

Python 数据分析实践 (Data Analysis Action)

Chap 2 Python 基础

1. Python 基本数据结构

2. Python 基本语法

内容:

- 熟悉基础环境 ipython notebook 和 ipython
- 熟悉 Anaconda 的基本命令进行 install 和 update
 - Install package / Import package

实践:

- 基本数据结构 (列表/字符串/字典)
- 基本语法 (条件/循环)

常用模块的命名惯例

- `import numpy as np`
- `import pandas as pd`
- `import matplotlib.pyplot as plt`

当看到 `np.arange` 时, 就应该想到它引用的是 NumPy 中的 `arange` 函数。这样做的原因是: 在 Python 软件开发过程中, 不建议直接引入类似 NumPy 这种大型库的全部内容 (不建议 `from numpy import *`)

本书需要下面的包:

- Python 科学计算基础库: NumPy, SciPy, Pandas , 和绘图库 Matplotlib 等大多数常用库默认在 Anaconda 中安装
- 其他库如 Statsmodels, PyTables, Scikit, xlrd, lxml, BeautifulSoup, pymongo 以及 requests 等, 它们被用在不同示例中, 可以需要时再安装

安装库的命令

- `pip install PACKAGE-name`
- `conda install PACKAGE-name` (在Anaconda模式下)

Python 帮助手册

- Tab键的补全功能, 输入函数名的前面字符, 再按 Tab 键
- 在对象后面输入一个句点以便自动完成方法和属性的输入
- 调用 `help` 命令, `help()`

- 通过在函数名后面加上问号？进行查询。前提是要知道函数名，好处是不必输入 help 命令

In [1]:

```
# 导入两个常用的库，在导入库之前先安装库
# numpy 和 pandas 在 anaconda 中已经默认安装
import numpy as np
import pandas as pd
```

In [2]:

```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

Python 基础

1. Python 基本数据类型

计算机程序理所当然地可以处理各种数值，但是，计算机能处理的远不止数值，还可以处理文本、图形、音频、视频、网页等各种各样的数据。不同的数据，需要定义不同的数据类型。

Python 能够直接处理的数据类型有以下几种：

- 整数：int，有正负；十六进制表示的整数用 0x(零x) 前缀和 0-9，a-f 表示
- 浮点数：float，即小数，如 1.23，3.14，对于很大或很小的浮点数用科学计数法表示，把 10 用 e 替代， 1.23×10^9 就是 1.23e9 或 12.3e8，0.000012 可以写成 1.2e-5，等等。
- 字符串：str (注：\t 等于一个 tab 键)
- 布尔值：只有 True、False 两种值，注意大小写
- 空值：空值是 Python 里一个特殊的值，用 None 表示，不能理解为 0，因为 0 是有意义的，而 None 是一个特殊的空值。

在计算机程序中，变量不仅可以是数值，还可以是任意数据类型。变量在程序中用一个变量名表示，变量名必须是大小写英文、数字和下划线的组合，且不能用数字开头。

常量是不能变的变量，比如常用的数学圆周率 π 是一个常量。在 Python 中，通常用全部大写的变量名表示常量：PI=3.1415；但事实上 PI 仍然是一个变量，Python 没有任何机制保证 PI 不会被改变，所以，用全部大写的变量名表示常量只是一个习惯上的用法，如果一定要改变变量 PI 的值，没人能阻拦住。

**** 延伸阅读 ****

[如何让你的Python代码更加pythonic \(http://wuzhiwei.net/be_pythonic/\)](http://wuzhiwei.net/be_pythonic/)

In [3]:



```
# 基本数据类型
a = 3 # 变量a是个整数
#a = -0xff00 # 用十六进制表示的整数用0x(零x)前缀和0-9, a-f表示, 例如: 0xa5b4c3d2,
#a = -1.0 # 变量a是个浮点数
#a = 12.4e2 # 即1240, 科学计数法表示浮点数
#a = 12.4e-2 # 即0.124
#a = 'this is a string' # 变量a是个字符串, 引号不是字符串的一部分
#a = "Hello, I'm new Pythoner!" # 字符串, 双引号可以把单引号作为字符串一部分
#a = "I'm \"Ok\"" # 如果字符串内部包含单引号'和双引号", 可以用转义字符\来标识
print(a, type(a))
```

3 <class 'int'>

In [4]:



```
# Python支持多重赋值
a, b, c = 2, "hello, ", 4
a, b, c
```

Out[4]:

(2, 'hello, ', 4)

In [5]:



```
# Python 支持字符串的灵活操作, 拼接
c = "Python"
b + c + '!'
```

Out[5]:

'hello, Python!'

注意: 转义字符\可以转义很多字符, 比如\n表示换行, \t表示制表符, 字符\本身也要转义, 所以\\表示的字符就是\, 可以在Python的交互式命令行用print打印字符串看看

如果字符串里面有很多字符都需要转义, 就需要加很多\, 为了简化, Python还允许用r'...'表示'...'内部的字符串默认不转义,

In [6]:



```
print("Change\ta \\new\nline") # 转义字符
```

Change a \new
line

In [7]:



```
# Python还允许用r''表示''内部的字符串默认不转义
print('\\\\t\\') # 转义
print(r'\\\\t\\') # 不转义
```

```
\\t\\
\\\\t\\
```

如果字符串内部有很多换行，用 `\n` 写在一行里不好阅读，为了简化，Python 允许用 `'''...'''` 的格式表示多行内容，如下：

- 如果在交互式命令行内输入

```
>>> print('''first line
... second line
... third line''')
```

- 如果写成程序的形式：

```
print('''first line
second line
third line''')
```

In [8]:



```
# 在程序内表示多行内容：
print('''first line
second line
third line''')
```

```
first line
second line
third line
```

布尔值：

只有 True、False 两种值，注意大小写

布尔值经常用在条件判断中

布尔值可以用 and、or 和 not 运算

- and 与运算，只有所有都为 True，and 运算结果才是 True
- or 或运算，只要其中有一个为 True，or 运算结果就是 True
- not 非运算，是一个单目运算符，把 True 变成 False，False 变成 True

In [9]:

```
print(True) # 布尔值
print(3 > 2) # 布尔值
print(3 > 5) # 布尔值

# 布尔值可以用 and、or 和 not 运算
print(3 > 2 and 3 > 5) # True and False = False
print(3 < 2 and 3 > 5) # False and False = False
print(3 > 2 or 3 > 5) # True or False = True
print(3 is 2) # 注意不是 3=2, = 是赋值符号, 不等同于数学的等号
print(not(3 is 2))
```

```
True
True
False
False
False
True
False
True
```

空值

空值是 Python 里一个特殊的值，用 None 表示。None 不能理解为 0，因为 0 是有意义的，而 None 是一个特殊的空值。

号 =

是赋值语句，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量。这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。例如 Java 是静态语言，赋值语句如下：

```
int a = 123; // a是整数类型变量
a = "ABC"; // 错误：不能把字符串赋给整型变量
```

In [10]:

```
a = 'ABC' # 1) 在内存中创建了一个'ABC'的字符串; 2) 内存中创建了a变量, 并指向'ABC'
b = a # 变量b指向变量a所指向的数据
a = 'XYZ' # 改变变量a的值, 但是没有改变变量b的值
print(a, b)
```

XYZ ABC

In [11]:

```
# 基本数据类型运算
a = "Hello, Python!" * 2
#a = 3 + 4
#a = 9 // 4 # 整数除法 (2)
#a = 9 / 4 # 精确除法 (2.25)
#a = 9.0 / 4 # 精确除法, 只需把其中一个整数换成浮点数做除法就可以
#a = 17 % 5 # 余数运算, 可以得到两个整数相除的余数
#a = 3 ** 2
#a = [1, 3, 5, 9] # 列表list, 可以使用 type(a) 查看, print a[0] = 1, type(a[0]) = int
#a = (1.0, 3, 7) # 元组tuple, 可以使用 type(a) 查看, print a[0] = 1.0, type(a[0]) = float
#a = ([1.0, 2], [3, 4]) # 元组tuple, 可以使用 type(a) 查看, print a[0] = [1.0, 2], type(a[0]) = 列表
#a = {'1.0': 'name', 2: 'age', 3: 'profession'} # 字典dict存储key-value对, 可以使用 type(a) 查看, 不可变
# 但可以使用print a['1.0'] = 'name', a[2] = 'age'; type(a[2]) = str; a.has_key(2) = True; print(a.i
print(a, type(a))
```

Hello, Python!Hello, Python! <class 'str'>

In [12]:

```
# Python编码, 最早的Python只支持ASCII编码, 普通的字符串'ABC'在Python内部都是ASCII编码的
# Python提供了ord()和chr()函数, 可以把字母和对应的数字相互转换:
print(ord('A')) # 字母转换为ASCII编码数字
print(chr(65)) # ASCII编码数字转换为字母
```

65

A

In [13]:

```
# Python后来添加了对Unicode的支持, 以Unicode表示的字符串用u'...'表示, 比如:
print(u'中文')
print(u'\u4e2d') # Unicode表示的中文字
print(u'中') # 中文的Unicode, 使用Print, 则直接进行编码转换
u'中' # u'中'和u'\u4e2d'是一样的, \u后面是十六进制的Unicode码。
```

中文

中

中

Out[13]:

'中'

两种字符串如何相互转换?

- 字符串 'xxx' 虽然是 ASCII 编码, 但也可以看成是 UTF-8 编码,
- 而 u'xxx' 则只能是 Unicode 编码。
- 把 u'xxx' 转换为 UTF-8 编码的 'xxx' 用 encode('utf-8') 方法
- 反过来, 把 UTF-8 编码表示的字符串 'xxx' 转换为 Unicode 字符串 u'xxx' 用 decode('utf-8') 方法

In [14]:

```
# 把u'xxx'转换为UTF-8编码的'xxx'用encode('utf-8')方法  
u'中文'.encode('utf-8') # 中文字符转换后1个Unicode字符将变为3个UTF-8字符
```

Out[14]:

```
b'\xe4\xb8\xad\xe6\x96\x87'
```

In [15]:

```
# 把UTF-8编码表示的字符串'xxx'转换为Unicode字符串u'xxx'用decode('utf-8')方法  
b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
```

Out[15]:

```
'中文'
```

In [16]:

```
print(b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8'))
```

```
中文
```

In [17]:

```
u'中文'.encode('gb2312')
```

Out[17]:

```
b'\xd6\xd0\xce\xcf'
```

In [18]:

```
u'中文'.encode('utf-8')
```

Out[18]:

```
b'\xe4\xb8\xad\xe6\x96\x87'
```

In [19]:

```
u'中文'.encode('GBK')
```

Out[19]:

```
b'\xd6\xd0\xce\xcf'
```

注意:

由于Python源代码也是一个文本文件，所以，当源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。

当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释；第二行注释是为了告诉Python解释器，按照UTF-8编码读取源代码，否则，在源代码中写的中文输出可能会有乱码。

如果你使用Notepad++进行编辑，除了要加上# -- coding: utf-8 --外，中文字符串必须是Unicode字符串：

```
print(u'中文测试正常')
```

申明了UTF-8编码并不意味着你的.py文件就是UTF-8编码的，必须并且要确保Notepad++正在使用UTF-8 without BOM编码：

输出格式化的字符串

当输出类似“亲爱的xxx你好！你xxx月的话费是xxx，余额是xxx”之类的字符串，而xxx的内容都是根据变量变化的，所以，需要一种简便的格式化字符串的方式。

%运算符 就是用来格式化字符串的。在字符串内部，%s表示用字符串替换，%d表示用整数替换，有几个%?占位符，后面就跟几个变量或者值，顺序要对应好。如果只有一个%?，括号可以省略。常见的占位符有：

- %d 整数
- %f 浮点数
- %s 字符串
- %x 十六进制整数

其中，格式化整数和浮点数还可以指定是否补0和整数与小数的位数

In [20]:

```
'Hello, %s' % 'world'
```

Out[20]:

```
'Hello, world'
```

In [21]:

```
'Hi, %s, you have $%.d.' % ('Michael', 1000000)
```

Out[21]:

```
'Hi, Michael, you have $1000000.'
```


In [22]:

```
#%s永远起作用，它会把任何数据类型转换为字符串  
'Hi, %s, you have $%s.' % ('Michael', 100.0)
```

Out[22]:

```
'Hi, Michael, you have $100.0.'
```

In [23]:

```
'%2d-%02d' % (3, 1) # 补0
```

Out[23]:

```
' 3-01'
```

In [24]:

```
'%.2f%' % 3.1415926 # 保留小数后2位，后面两个%表示转义
```

Out[24]:

```
'3.14%'
```

2. Python内置的数据类型（列表/元组/字典/集合）

- 列表list: 用 `[]`，可以修改
- 元组tuple: 用 `()`，不能修改
- 字典dict: 用 `{}`，`{key:value}`键值对
- 集合set: 用 `{}` ,set 和 dict 的区别仅在于没有存储对应的 value

1.列表list

Python内置的一种数据类型是列表list。list是一种有序的集合，可以随时添加和删除其中的元素。

列表/元组（后面讲到）的函数如表：

| 函数 | 功能 |
|-------------------------------------|--------------------|
| <code>comp(a,b)</code> | 比较两个列表/元组的元素 |
| <code>len(a)</code> | 列表/元组的元素个数 |
| <code>max(a), min(a), sum(a)</code> | 返回列表/元组的元素最大、最小和求和 |
| <code>sorted(a)</code> | 对列表的元素进行升序排序 |

列表相关的方法如下：

| 函数 | 功能 |
|--------------------------|-----------------------------|
| <code>a.append(1)</code> | 将1添加到列表a末尾，如果1是列表，则a列表中嵌套列表 |

| 函数 | 功能 |
|-----------------|-----------------------------|
| a.count(1) | 统计列表a中元素1出现的次数 |
| a.extend([1,2]) | 将列表[1,2]中的内容追加到列表a的末尾中，即a+b |
| a.index(1) | 从列表a中找出第一个1出现的索引位置 |
| a.insert(2, 1) | 将1插入到列表a的索引为2的位置 |
| a.pop(1) | 移除列表a中索引为1的元素 |

```
a = [1, 3, 5]
a.
a.append
a.count
a.extend
a.index
a.insert
a.pop
a.remove
a.reverse
a.sort
AttributeError
```

In [25]:

```
a = [1, 2]
b = [3, 4]
a + b
```

Out[25]:

[1, 2, 3, 4]

In [26]:

```
a.append(b)
a
```

Out[26]:

[1, 2, [3, 4]]

In [27]:

```
# 定义学生名册列表
namelist = ['Michael', 'Bob', 'Tracy']
namelist
```

Out[27]:

['Michael', 'Bob', 'Tracy']

In [28]:

```
# 用len()函数可以获得list元素的个数:  
len(namelist)
```

Out[28]:

3

In [29]:

```
# 用索引来访问list中每一个位置的元素，记得索引是从0开始的:  
print(namelist[0]); print(namelist[2])  
#namelist[3] # 超出index范围，报错
```

Michael
Tracy

In [30]:

```
# 如果要取最后一个元素，除了计算索引位置外，还可以用-1做索引，直接获取最后一个元素:  
namelist[-2]
```

Out[30]:

'Bob'

In [31]:

```
# list是一个可变的有序表，所以，可以往list中追加元素到末尾:  
namelist.append('Adam')  
namelist
```

Out[31]:

['Michael', 'Bob', 'Tracy', 'Adam']

In [32]:

```
# 也可以把元素插入到指定的位置，比如索引号为1的位置:  
namelist.insert(1, 'Jack')  
namelist
```

Out[32]:

['Michael', 'Jack', 'Bob', 'Tracy', 'Adam']

In [33]:

```
# 要删除list末尾的元素，用pop()方法：  
namelist.pop()
```

Out[33]:

```
'Adam'
```

In [34]:

```
namelist
```

Out[34]:

```
['Michael', 'Jack', 'Bob', 'Tracy']
```

In [35]:

```
# 要删除指定位置的元素，用pop(i)方法，其中i是索引位置：  
namelist.pop(1)
```

Out[35]:

```
'Jack'
```

In [36]:

```
namelist
```

Out[36]:

```
['Michael', 'Bob', 'Tracy']
```

In [37]:

```
# 要把某个元素替换成别的元素，可以直接赋值给对应的索引位置：  
namelist[0] = 'Isabel'  
print(namelist)
```

```
['Isabel', 'Bob', 'Tracy']
```

In [38]:

```
# list里面的元素的数据类型也可以不同  
a = [7, 3, 3, 5.0, 'hello', 'Python 2.7']
```

In [39]:

```
# list元素也可以是另一个list
b = ['python', 'java', ['asp', 'php'], 'scheme']
print(len(b))
```

4

In [40]:

```
# 两个列表相加，进行元素扩展，等同与aa.extend(b)
c = a + b
print(c)
```

[7, 3, 3, 5.0, 'hello', 'Python 2.7', 'python', 'java', ['asp', 'php'], 'scheme']

列表解析（列表推导式）：是个非常重要的功能，能够简化对列表内元素逐一进行操作的代码，非常方便，简洁！体现Python的人性化。

In [41]:

```
a = [7, 3, 3, 5.0, 'hello', 'Python 2.7']
#a.append(9) # 往list添加元素
#a.count(3) # list中元素3出现的次数
#b = ['Python', [1.0, 3]]; a.extend(b) # 往列表a中扩展列表b
#b = ['Python', [1.0, 3]]; a = a + b # 与上句等同，往列表a中扩展列表b
#b = ['Python', [1.0, 3]]; a.append(b) # 与上面不同，将列表b作为一个list元素增加到列表a中
#a.index(3) # list中元素3出现的第一个索引index（从0开始）
#len(a); a.insert(len(a)-1, 0) # L.insert(index, object) -- insert object before index
#a.pop() # 从list中删去最后一个元素
#a.remove(3) # 从list中删去第一次出现的元素3
# a.remove("python") # 如果不存在返回ValueError

#a = [7, 3, 3, 5.0]
#a.reverse() # 颠倒列表元素的顺序
#a.sort() # L.sort(cmp=None, key=None, reverse=False) 默认排序由小到大
#a.sort(reverse=True) # 改变排序顺序由大到小
print(a)
```

[7, 3, 3, 5.0, 'hello', 'Python 2.7']

In [42]:

```
# 列表解析，对a列表中每个元素求平方
a = range(5)
b = [i **2 for i in a]
b
```

Out[42]:

[0, 1, 4, 9, 16]

2. 元组tuple的属性 (attribute)

元组tuple与列表list非常类似，不同之处在于：

- 列表使用方括号[],元组使用小括号() (后面介绍的字典使用花括号{})
- 元组内的元素不能修改,是不可变列表list。一旦创建了一个 tuple 就不能以任何方式改变它，即元组是没有 append(), insert()这样的方法。

元组与列表的相同之处：

- 元组创建很简单,只需要在括号中添加元素,并使用逗号隔开即可
- 元组的元素与列表一样按定义的次序进行排序，索引一样从 0 开始,可以正常地使用tuple[0], tuple[-1], 但不能赋值成另外的元素，负数索引一样从 tuple 的尾部开始计数。
- 与 list 一样分片 (slice) 也可以使用。注意当分割一个 list 时, 会得到一个新的 list；当分割一个 tuple 时, 会得到一个新的 tuple。

Tuple 与 list 的转换：

- Tuple 可以转换成 list，反之亦然。
- 内置的 tuple 函数接收一个 list，并返回一个有着相同元素的 tuple。
- list 函数接收一个 tuple 返回一个 list。从效果上看，tuple 冻结一个 list，而 list 解冻一个 tuple。
- **不可变的tuple有什么意义？因为tuple不可变，所以代码更安全。如果可能，能用tuple代替list就尽量用 tuple。**
- **tuple的陷阱：当定义一个tuple时，在定义的时候，tuple的元素就必须被确定下来**

Tuple 不存在的方法：

- 不能向 tuple 增加元素。Tuple 没有 append 或 extend 方法。
- 不能从 tuple 删除元素。Tuple 没有 remove 或 pop 方法。
- 不能在 tuple 中查找元素。Tuple 没有 index 方法。然而, 可以使用 in 来查看一个元素是否存在于 tuple 中。

使用 Tuple 的好处：

- Tuple 比 list 操作速度快。如果定义了一个值的常量集，并且唯一要用它做的是不断地遍历它，请使用 tuple 代替 list。
- 如果对不需要修改的数据进行“写保护”，可以使代码更安全。使用 tuple 而不是 list 如同拥有一个隐含的 assert 语句，说明这一数据是常量。如果必须要改变这些值，则需要执行 tuple 到 list 的转换。

In [43]:

```
# 定义学生名册元组
namelist = ('Michael', 'Bob', 'Tracy')
namelist
```

Out[43]:

```
('Michael', 'Bob', 'Tracy')
```

现在，namelist这个tuple不能变了，它也没有append(), insert()这样的方法。其他获取元素的方法和list是一样的，可以正常地使用namelist[0], namelist[-1], 但不能赋值成另外的元素。

In [44]:

```
# 元组也是一个序列，可以使用下标索引来访问元组中的值
# 与字符串类似，下标索引从0开始，可以进行截取，组合等。
print(namelist[1:3])
```

('Bob', 'Tracy')

In [45]:

```
# 元组内的元素不可以修改，以下修改元组元素操作是非法的。
# tup1[1] = 'math' # 不能修改元组的元素
```

In [46]:

```
# 元组中的元素值是不允许修改的，但与字符串一样，元组之间可以使用 + 号和 * 号进行运算。
#这就意味着他们可以组合和复制，运算后会生成一个新的元组。
tup1 = (1, 3, 5)
tup2 = (6, 8)
tup3 = ()
tup3 = tup1 + tup2
print(tup3)
```

(1, 3, 5, 6, 8)

In [47]:

```
# 元组之间可以使用 * 号进行复制运算。
tup1 = (1, 3, 5)
tup2 = (3)
tup3 = tup1 * tup2
tup3
```

Out[47]:

(1, 3, 5, 1, 3, 5, 1, 3, 5)

In [48]:

```
#元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组
tup = ('physics', 'chemistry', 1997, 2000);
print(tup);
del tup;
print("After deleting tup : ")
#print(tup) # 删除元组后会报错
```

('physics', 'chemistry', 1997, 2000)
After deleting tup :

In [49]:



```
#任意无符号的对象，以逗号隔开，默认为元组
print('abc', -4.24e93, 18+6.6j, 'xyz')
x, y = 1, 2
print("Value of x , y : ", x, y)
print(x)
```

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y :  1 2
1
```

注意:

要定义一个只有1个元素的tuple时必须加一个逗号,, 来消除歧义:

```
t = (1,)
```

Python在显示只有1个元素的tuple时, 也会加一个逗号,, 以免你误解成数学计算意义上的括号。

In [50]:



```
# 下面定义的不是tuple, 是1这个数! 因为括号()既可以表示tuple, 又可以表示数学公式中的小括号,
# 为了避免歧义, Python规定, 这种情况下, 按小括号进行计算, 计算结果自然是1。
t = (1)
print(t)
```

```
1
```

In [51]:



```
# 只有1个元素的tuple定义时必须加一个逗号,, 来消除歧义
# Python在显示只有1个元素的tuple时, 也会加一个逗号,, 以免你误解成数学计算意义上的括号。
t = (1,)
print(t)
```

```
(1,)
```

In [52]:



```
#来看一个“可变的” tuple:
t = ('a', 'b', ['A', 'B'])
print(t)
# 元组的第三个元素为list类型, 这是不变的, 但是可以修改list类型中的元素值
t[2][0] = 'X'
t[2][1] = 'Y'
print(t)
```

```
('a', 'b', ['A', 'B'])
('a', 'b', ['X', 'Y'])
```

注意: 这里tuple的元素确实变了, 但其实变的不是tuple的元素, 而是list的元素。tuple一开始指向的list并没有改成别的list, 所以, tuple所谓的“不变”是说, tuple的每个元素, 指向永远不变, 即指向一个list, 就不能改成

指向其他对象，但指向的这个list本身是可变的！

Python中元组包含了以下内置函数：

- `cmp(tuple1, tuple2)`: 比较两个元组元素。
- `len(tuple)`: 计算元组元素个数。
- `max(tuple)`: 返回元组中元素最大值。
- `min(tuple)`: 返回元组中元素最小值。
- `tuple(seq)`: 将列表转换为元组

Python中元组的方法：

- `count()`: 查找元素在元组中出现的次数。
- `index()`: 查找元素的第一个索引值。

In [53]:

```
import operator
tup1 = ('physics', 'chemistry', '1997', '2000')
tup2 = (1, 2, 3, 4, 5)
# print(cmp(tup1, tup2)) # 比较两个元组是否不同, 0为相同, 1为不同 python2版本, 不支持python3, 可以用op.
print(len(tup1)) # 计算元组中元素个数
print(max(tup1)) # 返回元组中元素最大值, tup=('str', 1), max函数报错
print(min(tup1)) # 返回元组中元素最小值
```

```
4
physics
1997
```

In [54]:

```
list1 = [1, 3, 2, 5]
print(list1) # 列表
list2 = tuple(list1) # 将列表转换为元组
print(list2) # 注意: list2其实是个元组类型
```

```
[1, 3, 2, 5]
(1, 3, 2, 5)
```

In [55]:

```
tup1 = tup2 + list2
print(tup1)
print(tup1.count(1)) # 查找元素1在元组中出现的次数。
print(tup1.index(5)) # 查找元素5在元组中的第一个索引值
```

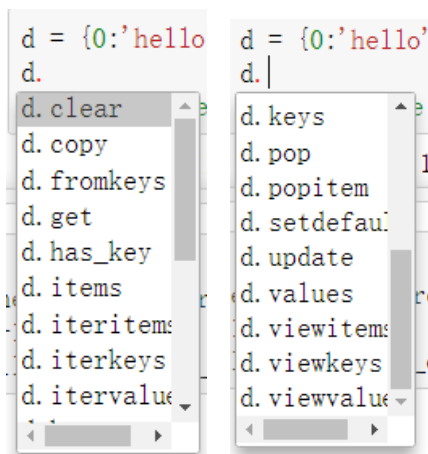
```
(1, 2, 3, 4, 5, 1, 3, 2, 5)
2
4
```

3. 字典dict的属性 (attribute)

- 在其他语言中也称为map，或关联数组或哈希表，具有极快的查找速度
- 字典由键和对应值成对组成。每个键与值用冒号隔开 (:)，每对用逗号，每对用逗号分割，整体放在花括号中{}
- 键必须独一无二，但值则不必唯一，如果同一个键被赋值两次，只有后一个值会被记住
- 值可以取任何数据类型，但必须是不可变的，所以可以用数，字符串或元组充当，但是不可以用列表

Python字典包含了以下内置方法：

- 1、dict.clear(): 删除字典内所有元素
- 2、dict.copy(): 返回一个字典的浅复制
- 3、dict.fromkeys(): 创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
- 4、dict.get(key, default=None): 返回指定键的值，如果值不在字典中返回default值
- 5、dict.has_key(key): 如果键在字典dict里返回true，否则返回false
- 6、dict.items(): 以列表返回可遍历的(键, 值) 元组数组
- 7、dict.keys(): 以列表返回一个字典所有的键
- 8、dict.setdefault(key, default=None): 和get()类似，但如果键不已经存在于字典中，将会添加键并将值设为 default
- 9、dict.update(dict2): 把字典dict2的键/值对更新到dict里
- 10、dict.values(): 以列表返回字典中的所有值



In [56]:

```
# 查找学生成绩，以前只能使用两个列表
names = ['Michael', 'Bob', 'Tracy']
scores = [95, 75, 85]
# 给定一个名字，要查找对应的成绩的步骤如下：
# 1. 先要在names中找到对应的位置，
# 2. 再从scores取出对应的成绩，
# 因此，list越长，耗时越长。
```

In [57]:

```
# zip函数将两个list对应的元素合并为一个dict的key和value
d = dict(zip(names, scores))
print(d)
```

```
{'Michael': 95, 'Bob': 75, 'Tracy': 85}
```

In [58]:



```
# 请务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。  
d
```

Out[58]:

```
{'Michael': 95, 'Bob': 75, 'Tracy': 85}
```

In [59]:



```
# 使用字典dict实现，只需要一个“名字” - “成绩”的对照表，  
# 直接根据名字查找成绩，无论这个表有多大，查找速度都不会变慢。  
d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}  
d['Michael']
```

Out[59]:

```
95
```

In [60]:



```
# 把数据放入dict的方法，除了初始化时指定外，还可以通过key放入：  
d['Adam'] = 67  
d['Adam']
```

Out[60]:

```
67
```

In [61]:



```
# 由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉  
d['Jack'] = 90  
print(d['Jack'])  
d['Jack'] = 88  
print(d['Jack'])
```

```
90
```

```
88
```

In [62]:



```
# 如果key不存在，dict就会报错：  
#d['Thomas'] # KeyError: 'Thomas'
```

In [63]:

```
# 要避免key不存在的错误有两种办法:  
# 方法一: 通过in判断key是否存在:  
'Thomas' in d
```

Out[63]:

False

In [64]:

```
# 方法二: 通过dict提供的get方法, 如果key不存在, 可以返回None, 或者自己指定的value:  
print(d.get('Thomas')) # 如果key不存在, 可以返回None  
# 注意: 返回None的时候Python的交互式命令行不显示结果, 所以这里使用print  
print(d.get('Thomas', -1)) # 如果key不存在, 可以返回自己指定的value, 如-1  
print(d.get('Thomas', 'No Such Key Error')) # key不存在返回自己指定的msg
```

None
-1
No Such Key Error

In [65]:

```
# 要删除一个key, 用pop(key)方法, 对应的value也会从dict中删除:  
print(d)  
d.pop('Bob')  
print(d)
```

```
{'Michael': 95, 'Bob': 75, 'Tracy': 85, 'Adam': 67, 'Jack': 88}  
{'Michael': 95, 'Tracy': 85, 'Adam': 67, 'Jack': 88}
```

注意:

和list比较, dict有以下几个特点:

- 1.查找和插入的速度极快, 不会随着key的增加而增加;
- 2.需要占用大量的内存, 内存浪费多。

而list相反:

- 1.查找和插入的时间随着元素的增加而增加;
- 2.占用空间小, 浪费内存很少。

所以, dict是用空间来换取时间的一种方法。

dict可以用在需要高速查找的很多地方, 在Python代码中几乎无处不在, 正确使用dict非常重要, 需要牢记的第一条就是**dict的key必须是不可变对象**。

这是因为dict根据key来计算value的存储位置, 如果每次计算相同的key得出的结果不同, 那dict内部就完全混乱了。通过key计算位置的算法称为哈希算法 (Hash) 。

要保证hash的正确性, 作为key的对象就不能变。在Python中, 字符串、整数等都是不可变的, 因此, 可以放心地作为key。而list是可变的, 就不能作为key, 如下会报错:

In [66]:

```
#key = [1, 2, 3]
#d[key] = 'a list' # 出错 TypeError: unhashable type: 'list'
```

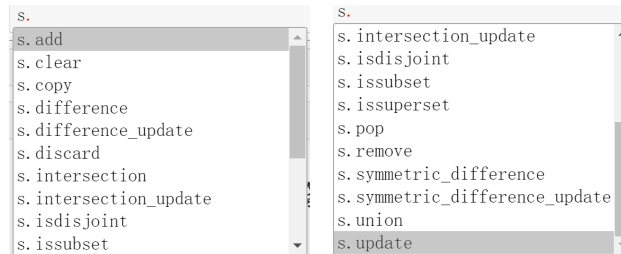
In [67]:

```
#d = {0:'hello', 1.2:1, 2.7:'Python', 3.4:[[1, 2] [3, 4]]}
d = {0:'hello', '1.2':1.0, 'name':'Python', 'list':[[1, 2], [3, 4]]}
print('d = %s' % str(d))
#d.clear() # 清空词典所有条目
#del d['name']; # 删除键是'name'的条目
#del d; # 删除词典
#c = d.copy(); print(c) # 得到字典的复制
#seq = ('name', 'age', 'sex'); d = dict.fromkeys(seq, 10); # dict.fromkeys(s[, v])创建一个新字典以后
#print(d.get('1.2')) # dict.get(key, default=None)返回指定键的值, 如果值不在字典中返回default值(默
print(d.items()) # 以列表返回可遍历的(键, 值) 元组数组
print(list(d.items())[0]) # 以列表返回可遍历的(键, 值) 元组数组的第一个元素值
for i in d.items(): print(i) # 返回可遍历的键值对
for i in d.keys(): print(i) # 返回可遍历的键
for i in d.values(): print(i) # 返回可遍历的值
print(d.keys()) # 以列表返回一个字典所有的键
print(d.pop(0)); print(d.pop('1', 'None')); print(d.pop('1.2', 'None')) #dict.pop(k[, d]) 返回指定k键
print(d.popitem()); print(d.popitem()) # 删除并返回 (k, v) 对的二元组, 如果d为空引发KeyError
d.setdefault(0, 7) # 设定字典中k=1的键的v=7, 如果k不在字典中, 增加(k, v)键值对, also set D[k]=d if k .
d.setdefault(1, 7) # 设定字典中k=1的键的v=7, 如果k不在字典中, 增加(k, v)键值对, also set D[k]=d if k .
d2 = {'name':'python', 'age':'2.7'}; d.update(d2) # 把字典d2的键/值对更新到dict里; 如果d2的k在d中, 只
print(d.values()) # 以列表返回一个字典所有的值
#d.viewitems() # python3中取消了
print(d)
```

```
d = {0: 'hello', '1.2': 1.0, 'name': 'Python', 'list': [[1, 2], [3, 4]]}
dict_items([(0, 'hello'), ('1.2', 1.0), ('name', 'Python'), ('list', [[1, 2], [3, 4]])])
(0, 'hello')
(0, 'hello')
('1.2', 1.0)
('name', 'Python')
('list', [[1, 2], [3, 4]])
0
1.2
name
list
hello
1.0
Python
[[1, 2], [3, 4]]
dict_keys([0, '1.2', 'name', 'list'])
hello
None
1.0
('list', [[1, 2], [3, 4]])
('name', 'Python')
dict_values([7, 7, 'python', '2.7'])
{0: 7, 1: 7, 'name': 'python', 'age': '2.7'}
```

4. 集合set的属性 (attribute)

- set 和 dict 类似，使用 {}，但 set 和 dict 的区别仅在于没有存储对应的 value
- set 的原理和 dict 一样，不可以放入可变对象，即不可以把 list 放入 set，因为无法判断两个可变对象是否相等，也就无法保证 set 内部“不会有重复元素”
- 由于 key 不能重复，所以集合 set 没有重复的 key 值
- set 可以看成数学意义上的无序，因此 set 不支持索引
- 两个 set 可以做数学意义上的交集 (&)、并集 (|)，求差 (-) 或对称差 (^) 等操作：



In [68]:

```
# 创建一个集合set可以通过 {}  
s = {1, 2, 3, 3} # 重复元素在set中自动被过滤  
print(s) # 这里显示的[]不表示这是一个list，只是说明这个set内部有1, 2, 3这3个元素
```

{1, 2, 3}

In [69]:

```
# 创建集合set，也可以使用set([])，根据已有的一个列表来创建集合：  
s = set([1, 2, 3, 3]) # 重复元素在set中自动被过滤  
s
```

Out[69]:

{1, 2, 3}

In [70]:

```
# 通过add(key)方法可以添加元素到set中，可以重复添加，但不会有效果  
s.add(4)  
print(s)  
s.add(4) # 重复添加没有效果  
print(s)
```

{1, 2, 3, 4}
{1, 2, 3, 4}

In [71]:

```
#通过remove(key)方法可以删除元素:  
s.remove(3)  
s
```

Out[71]:

```
{1, 2, 4}
```

In [72]:

```
# set看成数学意义上的无序和无重复元素的集合, 因此, 两个set可以做交集、并集等操作:  
s1 = set([1, 2, 3])  
s2 = set([2, 3, 4])  
print(s1 & s2) # 两个集合的交集  
print(s1 | s2) # 两个集合的并集  
print(s1 - s2) # 差集, 在s1中, 但不在s2中  
print(s2 - s1) # 差集, 在s2中, 但不在s1中  
print(s1 ^ s2) # 两个集合的对称差集, 元素在s1或s2中, 但不同时在两者中
```

```
{2, 3}  
{1, 2, 3, 4}  
{1}  
{4}  
{1, 4}
```

In [73]:

```
# set里面的元素不可以放入可变对象, 即不可以放入list  
#s = {[1, 2, 3, [4, 5]]} # 报错, unhashable type: 'list'
```

In [74]:

```
# list是可变对象, 因此, 对list进行操作, list内部的内容是会变化的, 比如:  
a = ['c', 'b', 'a']  
print(a, type(a))  
a.sort()  
a # 可变对象list a 的内容已经发生改变
```

```
['c', 'b', 'a'] <class 'list'>
```

Out[74]:

```
['a', 'b', 'c']
```

In [75]:



```
# 而对于不可变对象，比如str，对str进行操作：
a = 'abc'
a.replace('a', 'A') # replace方法创建了一个新字符串'Abc'并返回
print(a) # str是不可变对象，因此a的内容没有改变
b = a.replace('a', 'A') # replace方法创建了一个新字符串'Abc'并返回给b
print(b)
a = a.replace('a', 'A') # replace方法创建了一个新字符串'Abc'并重新赋值给a，此时a的内容改变
print(a)
```

```
abc
Abc
Abc
```

所以，对于不变对象来说，调用对象自身的任意方法，也不会改变该对象自身的内容。相反，这些方法会创建新的对象并返回，这样，就保证了不可变对象本身永远是不可变的。

2. Python 基本语法（条件/循环/函数/类/模块）

- 条件语句 (if)
- 循环语句
 - for x in list/tuple
 - while

1. 条件语句 (if)

- python 的 if 语句没有用括号来表示代码块，而是使用缩进
- 条件语句后面有冒号(:)
- 有多个条件时，可以使用 else，当条件不满足的时候执行它下面的语句块。else 是顶格写，并且后面记得写冒号(:)
- 如果还有更多的条件，我们可以使用 elif 做更细致的判断，可以有多个 elif，同样不要忘记冒号(:)和缩进
- if语句执行有个特点，它是从上往下判断，如果在某个判断上是 True，把该判断对应的语句执行后，就忽略掉剩下的 elif 和 else

In [76]:



```
age = 5
if age >= 18: # 注意不要少写了冒号:
    print("adult")
else: # 注意不要少写了冒号:
    print("teenager")
```

```
teenager
```


In [77]:

```
age = 5
if age >= 18:
    print("adult")
elif age >= 12: # 可以有多个elif语句
    print("teenager")
else:
    print("kid")
```

kid

注意：if 语句执行有个特点，它是从上往下判断，如果在某个判断上是 True，把该判断对应的语句执行后，就忽略掉剩下的 elif 和 else

所以，请测试并解释为什么下面的程序打印的是 teenager：

In [78]:

```
# 所以，请测试并解释为什么下面的程序打印的是 teenager:
age = 20
if age >= 12:
    print("teenager")
elif age >= 18:
    print("adult")
else:
    print("kid")
```

teenager

In [79]:

```
# if判断条件可以简写，只要x是非零数值、非空字符串、非空list等，就判断为True，否则为False
if age:
    print("True")
```

True

input()函数的用法

参考 `./data/input.py` 文件

注意：从 input() 读取的内容永远以**字符串**的形式返回，把字符串和整数比较就不会得到期待的结果，必须先用 int() 把字符串转换为我们想要的整型

2. 循环语句

Python 的循环有两种：

- 第一种是 for x in 循环，依次把 list 或 tuple 中的每个元素迭代出来代入变量 x，然后执行缩进块的语句
- 第二种是 while 循环，只要条件满足，就不断循环，条件不满足时退出循环
- print 函数默认在每一个输出后面添加了一个换行符，所以看起来输出的内容是一行一行的
- for 循环可以帮助我们处理字符串，假如我们想要分别输出字符串中的所有字母

- for 循环经常和 range 内置函数配合在一起使用，range 函数生成一个从零开始的列表(前面的例子)
- 还可以使用 for 循环来生成列表

In [80]:

```
# for...in循环, 依次把list或tuple中的每个元素迭代出来
names = ['Michael', 'Bob', 'Tracy']
for name in names:
    #print(name)          # 在每个输出后, 默认 newline, 变成多行输出
    print(name, end=',')  # 在每个输出后面加逗号, 变成每个输出之间有逗号的一行输出
```

Michael, Bob, Tracy,

In [81]:

```
# 再比如我们想计算1-10的整数之和, 可以用一个sum变量做累加:
sum = 0
for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    sum = sum + x
print(sum)
```

55

In [82]:

```
# 如果要计算1-100的整数之和, 从1写到100有点困难,
# Python提供range()函数可以生成一个整数序列, range(5)生成的序列是从0开始小于5的整数:
range(10)  # 返回 range(10), python3中返回的是迭代器
list(range(10))
```

Out[82]:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [83]:

```
# 计算1-100的整数之和,
sum = 0
for x in range(101):
    sum = sum + x
print(sum)
```

5050

In [84]:



```
# 第二种循环是while循环，只要条件满足，就不断循环，条件不满足时退出循环。
# 比如我们要计算100以内所有奇数之和，可以用while循环实现：
# 在循环内部变量n不断自减，直到变为-1时，不再满足while条件，循环退出。
sum = 0
n = 99
while n > 0:
    sum = sum + n
    n = n - 2
print(sum)
```

2500

注意：

循环是让计算机做重复任务的有效的办法，有些时候，如果代码写得有问题，会让程序陷入“死循环”，也就是永远循环下去。这时可以用 Ctrl+C 退出程序，或者强制结束 Python 进程

总结：

1. Python 的数据类型包括：整数，浮点数，字符，列表，元组，字典，集合，其中列表是可变类型
2. 条件 if 语句使得程序可以有判断力，循环是让计算机做重复任务的有效的办法