/l-cn-python-optim/)；

[Python性能优化技巧](http://kuanghy.github.io/2016/09/26/python-optimize)；

[Python性能优化指导](http://jinzhao.me/archives/274)；

[Python性能优化](http://blog.wongxinjie.com/2015/10/04/Python%E6%80%A7%E8%83%BD%E4%BC%98%E5%8C%96/)；