macports.org/。首先必须下载MacPorts，最好的方法就是下载正确的.dmg文件。在该站点选择当前版本的Mac OS X系统对应的.dmg文件。下载完成后进行安装，MacPorts安装完毕后打开一个新的终端窗口，输入以下命令：

```
>sudo port install py27-matplotlib
```

这会同时启动Python、NumPy和Matplotlib的安装。根据机器配置和网络速度的不同，安装时间也不尽相同，如果安装过程持续一小时也都是正常的。

当然，如果读者不想安装MacPorts，也可以分别安装Python、NumPy和Matplotlib。它们均有Mac OS X版本的二进制安装软件包，安装起来也很方便。

A.1.3　Linux

在Debian/Ubuntu系统下安装Python、NumPy和Matplotlib的最佳方式是使用apt-get或者其他发布版本中相应的软件包管理器。安装Matplotlib时会检查其依赖组件是否已经安装。由于Matplotlib依赖于Python和NumPy，因此安装Matplotlib时要确保已经安装了Python和NumPy。

要安装Matplotlib，打开命令行输入以下命令：

```
>sudo apt-get install python-matplotlib
```

根据机器配置和网络速度的不同，安装时间也不相同，整个安装过程需要花费一些时间。

至此Python已经安装完毕，下节将介绍Python中的几种数据类型。

A.2　Python 入门

下面介绍本书中用到的Python功能。本书不对Python做详尽的描述，如果读者有兴趣，推荐阅读Elkner、Downey和Meyers的在线免费资料：“How to Think Like a Computer Scientist”（http://openbookproject.net/thinkCS/）。本节还将介绍*容器类型*（collection type）和*控制结构*（control structure）。几乎每种编程语言都有类似的功能，这里着重给出它们在Python中的用法。本节最后介绍了*列表推导式*（list comprehension），这是初学Python时最容易感到困惑的部分。

A.2.1　容器类型

Python提供多种数据类型来存放数据项集合。此外，用户还可以通过添加模块创建出更多容器类型。下面列出了几个Python中常用的容器。

(1) 列表（List）—— 列表是Python中存放有序对象的容器，可以容纳任何数据类型：数值、布尔型、字符串等等。列表一般用两个括号来表示，下面的代码演示了如何创建一个名为`jj`的列表，并在列表内添加一个整数和一个字符串：

```
>>> jj=[]
>>> jj.append(1)
>>> jj.append('nice hat')
>>> jj
[1, 'nice hat']
```

当然，还可以把元素直接放在列表里。例如，上述列表jj还可以用下面的语句一次性构建出来：

```
>>> jj = [1, 'nice hat']
```

与其他编程语言类似，Python中也有数组数据类型。但数组中仅能存放同一种类型的数据，在循环的时候它的性能优于列表。为避免跟NumPy中的数组产生混淆，本书将不会使用该结构。

(2) 字典（Dictionary）—— 字典是一个存放无序的键值映射（key/value）类型数据的容器，键的类型可以是数字或者字符串。在其他编程语言中，字典一般被称为关联数组（associative array）或者映射（map）。下面的命令创建了一个字典并在其中加入了两个元素：

```
>>> jj={}
>>> jj['dog']='dalmatian'
>>> jj[1]=42
>>> jj
{1: 42, 'dog': 'dalmatian'}
```

同样，也可以用一条命令来完成上述功能：

```
>>> jj = {1: 42, 'dog': 'dalmatian'}
```

(3) 集合（Set）—— 这里的集合与数学中集合的概念类似，是指由不同元素组成的合集。下面的命令可以从列表中创建一个集合来：

```
>>> a=[1, 2, 2, 2, 4, 5, 5]
>>> sA=set(a)
>>> sA
set([1, 2, 4, 5])
```

集合支持一些数学运算，例如并集、交集和补集。并集用管道的符号（|）来表示，交集用&符号来表示。

```
>>> sB=set([4, 5, 6, 7])
>>> sB
set([4, 5, 6, 7])
>>> sA-sB
set([1, 2])
>>> sA | sB
set([1, 2, 4, 5, 6, 7])
>>> sA & sB
set([4, 5])
```

A.2.2　控制结构

Python里的缩进非常重要，这点也引来了也不少人的抱怨，但严格的缩进也能迫使编程人员写出干净、可读性强的代码。在for循环、while循环或者是if语句中，缩进用来标识出哪一段代码属于本循环。这里的缩进可以采用空格或者制表符（tab）来完成。而在其他的编程语言中，

一般使用大括号{ }或者关键字来实现这一点。所以通过使用缩进来代替括号，Python还节省了不少代码空间。下面来看一些常用控制语句的写法：

(1) If——if语句非常直观，可以在一行内完成：

```
>>> if jj < 3:  print "it's less than three man"
```

也可以写成多行，使用缩进来告诉编译器本语句尚未完成。这两种格式都是可以的。

```
>>> if jj < 3:
...     print "it's less than three man"
...     jj = jj + 1
```

多条件语句的关键字else if在Python中写做elif，而else在Python中仍写做else。

```
>>> if jj < 3: jj+=1
... elif jj==3: jj+=0
... else: jj = 0
```

(2) For——Python中的for循环与Java或C++0x[①]中的增强的for循环类似，它的意思是用for循环遍历集合中的每个元素。下面分别以列表、集合和字典为例来介绍for循环的用法：

```
>>> sB=set([4, 5, 6, 7])
>>> for item in sB:
...     print item
...
4
5
6
7
```

下面遍历一部字典：

```
>>> jj={'dog': 'dalmatian', 1: 45}
>>> for item in jj:
...     print item, jj[item]
...
1 45
dog dalmatian
```

可以看到，字典中的元素会按键值大小顺序遍历。

A.2.3　列表推导式

新手接触到Python最容易困惑的地方之一就是列表推导式。列表推导式用较为优雅的方式生成列表，从而避免大量的冗余代码。但语法有点别扭，下面先看一下实际效果然后再做讨论：

```
>>> a=[1, 2, 2, 2, 4, 5, 5]
>>> myList = [item*4 for item in a]
>>> myList
[4, 8, 8, 8, 16, 20, 20]
```

① C++0x，后来也称为C++11，即ISO/IEC 14882:2011，是目前的C++编程语言的正式标准。它取代第二版标准ISO/IEC 14882:2003(第一版ISO/IEC 14882:1998公开于1998年，第二版于2003年更新，分别通称C++98以及C++03，两者差异很小)。——译者注

列表推导式总是放在括号中。上述代码等价于：

```
>>> myList=[]
>>> for item in a:
...     myList.append(item*4)
...
>>> myList
[4, 8, 8, 8, 16, 20, 20]
```

可以看到，得到的myList结果是一样的，但列表推导式所需的代码更少。可能造成困惑的原因是加入到列表的元素在for循环的前面。这点违背了从左到右的文本阅读方式。

下面来看一个更高级的列表推导式的用法。如果只想保留大于2的元素：

```
>>> [item*4 for item in a if item>2]
[16, 20, 20]
```

用列表推导式可以写出更有创意的代码，当然如果代码很难被读懂，应尽量实现出更好的高可读性的代码。回顾完这些基础知识之后，下一节将介绍如何安装本书用到的各种Python模块。

对大多数纯Python模块（没有和其他语言的绑定模块）来说，直接进入代码的解压目录，输入>python setup.py install即可安装。这是默认的安装方式，如果对模块安装方法不太明确时，可以尝试一下上述命令。Python将把这些模块安装在Python主目录下的Libs\site-packages子目录里，因此不必担心模块究竟被安装到哪个地方或者清空下载目录会不会把它删掉。

A.3　NumPy 快速入门

NumPy库安装完成后，读者可能在想：“这东西有什么好处？”正式来说，NumPy是Python的一个矩阵类型，提供了大量矩阵处理的函数。非正式来说，它是一个使运算更容易、执行更迅速的库，因为它的内部运算是通过C语言而不是Python实现的。

尽管声称是一个关于矩阵的库，NumPy实际上包含了两种基本的数据类型：数组和矩阵。二者在处理上稍有不同。如果读者熟悉MATLAB™的话，矩阵的处理将不是难事。在使用标准的Python时，处理这两种数据类型均需要循环语句。而在使用NumPy时则可以省去这些语句。下面是数组处理的一些例子：

```
>>> from numpy import array
>>> mm=array((1, 1, 1))
>>> pp=array((1, 2, 3))
>>> pp+mm
array([2, 3, 4])
```

而如果只用常规Python的话，完成上述功能需要使用for循环。

另外在Python中还有其他一些需要循环的处理过程，例如在每个元素上乘以常量2，而在NumPy下就可以写成：

```
>>> pp*2
array([2, 4, 6])
```

还有对每个元素平方：

```
>>> pp**2
array([1, 4, 9])
```

可以像列表中一样访问数组里的元素：

```
>>> pp[1]
2
```

NumPy中也支持多维数组：

```
>>> jj = array([[1, 2, 3], [1, 1, 1]])
```

多维数组中的元素也可以像列表中一样访问：

```
>>> jj[0]
array([1, 2, 3])
>>> jj[0][1]
2
```

也可以用矩阵方式访问：

```
>>> jj[0,1]
2
```

当把两个数组乘起来的时候，两个数组的元素将对应相乘：

```
>>> a1=array([1, 2,3])
>>> a2=array([0.3, 0.2, 0.3])
>>> a1*a2
array([ 0.3,  0.4,  0.9])
```

下面来介绍矩阵。

与使用数组一样，需要从NumPy中导入matrix或者mat模块：

```
>>> from numpy import mat, matrix
```

上述NumPy中的关键字mat是matrix的缩写。

```
>>> ss = mat([1, 2, 3])
>>> ss
matrix([[1, 2, 3]])
>>> mm = matrix([1, 2, 3])
>>> mm
matrix([[1, 2, 3]])
```

可以访问矩阵中的单个元素：

```
>>> mm[0, 1]
2
```

可以把Python列表转换成NumPy矩阵：

```
>>> pyList = [5, 11, 1605]
>>> mat(pyList)
matrix([[   5,   11, 1605]])
```

现在试试将上述两个矩阵相乘：

```
>>> mm*ss
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "c:\Python27\lib\site-packages\numpy\matrixlib\defmatrix.py",
line 330, i
```

```
n __mul__
    return N.dot(self, asmatrix(other))
ValueError: objects are not aligned
```

可以看到出现了一个错误：乘法不能执行。矩阵数据类型的运算会强制执行数学中的矩阵运算，1×3的矩阵是不能与1×3的矩阵相乘的（左矩阵的列数和右矩阵的行数必须相等）。这时需要将其中一个矩阵转置，使得可以用3×1的矩阵乘以1×3的矩阵，或者是1×3的矩阵乘以3×1的矩阵。NumPy数据类型有一个转置方法，因此可以很方便地进行矩阵乘法运算：

```
>>> mm*ss.T
matrix([[14]])
```

这里调用了.T方法完成了ss的转置。

知道矩阵的大小有助于上述对齐错误的调试，可以通过NumPy中的shape方法来查看矩阵或者数组的维数：

```
>>> from numpy import shape
>>> shape(mm)
(1, 3)
```

如果需要把矩阵mm的每个元素和矩阵ss的每个元素对应相乘应该怎么办呢？这就是所谓的元素相乘法，可以使用NumPy的multiply方法：

```
>>> from numpy import multiply
>>> multiply(mm, ss)
matrix([[1, 4, 9]])
```

此外，矩阵和数组还有很多有用的方法，如排序：

```
>>> mm.sort()
>>> mm
matrix([[1, 2, 3]])
```

注意该方法是原地排序（即排序后的结果占用原始的存储空间），所以如果希望保留数据的原序，必须事先做一份拷贝。也可以使用argsort()方法得到矩阵中每个元素的排序序号：

```
>>> dd=mat([4, 5, 1])
>>> dd.argsort()
matrix([[2, 0, 1]])
```

可以计算矩阵的均值：

```
>>> dd.mean()
3.3333333333333335
```

再回顾一下多维数组：

```
>>> jj = mat([[1, 2, 3,], [8, 8, 8]])
>>> shape(jj)
(2, 3)
```

这是一个2×3的矩阵，如果想取出其中一行的元素，可以使用冒号（:）操作符和行号来完成。例如，要取出第一行元素，应该输入：

```
>>> jj[1,:]
matrix([[8, 8, 8]])
```

还可以指定要取出元素的范围。如果想得到第一行第0列和第1列的元素，可以使用下面的语句：

```
>>> jj[1,0:2]
matrix([[8, 8]])
```

这种索引方法能够简化NumPy的编程。 在数组和矩阵数据类型之外，NumPy还提供了很多其他有用的方法。我建议读者浏览完整的官方文档http://docs.scipy.org/doc/。

A.4　Beautiful Soup包

本书使用Beautiful Soup包来查找和解析HTML。要安装Beautiful Soup，请下载对应的模块：http://www.crummy.com/software/BeautifulSoup/#Download。

下载之后解压，进入解压目录，输入下面的命令：

```
>python setup.py install
```

如果在Linux下没有权限安装的话，使用以下命令：

```
>sudo python setup.py install
```

Python的模块大多都是这样安装的，记得阅读每个模块自带的`README.txt`文件。

A.5　Mrjob

Mrjob用于在Amazon网络服务上启动MapReduce作业。安装mrjob与Python中其他模块一样方便：打开https://github.com/Yelp/mrjob，在页面左边可以看到“ZIP”按钮，点击该按钮下载最新的版本。用unzip和untar解压文件，进入到解压目录后在Python提示符下输入：

```
>python setup.py install
```

GitHub已经列出了很多代码的样例。此外还有一个不错的网站http://packages.python.org/mrjob/ 也提供了一些Python的官方文档。

在AWS上正式使用mrjob之前，需要设置两个环境变量： `$AWS_ACCESS_KEY_ID`和`$AWS_SECRET_ACCESS_KEY`。它们的值应该设置成你的账号（如果你拥有账号的话），该账号信息可以在登陆AWS后，在Account > Security Credentials页面看到。

下面来设定一下这些环境变量，打开命令行提示符，输入以下命令：

```
>set AWS_ACCESS_KEY_ID=1269696969696969
```

验证一下是否有效：

```
>echo %AWS_ACCESS_KEY_ID%
```

同样的方法可以完成`AWS_SECRET_ACCESS_KEY`的设置。

如果要在Mac OS X上设置这些环境变量，打开终端窗口（新版本的OS X使用bash命令行），输入以下命令：

```
>AWS_ACCESS_KEY_ID=1269696969696969
>export AWS_ACCESS_KEY_ID
```

同样的方法可以完成`AWS_SECRET_ACCESS_KEY`的设置，注意字符串不需要引号。Ubuntu Linux