统计数据分析与应用丛书

Statistics

Python
——统计人的视角

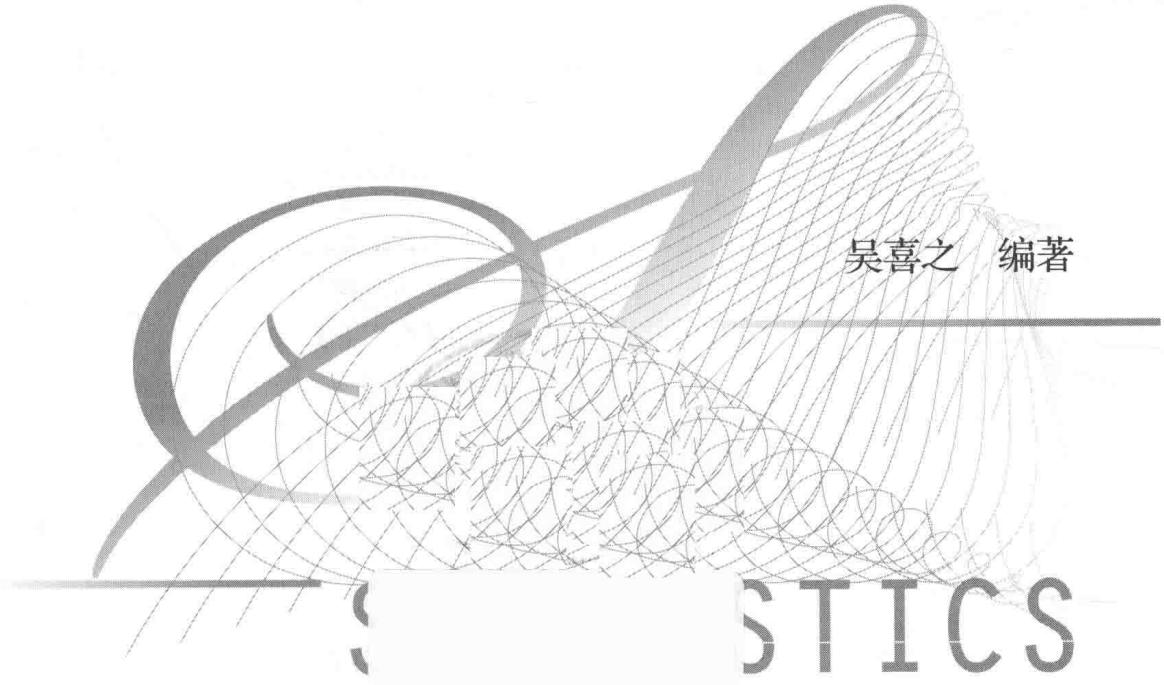
吴喜之 编著

STATISTICS

中国人民大学出版社

统计数据分析与应用丛书 Statistics

Python
——统计人的视角



吴喜之 编著

中国人民大学出版社
· 北京 ·

图书在版编目（CIP）数据

Python：统计人的视角 / 吴喜之编著. — 北京 ：中国人民大学出版社，2018.1
（统计数据分析与应用丛书）
ISBN 978-7-300-25216-2

Ⅰ.①P… Ⅱ.①吴… Ⅲ.①软件工具-程序设计 Ⅳ.①TP311.561

中国版本图书馆 CIP 数据核字（2017）第 300685 号

统计数据分析与应用丛书
Python——统计人的视角
吴喜之 编著
Python：Tongjiren de Shijiao

出版发行	中国人民大学出版社		
社　　址	北京中关村大街 31 号	**邮政编码**	100080
电　　话	010－62511242（总编室）	010－62511770（质管部）	
	010－82501766（邮购部）	010－62514148（门市部）	
	010－62515195（发行公司）	010－62515275（盗版举报）	
网　　址	http://www.crup.com.cn		
	http://www.ttrnet.com（人大教研网）		
经　　销	新华书店		
印　　刷	北京昌联印刷有限公司		
规　　格	185 mm×260 mm　16 开本	**版　　次**	2018 年 1 月第 1 版
印　　张	8 插页 1	**印　　次**	2018 年 1 月第 1 次印刷
字　　数	180 000	**定　　价**	36.00 元

前　言

Python 是一款非常优秀的通用软件; 它免费、开源; 它的模块数目有几万个, 而且还在飞速增长. Python 是目前几乎所有的知识探索及应用领域最重要的软件工具之一.

各个领域对 Python 的广泛需求产生了很多关于 Python 的教材. 但是, 由于 Python 的应用领域太多, 不同领域对 Python 语言的需求大相径庭, 每本书可能仅适应于某一类读者. 本书面对的是 (非计算机背景的) 统计、应用数学及数据分析方面的师生和实际工作者, 力图以最简单的方式让读者尽快地掌握 Python 的精髓.

本书旨在介绍计算机语言, 因此不应被看成是统计教材, 其中涉及的一些统计内容仅仅是学习 Python 的载体, 所以并不追求统计内容的完整和全面, 目的是向已经有些统计知识的人介绍 Python.

目前世界经济是被技术驱动的, 而拥有编程技能则是一种优势. 在科学、技术、工程等领域, 有过半的工作是由计算机完成的. 对能够编程的人才的需求远远超过供给. 学习编程不仅是社会需要, 而且**能够使人学会思考**.[①]

能不能迅速学会编程, 关键在于对其是否感兴趣. 当然, 从来没有写过程序的人不可能事先就感兴趣, 但人生绝大多数兴趣都是后天培养的. 对编程的爱好是在编程中培养的. **如果你能够把编程作为一种艺术来欣赏, 作为一种嗜好来实践, 那么你的目的就达到了.**

在大数据时代的数据分析, 最重要的不是掌握一两种编程语言, 而是拥有**泛型编程能力** (也是一种思维方式). 有了这种能力, 语言之间的不同不会造成太多的烦恼. Python 仅仅是一种编程语言, 但对于编程的初学者来说, 却是一个良好的开端.

关于 Python 和 R 的比较, 一些人说 Python 比 R 好学, 而另一些人正相反, 觉得 R 更易掌握. 其实, 对于熟悉编程语言的人来说, 学哪一个都很快. 它们的区别大体如下: 由于有统一的志愿团队管理, R 的语法相对比较一致, 安装程序包很简单, 而且很容易找到帮助和支持, 但由于 R 主要用于数据分析, 所以一些对于统计内容不那么熟悉的人可能觉得对象太专业了. Python 则是一款通用软件, 比 C++ 容易学, 功能并不差, 基于 Python 改进的诸如 Cython 那样的改进或包装版软件运行速度也非常快. 但是, Python 没有统一的团队管理, 针对不同 Python 版本的模块非常多. 因此对于不同的计算机操作系统、不同版本的 Python、不同的模块, 首先遇到的就是安装问题, 语法习惯也不尽相同. 另外, R 软件的基本语言 (即下载 R 之后所装的基本程序包) 本身就可以应付相当复杂的统计运算, 而相比之下 Python 的统计模型没有那么多, 做一些统计分析不如 R 那么方便, 但从其基本语法所产生的成千上万的模块使它几乎可以做任何想做的事情.

[①] Steve Jobs. Everybody in this country should learn to program a computer, because it teaches you how to think.

学习自然语言必须依靠实践，而不能从背单词和学习语法入手. 学习计算机语言也是一样，本书不采用详尽的使用手册式教学，而是让读者通过实践来学会编程语言. 当需要查找某些特定的定义或语法细节时，网络查询则是最好的途径.

吴喜之

目 录

第 6 章　类和子类简介

第 1 章　引　言

1.1　下载及安装 Python

可以从不同的平台来下载、安装和使用 Python. 根据笔者的经验, 这方面最好的教师是网络, 每种操作系统都有一些最适应的方式, 而且这些都随着操作系统的更新、平台的更新而不断变化. 笔者觉得对于初学者最方便的平台是 Anaconda. 只要进入网页 https://www.continuum.io/downloads, 就知道如何在各种操作系统 (Windows, macOS 及 Linux) 中安装 Anaconda. 目前 Python 有 3 及 2 两个版本 (本书使用 3 版本), 另外可以选 64-bit 和 32-bit. 安装之后, 一些最基本的模块, 比如 numpy, pandas, matplotlib, IPython, scipy 就都一并安装了. 此后, 可以以各种方式使用 Python, 比如通过 Jupyter Notebook, IPython, Spyder 等界面来运行. 从运算来说, 各种界面没有区别, 但不同人的习惯不同, 会有某些偏好.

笔者所常用的界面是 Jupyter Notebook, 觉得它对于初学者来说更加方便, 因为它把每一步程序及结果都自动记录下来. 而 Spyder 为交互式的环境, 提供了若干窗口分别编写程序文件、交互式输入代码、输出结果以及为读者提供帮助等, 有其好处, 只是除了文件编写窗口外, 不记录敲入的结果.

另外要注意, 诸如 Windows 和 Mac 的 OS 系统等都在不断升级, Anaconda 也在改变, 同时 Python 及各个模块还在不断升级, 本书所说的具体操作和代码会随之变化. 相信读者会不断适应这些变化, 与时俱进.

1.2　Anaconda 的几种界面

1.2.1　使用 Notebook

安装完 Anaconda 之后, 就可以运行 Notebook 了. 在 Windows 下打开 Notebook 有两种方法:

一种方法是在`CMD` 窗口 (即终端) 进入你的程序文件所在的目录. 如果还没有程序文件, 就事先产生一个新目录, 假定是 `D` 盘的 `D:/Python Work` 文件夹. 这样, 在 `CMD` 界面中敲入 `D:`, 回车后就到了`D` 盘; 然后键入 `cd Python Work` 即可到达你的工作目录; 再键入 `jupyter notebook`, 则在默认浏览器产生一个工作界面 (称为 "Home"). 如果你已经有文件, 则会有书本图标开头的列表, 你的文件名以 `.ipynb` 为扩展名. 如果没有现成的文件, 可创建新的文件, 点击右上角 `New` 并选择`Python3` (如果你用 Python3 的话), 则产生一个没有名字的 (默认是 Untitled) 以 `.ipynb` 为扩展名的文件 (自动存在你的工作目录中) 的页面, 文件名字可以随时任意更改.

另一种是在电脑的程序列表中寻找 Anaconda3, 点击后在子目录中找到 Jupyter Notebook, 点击即可得到默认浏览器产生的工作界面, 和上面一样.

在你的文件页中会出现 `In [ ]:` 标记, 可以在此输入代码, 然后得到的结果就出现在代码 (代码所在的部位称为 "cell") 的下面. 一个 cell 中可有一群代码, 可以在其上下增加 cell, 也可以合并或拆分 cell, 相信读者会很快掌握这些小技巧.

在 Mac 机的 OS 系统可以用 terminal 进入 Python 界面, 如同在 Windows 系统一样, 先用类似于 `cd Python Work` 的命令进入工作目录, 然后用 `jupyter notebook` 进入 Anaconda. macOS 的 Anaconda 和 Windows 的没有多少区别. 在 Mac 机里面也可以通过在应用程序中寻找并点击 Anaconda-Navigator 的图标进入各种界面.

作为开始, 你可以先键入下面的代码:

```
3*' Python is easy!'
```

用`Ctrl+Enter` (不会产生新的 cell) 或者`Shift+Enter` (这会把光标移到下面新产生的输入 cell) 就输出

```
' Python is easy! Python is easy! Python is easy!'
```

实际上这一代码等价于 `print(3*' Python is easy!')`(在 Python2 中, 打印内容不一定非得放在圆括号中, 而在 Python3 中必须把打印内容放在圆括号中). 在一个 cell 中, 如果有可以输出的几条语句, 则只输出有 `print` 的行及最后一行代码 (无论有没有 `print`) 的结果.

在 Python 中, 也可以一行输入几个简单 (不分行的) 命令, 用分号分隔. 要注意, Python 和 R 的代码一样是分大小写的. Python 与 R 的注释一样, 在 # 号后面的符号不会被当成代码执行.

当前工作目录是在存取文件、输入输出模块时只敲入文件或模块名称而不用敲入路径的目录. 查看目前的工作目录和改变工作目录的代码为:

```
import os
print(os.getcwd()) #查看目前的工作目录
os.chdir('D:/Python work') #改变工作目录
```

1.2.2 使用 Spyder

在程序中寻找 Anaconda3, 点击后在子目录中找到 Spyder, 点击即可得到界面 (见图 1.1). 图中的界面是默认界面, 完全可以改变界面的布局、窗口的增减和大小, 图 1.1 左侧是编辑 py 文件的编辑器, 右下窗口是输入代码及得到输出结果的交互式 Console 界面, 右上窗口可以查询帮助或做其他用途.

在 Mac 机中, 只要打开 terminal 并键入`spyder` 即可进入 Spyder 界面, 也可以在应用程序中寻找并点击程序 Anaconda, 并在其子目录中点击 Spyder 进入, 其余和 Windows

类似.

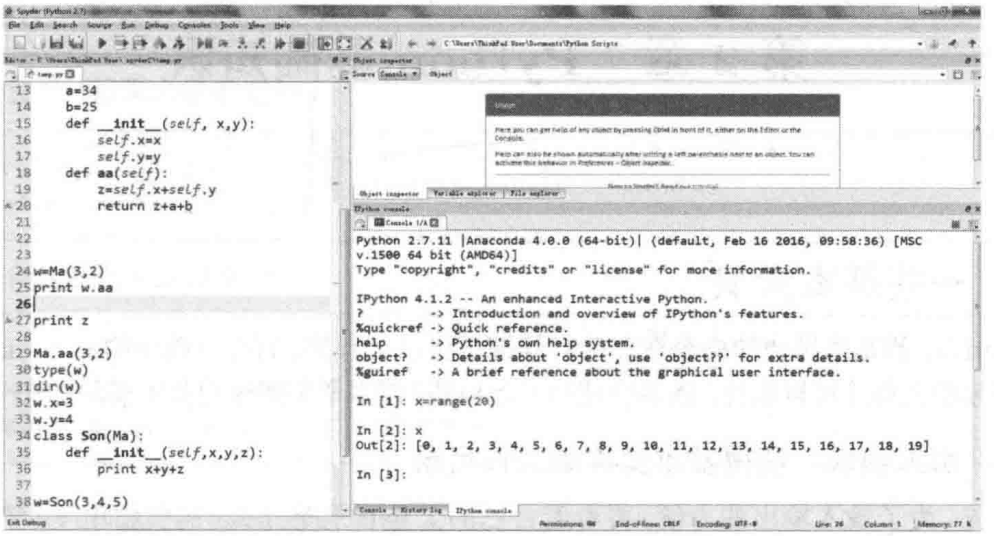


图 1.1 Spyder 界面

1.2.3 使用 IPython 或者终端界面

IPython 就是一个交互式 Console 界面. 因此和 Spyder 的 Console 界面等价. 在程序中寻找 Anaconda3, 点击后在子目录中找到 IPython, 点击即可得到界面.

也可以直接用终端 (比如在 Windows 中从 CMD 界面或从 Mac 中的 terminal 界面), 敲入 “Python” 或 “Python3” (依设定的命令) 直接使用, 但由于在 IPython 中每一个命令, 包括 Python 一些群命令的缩进等, 都要手动敲入, 不那么方便.

第 2 章　Python 基础知识

2.1　一些基本常识

Python 的基本模块的命令在任何时候都是可以运行的, 而有些模块的命令, 必须在输入那些模块之后才可以运行. 虽然会使用其他模块, 但本章主要使用基本模块的命令.

2.1.1　引入模块、获得及改变你的工作目录

首先, 为了输入输出的方便, 要看看自己的工作目录是什么, 需要使用 os 模块中的 getcwd() 函数:

```
import os
os.getcwd()
```

这就会得到目前的路径, 在这个路径下存取文件不用另外敲入路径. 由于 getcwd() 函数不属于基本模块, 前面必须加上 os 而成为 os.getcwd(). 如果使用语句 from os import * (这意味着 os 中的所有函数都放入了内存) 或者使用语句 from os import getcwd(这意味着 os 中的 getcwd 放入了内存), 则可以直接使用 getcwd(). 但在有很多来自不同模块的函数的情况下, 如果没有注明函数的来源, 人们可能会对这些函数产生混淆.

有些模块名字太长, 可以简化, 比如 import matplotlib.pyplot as plt, 那么 (如果只用 import matplotlib) 函数 matplotlib.pyplot.subplot 可以简化成 plt.subplot.

如果要改变你的工作目录 (比如到 'D:/work'), 则可以用下面语句:

```
import os
os.chcwd('D:/work')   #或者os.chcwd('D:\\work')
```

命令可以直接在交互式 Console 界面输入, 也可以写在以 ".py" 结尾的文件中 (比如 "test.py"), 然后用命令 (如果该文件在工作目录中, 则不用写路径) import test 来执行文件 (test.py) 中的所有代码.

2.1.2　目录的建立和删除, 文件的重命名及删除

下面是 (在工作目录中) 建立新目录、删除已有目录或文件及对文件重命名的例子:

1. 建立新目录

```
import os
```

```
os.mkdir('work2')
```

2. 删除目录 (目录必须是空的)

```
import os
os.rmdir('work2')
```

3. 对文件重命名和删除文件

```
import os
os.rename('fff.txt','fool.txt')  #重命名
os.remove('h.txt')               #删除文件
```

2.1.3 list(列表)、循环语句和缩入

list(列表) 直观上显示为由方括号所包含的元素, 下面输入的是基本命令, 包括一个赋值语句, 一个循环打印语句:

```
x=[list(range(5)),75,"Python","is","great!",["Program is art"],abs(-2.34)]
for i in x:
    print(i)
```

得到

```
[0, 1, 2, 3, 4]
75
Python
is
great!
['Program is art']
2.34
```

这些命令都是基本模块的命令. 其中 x 的类型为 list (可用代码 type(x) 得到), list 的元素可以是字符 (str, 即用单引号或双引号标明的字符串, 比如 "Python")、数字, 也可以是 list, 比如 list (range(5)) (即由 0,1,$\cdots$,4 组成的 5 个从 0 开始的整数) 和["Program is art"](由单独字符串组成的列表). 而代码 for i in x: 则对 x 中的每个元素 (临时用 i 代表) 做后面的操作, 由于这是一系列 (这里 x 有 7 个元素) 打印 (print) 操作, 因此在冒号 (:) 下面的一行必须缩进 (这里是 4 个空格, 所有的缩进必须统一). 除了for 之外, 其他循环 (loop) 都要求这种缩进, 比如 while, if, elseif, else 等, 在函数或类的定义中也需要这种缩进. 在某些语言中用括号表示 (比如 R 中用花括号表示) 这种批处理环境.

函数 range 是个很常用的产生数值的函数, 变元可以是 1 个整数 n (为从 0 开

始到 n-1 的间隔为 1 的自然数列)、2 个整数 (n,p) (为从 n 开始到 p-1 的自然数列)、3 个整数 (n,p,r) (为从 n 开始到 p-1 的间隔为 r 的自然数列). 因此, range(5) 和range(0,5) 及range(0,5,1) 是等价的. 注意, 在 Python2 中range(x) 命令本身就可以输出数值, 而在 Python3 中则不行. 比如, 在 Python3 中单独运行range(5) 只会输出range(0,5), 要输出数值则需要把它转换成list, 也就是使用list(range(5)) 才能输出结果[0, 1, 2, 3, 4]. 请运行下面的代码并查看结果.

```
print(list(range(-1,11,2)), list(range(2,7)), list(range(10,-10,-3)))
```

由于 x 中的每个元素还有若干元素, 可以试试输入下面的命令看输出是什么 (这里有两层缩进).

```
x=[list(range(5)),"Python","is","great!",["Program is art"]]
for i in x:
    for j in i:
        print(j)
```

2.1.4 下标

在list、数列、矩阵或者字符串中, 都会有很多元素, 而每个元素或元素集合都可用下标表示. 熟悉某些软件 (比如 R) 的一些人不习惯 Python 的下标从 0 开始 (第 0 个元素), 而且下标区间都是半开区间 (右边是开区间), 比如, x[:3] 代表 x 的第 0, 1, 2 等三个元素; x[7:] 代表 x 的第 7 个 (包含第 7 个) 以后的元素; x[3:6] 代表 x 的第 3, 4, 5 个 (不包含第 6 个) 元素. 例如, 输入下面的代码:

```
x=list(range(10))  #=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(x[:3],x[7:],x[3:6],x[-3:],x[-1],x[:-4])
```

得到

```
[0, 1, 2] [7, 8, 9] [3, 4, 5] [7, 8, 9] 9 [0, 1, 2, 3, 4, 5]
```

注意: 下标 [-1] 表示最后一个, [-3:] 表示从倒数第 3 个开始往后的所有元素, [:-4] 表示从倒数第 4 个开始 (不包括倒数第 4 个) 往前的所有元素.

使用从 0 开始的下标以及半开区间有方便的地方, 比如下标 [:3] 实际上是 0,1,2, 类似地, [3:7] 是 3,4,5,6, 这样, [:3], [3:7], [7:10] 首尾相接, 实际上覆盖了从 0 到 9 的所有下标, 而在 R 中, 这种下标必须写成 [1:2], [3:6], [7:10], 由于是闭区间, 中间的端点不能重合. 请试运行下面的语句, 一些首尾相接的下标区间得到完整的下标群:

```
x='A poet can survive everything but a misprint.'
x[:10]+x[10:20]+x[20:30]+x[30:40]+x[40:]
```

得到完整的句子:

```
'A poet can survive everything but a misprint.'
```

如果对象的元素本身有多个元素, 也可以用复合下标来表示感兴趣的部分 (这一点和 R 类似).

```
y=[[1,2],[1,15,3],['Success','is','a','science']]
print(y[0],y[1][1:2],y[2][:2]+y[2][2:],y[2][0][:5],y[2][0][:5][0])
```

输出为:

```
[1, 2] [15] ['Success', 'is', 'a', 'science'] Succe S
```

其中, y[0] 就是 y 的第 0 个元素 [1,2], y[1][1:2] 就是 y[1]([1,15,3]) 的第 1 个元素 15, y[2][:2]+y[2][2:] 等于 y[2] 的第 0,1 个元素以及 y[2] 的第 2 个 (包括第 2 个) 之后的所有元素 (也就组成了 y[2] 的所有元素), y[2][0][:5] 是 y[2][0]('Success') 的前面 0 ~ 4 个元素 (即 " Succe"), 而 y[2][0][:5][0] 是 Succe 的第 0 个元素 " S".

2.1.5 list 中元素增减所用的函数

1. 函数 append

给一个 list 增加一个元素可执行下面的代码:

```
x=[[3,5,7],'Oscar Wilde']
y=['save','the world',['is','impossible']]
x.append(y);print(x)
```

得到

```
[[3, 5, 7], 'Oscar Wilde', ['save', 'the world', ['is', 'impossible']]]
```

显然, 这里的 x.append(y) 是把 list y 整体作为一个元素加入到 x 中.

2. 函数 extend

在一个 list 中加入另外一个 list 的元素可执行下面的代码:

```
x=[[3,5,7],'Oscar Wilde']
y=['save','the world',['is','impossible']]
x.extend(y);print(x)
```

得到

```
[[3, 5, 7], 'Oscar Wilde', 'save', 'the world', ['is', 'impossible']]
```

这里的 x.extend(y) 是把 list y 中的元素个体 (但不拆开 y 中作为 list 的个体) 加入到 x 中.

3. 函数 pop

这是按照下标来删除 list 元素的函数. 请执行下面的代码:

```
x=[[1,2],'Word',[3,5,7],'Oscar Wilde']
x.pop();print(x)  #去掉最后一个
x=[[1,2],'Word',[3,5,7],'Oscar Wilde']
x.pop(2);print(x) #去掉下标为2的元素(即[3,5,7])
```

得到

```
[[1, 2], 'Word', [3, 5, 7]]
[[1, 2], 'Word', 'Oscar Wilde']
```

这里的 x.pop() 去掉 x 中的最后一个元素, 而 x.pop(2) 则去掉 x 中的第 2 个元素 (这里删除的是 [3, 5, 7]).

4. 函数 remove

这是按照内容来删除 list 元素的函数. 请执行下面的代码:

```
x=[[1,2],'Word',[3,5,7],'Oscar Wilde',[3,5,7]]
x.remove([3,5,7]);print(x)
x.remove([3,5,7]);print(x)
x.remove('Word');print(x)
```

得到

```
[[1, 2], 'Word', 'Oscar Wilde', [3, 5, 7]]
[[1, 2], 'Word', 'Oscar Wilde']
[[1, 2], 'Oscar Wilde']
```

这里的 x.remove([3,5,7]) 是把 x 中的 [3,5,7] 去掉, 但如果有重复的内容, 每次仅仅去掉下标最小的一个.

2.1.6 数量变量的类型及运算

和其他语言一样, Python 的变量有不同的类型, 变量的类型用代码 type() 显示. 各种不同类型变量之间的运算有一定的规则. 特别要注意的是数量变量运算中整型变量 (int) 和浮点型变量 (float) 的运算规则. 请运行下面进行算术运算并显示结果类型的语句:

```
print(type(2))
print(type(2.5))
x=2+2.5;print(x,type(x))
x=20/3;print(x,type(x))
x=20/3.;print(x,type(x))
x=20**3.;print(x,type(x))
y=3;x=20/float(y);print(x,type(x))
```

得到

```
<type 'int'>
<type 'float'>
4.5 <type 'float'>
6 <type 'int'>
6.66666666667 <type 'float'>
8000.0 <type 'float'>
6.66666666667 <type 'float'>
```

这里显示了两种数值类型: 整型和浮点型. 而整型数值与整型数值运算, 结果也是整型数值, 比如上面 20/3, 得到 6. 但整型数值与浮点型数值运算, 结果是浮点型数值, 比如上面 20/3., 得到浮点值 6.66666666667.

对于 list 中单个元素的运算, 整型数值和浮点型数值的运算规则也是一样的, 但注意, list 不能直接 (如在 R 中那样) 做通常数学中的向量运算, 比如 [1,3]*2 得到 [1, 3, 1, 3], 而不是想象的 [2, 6]. 对数组的元素逐个做数值运算, 一般人都使用后面将要介绍的诸如 numpy 或 scipy 那样的模块. 但也可以用其他方式对 list 之类的数组元素做一些数值运算, 比如可用下列方式对数组 list 的每个元素做加、减、乘、除、乘方运算:

```
[x/4.+5*x**2-20 for x in [210,14,-5,9]]
```

得到 4 元素数组 [210,14,-5,9] 中每个元素 x 对应的 $x/4+5x^2-20$ 的值 (输出 4 个结果值):

```
[220532.5, 963.5, 103.75, 387.25]
```

2.1.7 tuple (多元组), dict (字典)

tuple 和 list 非常类似 (一个用圆括号表示, 一个用方括号表示), 它们的一个主要区别在于 tuple 不能增减或更改其元素, 而 dict (dictionary) 则是有索引的多元组 (用花括号表示), 有其方便的地方.

关于 tuple, 看下面的代码:

```
x=(0,[12,345,67],9,'we','they')
print(x[-3:])
print(type(x),x[-5:-1])
for i in x:
    print(i)
```

得到

```
(9, 'we', 'they')
<type 'tuple'> (0, [12, 345, 67], 9, 'we')
0
[12, 345, 67]
9
we
they
```

我们可以用 `list(x)` 来把变量 `x` 转换成 `list` 类型, 比如代码

```
x=(2,"I am OK",[3,2,9])
print(x,x[1],type(x))
x1=list(x)
print(x1,x1[1],type(x1))
```

得到

```
(2, 'I am OK', [3, 2, 9]) I am OK <class 'tuple'>
[2, 'I am OK', [3, 2, 9]] I am OK <class 'list'>
```

关于 `dict`, 看下面的代码:

```
data={'age': 34, 'Children' : [1,2], 1: 'apple'}
print(type(data))
data['age']
data['age']='99'
data['name']='abc'
data[1]='orange'
print(data)
del data['age']
print(data)
```

得到