

菜鸟的 **Python** 笔记



目录

- ✓ 一、热身
- ✓ 二、开始编程
- ✓ 三、类型与操作
- ✓ 四、数字
- ✓ 五、字符串
- ✓ 六、列表
- ✓ 七、字典和元组
- ✓ 八、文件
- ✓ 九、基本语句
- ✓ 十、if 和 while 控制语句
- ✓ 十一、for 循环控制语句
- ✓ 十二、函数
- ✓ 十三、函数高级话题
- ✓ 十四、列表推导式(List comprehension)
- ✓ 十五、模块
- ✓ 十六、异常处理

一、热身

为什么选择 Python

软件质量：可读性、可复用性以及可维护性。

生产力：代码量相对小。

可移植性：可以跨越 Windows 和 Linux。

组建集成：Python 可以和 C 或者 C++等语言结合使用。

Python 的能力范围

Python 能做什么呢？

系统编程，GUI，网络编程以及数据库编程等等。

Python 的运行方式

Python 程序在运行之前要先从源程序被编译成字节码，这样可以加快程序的运行速度。字节码由 Python 虚拟机执行（PVM）。

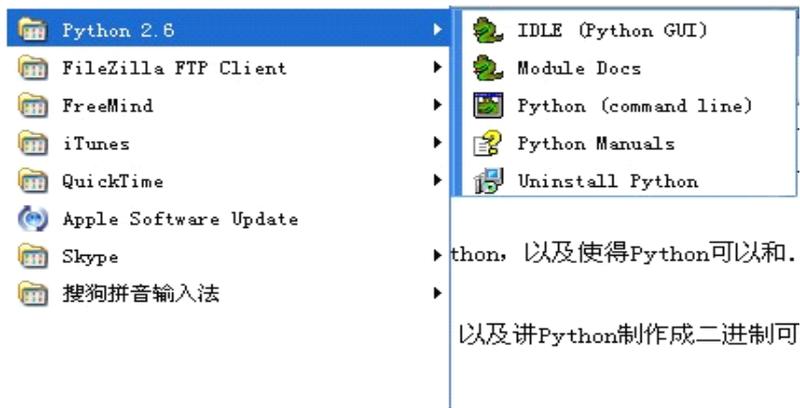
除了标准的 CPython 之外，还有 Java 上的 Python 实现——Jython，以及使得 Python 可以和.NET 互通的 IronPython。

除此之外，有专门的 Python 优化程序可以提高代码的效率，以及讲 Python 制作成二进制可执行文件的程序。

安装 Python

到 <http://www.python.org/> 下载最新的稳定版本 Python。在 Windows 平台下，安装过程和普通的软件没有什么区别，一路下一步就可以了。

安装成功之后，开始菜单里应该多出如下一项：



Python 是根据一个英国喜剧 [Monty Python](#) 命名的。

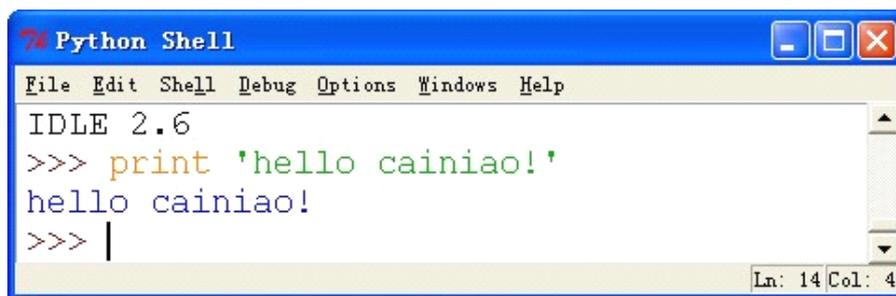
二、开始编程

命令行方式

在 Python 项目下有一个 IDLE 程序，打开它之后会出现 Python 的命令行窗口，可以以交互的形式执行程序。例如输入：

```
print 'hello cainiao!'
```

第一个程序就这么诞生了！如下如：



分别输入如下命令，可以简单地熟悉一下命令行环境：

```
2*2是4
```

```
2**10是1024
```

```
2***100是语法错误.....
```

不能随便空格：

```
print 100
```

#在语句前面输入空格是语法错误!!

注释格式：

```
print 100 #注释是这个样子的。
```

将程序存储在文件里

.py 文件

将 Python 语句保存到后缀为.py 的文件里即可。之后在 cmd 下执行

```
path/to/file.py
```

就可以运行程序了。

重定向

在 cmd 下执行：

```
path/to/file.py > save.txt
```

将会把 file.py 的输出存储在 save 文本文件里。

解决双击执行 Python 程序后 cmd 窗口消失

在 Windows 下，直接双击.py 文件执行程序，通常会闪过一个黑窗口，无法观察到任何输出。可以在程序的最后添加一句：

```
raw_input()
```

可以避免窗口消失。

Linux 可执行脚本

直接可以执行的脚本，关键是第一行的注释：

```
#!/usr/local/bin/python
```

```
print 'The Bright Side of Life...' # Another comment here
```

直接从书里抄下来的代码，第一行是指向 Python 解释器的路径。

模块(Python module)

引入模块

引入一个模块的语法：

```
import module #without suffix
```

注意，是不需要添加.py 后缀的。

重载模块

执行上面代码之后再次 import module 将不会有任何事情发生。需要：

```
reload(module)
```

实例

在 top.py 程序中调用另一个模块。

test.py 内容如下：

```
test = "testing"
```

top.py 内容如下：

```
import test  
print test.test
```

失败，对比书上也没找到毛病，最后随便敲了一下 test 发现原来 Python 内部有这么个模块，名字冲突了……汗，把文件名改成 myTest 就没问题了。

myTest.py 内容如下：

```
test = "testing"
```

top.py 内容如下：

```
import myTest  
print myTest.test
```

这里，test 叫做 myTest 模块的属性。

再次强调：import 对同一个模块来说，一个进程值可以 import 一次。

from

```
from myTest import test
```

使用 `from` 调用模块的时候，再使用 `test` 的时候，可以不用点来访问了，直接 `test` 就好。

dir

无论使用 `import` 还是 `from`，所有属性都会被引入。使用 `dir` 可以查看已经引入模块的所有属性。

```
print dir(myTest)
```

结果：

```
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'test']
```

其中有前后双下划线的东西是 Python 内置的，不用管。模块维护自己的命名空间

execfile

```
execfile('module.py')
```

另外一种运行文件的方式，可以在一个进程中多次运行文件，但是小心覆盖正在使用的变量。与其它语言的互通，书上把这个叫做 `Embedding Calls`。书上举了一个 C 调用 Python 的简单示例：

```
#include <Python.h>
...
Py_Initialize();
PyRun_SimpleString("x = brave + sir + robin");
```

三、类型与操作

基本变量类型

Number, 数字: 8。

String, 字符串: 'test'

List, 列表: [1,2,4,5]

Dictionaries, 字典: {'food': 'spam', 'taste': 'yum'}

Tuple, 元组: (1,'spam', 4, 'U')

File 文件: myfile = open('eggs', 'r')

其它类型: Sets, types, None, Booleans

数字简介

```
print 3.14 * 2
#math
import math
print math.pi
#random
import random
print random.random()
print random.random()
print random.choice([1,2,3,4])
print random.choice([1,2,3,4])
```

字符串简介

```
#Chinese

#中文字符串

strCN = '春暖花开'

#输出字符串

print strCN

#len 函数，字符串的长度

print len(strCN)

#访问第一个字符

print strCN[1]

#访问最后一个字符

print strCN[-1]

#English

#英文字符串

strEN = 'spring comes and the flower is open...'

print strEN

print len(strEN)

print strEN[0]

print strEN[-1]

#从0到5但是不包括5。

print strEN[0:5]#from 0 to 4

print strEN[0:-1]

#简略写法：分别表示到字符串结尾，和从字符串开头。

print strEN[0:]

print strEN[:-1]

#字符串可以进行+链接和乘法运算

print strCN + strEN

print strCN * 2
```

```

#String methods
#字符串方法，注意：所有的方法都不会改变字符串的值。
#需要再赋值给另一个新建字符串来保存修改后的值。

print strEN.find('flower')

print strEN.replace('flower','~_~')

print strEN.split(' ')

print strEN.upper()

print strEN.isalpha()

print strEN.rstrip()

#to get help
#取得帮助

print dir(strEN)

help(str)

#multiline
#多行字符串

print """<div>
    <p>HTML Code goes here...</p>
</div>"""

#pattern matching
#模式匹配

import re

result = re.match('(w+)',strEN)

print result.group(1)

```

列表 List 简介

```
#list
#新建列表
testList=[10086,'中国移动',[1,2,4,5]]

#访问列表长度
print len(testList)

#到列表结尾
print testList[1:]

#向列表添加元素
testList.append('i\'m new here!')

print len(testList)
print testList[-1]

#弹出列表的最后一个元素
print testList.pop(1)
print len(testList)
print testList

#list comprehension
#后面有介绍，暂时掠过
matrix = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]
print matrix
print matrix[1]

col2 = [row[1] for row in matrix]#get a column from a matrix
print col2

col2even = [row[1] for row in matrix if row[1] % 2 == 0]#filter odd item
```

```
print col2even
```

字典简介:

```
#新建一个字典
testDict={'time':'时间','machine':'机器','time machine':'时间机器'}
print testDict['time']
#do it in another way
#另一种构造字典的方式
newDict = {}
newDict['stuff']='start'
print newDict
#a person in Python from the book
#字典的属性可以是字典
rec = {'name': {'first': 'Bob', 'last': 'Smith'},
      'job': ['dev', 'mgr'],
      'age': 40.5}
print rec
rec = 0 #memory freed when the last reference is gone
#sort a dictionary
#字典排序
D = {'a': 1, 'b': 2, 'c': 3}
print D
for i in sorted(D):
    print i
    print D[i]
```

```
# optimization: list comprehension runs faster
# 优化: list comprehension 比 for 循环要快一些
squares = [x ** 2 for x in [1, 2, 3, 4, 5]]
print squares

#faster than
squares = []
for x in [1, 2, 3, 4, 5]:
    squares.append(x ** 2)
print squares

#test if a key exist in a dictionary
#查看某个字典中是否存在某个 key。
if not D.has_key('f'):
    print 'missing'
```

元组 Tuples 简介

和 list 类似，但是具有不变性，像 number 和 string 一样：

```
testTuple = (1,2,3,4)
print len(testTuple)
```

不变性 Immutability:

注意，以上的操作之后字符串本身是不变的，而且也不能像这样修改一个字符串：

```
someStr[0] = 't'
```

上面介绍的数字、字符串和元组都具有不可变性。而列表和字典则是可以改

变的。

四、数字

简单的运算符

除了之前简单介绍的运算之外，数字还有一个除完后约去小数的运算“//”：

```
print 50.0/3
print 50.0//3#floor division
```

此外，还有求余运算符“%”，幂运算符“**”。

位操作：左移“<<”、右移“>>”、按位或“|”、按位与“&”。

复数

在 Python 中，有内置的方法来表示复数，虚部使用 j 来表示。

```
#complex number
print 2+3j
print 2+3j*3
```

使用 **Decimal** 控制精确的小数点位数

```
#decimal
from decimal import Decimal
print Decimal('1.0') + Decimal('1.2')
```

布尔值与数字

真为1，假为0。

```
print bool(1)
```

```
print True + 2#True is 1 and False is 0
```

数字相关的模块

和数字相关的模块有：`math` 数学模块、`random` 随机模块、

等于和是的概念

“等于”和“是”（`==` and `is`）

```
#== tests if the two variables refer equal objects
L = [1,2,3]
M = [1,2,3]
print L == M
print L is M
#is tests if two variables refer the same object
```

等于检查两个变量是不是相等，而 `is` 检查两个变量是不是引用同一个对象。

五、字符串

转义字符

字符串可以使用双引号和单引号：

```
testStr = 'string'
testStr = "string"
```

换行和引号等需要使用转义字符来表示，例如：

\ " 表示 "

\\ 表示 \

在字符串前面加一个 r，这种字符串叫做 raw string，表示本字符串不转义：

```
myfile = open(r'C:\new\text.dat', 'w')
```

Unicode 字符串

如果要在字符串中存储 Unicode 的字符串，需要在字符串的前面加一个 u。在普通字符串上执行 unicode 函数也可以将字符串转换为 Unicode 的。

```
uniStr = u'unicode 的字符串'  
print uniStr  
print len(uniStr)  
print uniStr[10]  
print str(u'should be ASCII')#convert a unicode string to ASCII string  
print unicode('going to be unicode')
```

字符串方法

不能直接将字符串和数字相加，这和 [JavaScript](#) 不同。要使用 str 先将数字转换为字符串再相加。

```
#string operations  
#error >>>print 'add str to number ' + 9  
print 'add str to number ' + str(9) #this time it works
```

切割字符串

```
#slice a string  
#第七个字符到第十个字符  
print uniStr[7:11]  
#同上，但是只有偶数字符  
print uniStr[7:11:2]#only even char  
#从右向左
```

```
print uniStr[::-1]#right to left
#从右向左，每隔两个
print uniStr[::-2]#and one every other
#第一个是错误，第二个正确
print uniStr[7:11:-1]#why doesn't it work?
print uniStr[11:7:-1]#because...
```

删除输入字符串最后的换行

```
#sliceTestStr[:-1]
```

取得程序的参数列表

```
sliceTestList = ['echo.py', '-a', '-b', '-c']
sliceTestList[1:]#will get the arguments
```

ASCII 码的转换

ord 函数将字符转换为 ASCII 码。chr 函数则相反，将 ASCII 码转换为字符。

```
#convert between ASCII and character
print ord('d')#only works on one single ASCII char
print chr(97)
```

格式化字符串

类似 C 语言里的 printf，后面的变量会取代格式字符串里的%s。

```
#formatting strings
print "%s -- %s -- %s" % (42, 3.14159, [1, 2, 3])
```

以字典为基础的格式化字符串

在格式字符串中使用 %(name) 的格式，将被字典里的 key 为 name 的值所替代，例如：

```
#dictionary based formatting
print 'My name is %(name)s and I'm %(age)s years old.' % {'name':'Chen Zhe','age':'23'}
```

六、列表

列表的运算

列表也可以进行加和乘。

```
testList = [1,2,3,4,5]
print testList
print testList * 2
```

上面进行了列表的乘法，注意并不是每个元素乘以二，而是整个列表被重复了两次，重新接合成一个新的列表。

列表操作

```
#number of deleted and inserted need not match
#向列表中插入的元素和被插入的元素没有必要相等
testList[0:2] = [1,0,0,0]
print testList
#在列表的最后添加一个元素
testList.append(u'菜鸟')
print testList
#列表排序
testList.sort()
#扩展列表
testList.extend([7,8,9,10])
print testList
#讲列表反转
testList.reverse()
print testList
#弹出最后一个列表项目
```

```
testList.pop()
print testList
#弹出第一个列表项目
poppedItem = testList.pop(0)
print testList
print poppedItem
#删除部分列表
del testList[0:4]
print testList
```

重点：在 list 上调用 sort 之类的函数会造成 list 本身被改变，这个 immutable 的变量是不一样的。一定不能重新赋值，例如：

```
testList = testList.sort()
```

将会导致 testList 失去对原对象的引用。

注意引用的问题

用一个例子程序来说明这个问题：

```
#reference, no cot copy
#引用，而非拷贝
testList = [1,2,3,4]
#reference to testList
#引用 testList
newTestList = [9,8,7,6,testList]
print newTestList
#if testList is changed
#如果 testList 被改变了
testList[:] = [5,4,3,2,1]
```

```
#it will affect newTestList
#就会影响 newTestList
print newTestList
#but if testList is reassigned
#但是如果 testList 被重新定义了
testList = ['t','t','t','t','t']
#nothing happens to newTestList
#就不会影响 newTestList
print newTestList
```

七、字典和元组

字典

```
testDict = {'name':'Chen Zhe','gender':'male'}
print testDict
```

其中 name 和 gender 是字典的 key、而 Chen Zhe 和 male 是字典的 value。

取得 key 对应的 value

可以使用索引的方式，也可以使用 get 方法：

```
print testDict['name']
print testDict.get('name')#same as the above
```

字典操作

```
#判断某个字典里是否包含某个 key
```

```

print testDict.has_key('gender')
#所有的 key
print testDict.keys()
#所有的值
print testDict.values()
#所有的(key,value)元组
print testDict.items()
#给字典添加(key,value)对
updateDict = {'skill': 'JavaScript'}
testDict.update(updateDict)
print testDict
#使用 dict 构造函数来构造字典
#注意 key 不使用引号括起来
print dict(name='gaoshou')
print dict([('name', 'Chen Zhe'), ('gender', 'male')])
#暂时讲所有值都指定为0
#适合 keys 已知而值为动态决定的时候
print dict.fromkeys(['a', 'b'], 0)

```

给不存在的 key 赋值会扩展字典。key 不一定是 string。可以是任何 immutable 类型。

字典在内部是使用[哈希表](#)来实现的。

元组

元组与列表十分相似，但是类似字符串和数字它是不可改变的。

```

testTuple = (1,4,0)
print testTuple
#immutable,so if you want to sort a tuple:

```

```
#不可变，如果想要排序需要转换为列表
testTupleList = list(testTuple)
#在排序
testTupleList.sort()
sortedTuple = tuple(testTupleList)
#再转换为元祖
print sortedTuple
```

Python 中的三个数据类型

至此，数据类型大概有三个：

- 数字（number） 可以加和乘
- 序列（sequence） 可以索引、切割和连接（index、slice 和 concatenation）
- 映射（mapping） 可以使用 key 来索引（index）。

八、文件

打开一个文件并向其写入内容

Python 的 open 方法用来打开一个文件。第一个参数是文件的位置和文件名，第二个参数是读写模式。这里我们采用 w 模式，也就是写模式。在这种模式下，文件原有的内容将会被删除。

```
#to write
testFile = open('cainiao.txt','w')
#error testFile.write(u'菜鸟写 Python!')
#写入一个字符串
testFile.write('菜鸟写 Python!')
#字符串元组
codeStr = ('<div>','<p>', '完全没有必要啊!', '</p>', '</div>')
```

```
testFile.write('\n\n')
#将字符串元组按行写入文件
testFile.writelines(codeStr)
#关闭文件。
testFile.close()
```

向文件添加内容

在 `open` 的时候制定'a'即为（**append**）模式，在这种模式下，文件的原有内容不会消失，新写入的内容会自动被添加到文件的末尾。

```
#to append
testFile = open('cainiao.txt','a')
testFile.write('\n\n')
testFile.close()
```

读文件内容

在 `open` 的时候制定'r'即为读取模式，使用

```
#to read
testFile = open('cainiao.txt','r')
testStr = testFile.readline()
print testStr
testStr = testFile.read()
print testStr
testFile.close()
```

在文件中存储和恢复 Python 对象

使用 Python 的 pickle 模块，可以将 Python 对象直接存储在文件中，并且可以以后再需要的时候重新恢复到内容中。

```
testFile = open('pickle.txt','w')
#and import pickle
import pickle
testDict = {'name':'Chen Zhe','gender':'male'}
pickle.dump(testDict,testFile)
testFile.close()
testFile = open('pickle.txt','r')
print pickle.load(testFile)
testFile.close()
```

二进制模式

调用 open 函数的时候，在模式的字符串中使用添加一个 b 即为二进制模式。

```
#binary mode
testFile = open('cainiao.txt','wb')
#where wb means write and in binary
import struct
bytes = struct.pack('>i4sh',100,'string',250)
testFile.write(bytes)
testFile.close()
```

读取二进制文件

```
testFile = open('cainiao.txt','rb')
data = testFile.read()
values = struct.unpack('>i4sh',data)
print values
```

九、基本语句

基本语句的结束

与 C 语言不同, Python 的语句不需要用分号来结尾, 而是由解释器按照换行来判断语句的结束。

例如:

```
print 100  
print 300
```

以下情况除外, 一行多个语句的时候是需要分号的:

```
a =1;b=2;print a;
```

赋值语句

```
#normal  
#普通赋值  
a =1  
b =2  
print a,b  
  
#tuple assignment  
#元组赋值  
a,b = 'a','b'  
print a,b  
  
#and list  
#列表赋值  
[a,b]=[1,2]  
print a,b  
  
#and this is called sequence assignment  
顺序赋值: 下例分别存储一个字幕
```

```
a,b,c,d = 'HTML'

print a,b,c,d

#multiple target
#多目标赋值
a=b=c=d='菜'

print a,b,c,d
```

使用元组赋值的实例：

```
L = [1, 2, 3, 4]

while L:

    front, L = L[0], L[1:]

    print front, L
```

没有++

在 Python 中有+=之类的赋值，但是没有++和--这类运算符。

重定向输出流

下面的例子将 Python 的输出流绑定到一个文件上。

```
#change the stdout
#修改 stdout

from sys import stdout

temp = stdout #for later use

outputFile = open('out.txt','a')

stdout = outputFile

stdout.write('just a test')

#回复输出流

stdout = temp #restore the output stream
```

```
print >> outputFile,'changed for a little while\n'  
from sys import stderr  
print >> stderr,'error!\n'
```

十、if 和 while 控制语句

Python 语句的特点

与 C 语言对比（也就是说与大多数语言对比），Python 的语法结构有一些不同：

- 代码块是不需要使用大括号来括起来的。
- if, while 等等的条件是不需要使用小括号括起来的。
- 但是控制语句都需要添加一个冒号“:”。

if 语句

在下面的 if 语句中，代码块是按缩进的空格数量来判断的。也就是说空格数量一致的相邻行会被当作一个代码块，例如下面代码中红色表示的部分就是一个代码块，当 if 的条件成立的时候它就会得到执行。

```
#if elif else  
x = 100  
if x>50:
```

```
    print 'x is high, '
```

```
        print 'and high up in the sky!'
```

```
elif x==50:
```

```
    print 'x is middle'
```

```
else:
```

```
    print 'x is low'
```

逻辑运算符

```
#logic and, or, not
y=0
if x:
    print 'x is True'
if x and y:
    print 'x and y are all True'
else:
    print 'there\'s a bad guy!'
```

三元运算符

在类 C 的编程语言中，有一个三元运算符可以代替简单的 if、else 逻辑。例如：

```
A = X? Y : Z
```

在 Python 中如下表示：

```
A = Y if X else Z
```

while 语句

同 C 语言一样，Python 的循环中也有 break 和 continue。而且还添加了一个很方便的 else 功能，当 while 的判断的条件不成立的时候，不执行循环体，而是执行 else 中的代码块。

```
x = 5
while x:
    print 'Python'*x
    x -= 1
#if something bad:
# break
#if something
```

```
# continue
else:
#when the while loop doesn't exit with break
    print 'finished!'
```

十一、for 循环控制语句

基本的 for 循环语句

用一个列表来确定 for 循环的范围

```
x = [1,2,3,4]
for i in x:
    print i
else:
    print 'finished!'
```

循环一个字符串

```
#for in string (and also in string)
x = 'Python'
for i in x:
    print i
```

元组 for 循环

```
#tuple for in
x = [('XHTML','CSS'),('JavaScript','Python')]
for (a,b) in x:
```

```
print (a,b)
```

迭代器 (iterator)

下面用一个例子程序来简单了解 Python 中的迭代器

```
#file iterator,best practice to read a file
#and also the best loop form
#文件迭代器，读取文件的最佳实践
for line in open('test.txt'):
    print line.upper()
#dictionary
#字典迭代器
testDict = {'name':'Chen Zhe','gender':'male'}
for key in testDict:
    print key + ':'+ testDict[key]
#iterator in list comprehension
#list comprehension 中的迭代器
testList = [line for line in open('test.txt')]
print testList
```

迭代协议

有一些函数可以在支持迭代协议的对象上运行。例如：

```
#based on iteration protocol
testList = [9,8,7,6,5]
print sorted(testList)#doesn't change testList
#求和
print sum(testList)
```

```
#判断：至少有一个为真
print any(testList)#any is True
print all(testList)#all are True
print testList

#tuple, list, and join all work on iteration protocol
#元组、列表的构造函数以及 join 都可以对支持迭代协议的对象操作
print list(open('test.txt'))
print tuple(open('test.txt'))
print ('--').join(open('test.txt'))
```

循环技巧

使用 range 函数来产生循环的范围

```
#counter loop
#for(i=0;i<5;i++)
for i in range(5):
    print str(i) + ' is the current value'
else:
    print '-----finished-----'
for i in range(2,8):#for(i=0;i<5;i++)
    print str(i) + ' is the current value'
else:
    print '-----finished-----'
for i in range(2,9,2):#for(i=0;i<5;i++)
    print str(i) + ' is the current value'
else:
    print '-----finished-----'
```

zip 拉链

使用 `zip` 函数可以把两个列表合并起来，成为一个元组的列表。

```
L1 = [1,3,5,7]
L2 = [2,4,6,8]
#使用 zip 将两个列表合并
print zip(L1,L2)
for (a,b) in zip(L1,L2):
    print (a,b)
L3 = [2,4,6]
#当长度不一的时候，多余的被忽略
print zip(L1,L3)#extra items are ignored
#map 则不会忽略，例如下例。当 L1和 L3长度不一的时候，
#会用第一个参数来填充。
print map(None,L1,L3)
#'zip' a dictionary
#使用 zip 来造出一个字典。
keys = ['name','age']
values = ['Chen Zhe ',22]
print dict(zip(keys,values))
```

enumerate

`enumerate` 在循环的同时可以访问到当前的索引值。

```
#both offset and item
testStr = 'cainiao'
for (offset,item) in enumerate(testStr):
    print item,'appears at offset:',offset
```

十二、函数

函数

Python 是对接口编程，而不是对数据类型编程。例如我们定义了一个函数，在函数里用到了 `in` 这个接口，那么只要传入的参数实现了这个接口就可以，我们不在乎它是 `list` 还是 `tuple`。

简单的函数

使用 `def` 定义一个 `myAdd` 函数

```
def myAdd(a,b):  
    return a+b  
  
print myAdd(4,5)  
  
L1,L2 = [1,3,5],[2,4,6]  
print [myAdd(x,y) for(x,y) in zip(L1,L2)]
```

全局变量

函数里使用的变量为局部变量，可以使用 `global` 将变量的作用域扩大到文件内部。

```
def myGlobal():  
    global g  
    g = 100  
  
#out of the funcion body  
#在函数外部也可以使用变量 g  
myGlobal();  
print g
```

关键字调用函数（#keyword call of functio）

在调用函数的过程中可以不使用函数定义时候的参数顺序，但是一定要指明

参数的名称:

```
#keyword call of function

def show(a,b):
    print a,b

show(1,2)

show(b=1,a=2)
```

默认参数

可以在定义函数的时候给参数设定默认值，这样当调用函数的时候没有给这个参数赋值的时候，在函数内部将使用默认值。

```
#default

def showDefault(a,b=2,c=3):
    print a,b,c

showDefault(1,4,5)

showDefault(1,4)

showDefault(1)
```

结合参数 (#collecting arguments)

在定义函数的时候可以使用*args 指定在函数中使用元组的形式访问参数，使用**args 来指定按照字典形式来使用参数：

```
#collecting arguments

#in a tuple

#元组

def showArgs(*args):
    print args

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

```
#in a dictionary
#字典
def showArgsDict(**args):
    print args
showArgsDict(name = 'chenzhe',age=22)
```

拆解参数 (#unpacking arguments)

拆解参数是一个与集合参数相对应的概念，定义的时候采用常规的参数表示方式，但是调用的时候使用列表或者字典的方式：

```
def showArgsUnpacking(a,b,c,d):
    print a,b,c,d
args = [1,2,3,4]
#error showArgsUnpacking(args)
showArgsUnpacking(*args)
#this example doesn't work as the book
argsDict = {'a':1,'b':2,'c':3,'d':4}
showArgsDict(**argsDict)
```

将函数当作参数

在 Python 中，函数也可以被当作参数来传递。例如：

```
#pass function as a argument
def minmax(test, *args):
    res = args[0]
    for arg in args[1:]:
        if test(arg, res):
```

```
    res = arg
    return res
def lessthan(x, y): return x < y
def grtrthan(x, y): return x > y
print minmax(lessthan, 4, 2, 1, 5, 6, 3)
print minmax(grtrthan, 4, 2, 1, 5, 6, 3)
```

十三、函数高级话题

lambda

lambda 可以理解为一种小函数，但是它是一个表达式，而不是一个语句，所以在 def 不允许出现的地方仍然可以使用 lambda 函数，例如 list 里。但是 lambda 内只可以执行一个表达式。

```
#lambda
#普通的函数
def add(x,y):
    return x+y
print add(1,2)
#lambda 函数
func = lambda x,y:x+y
print func(1,2)
#default argument in lambda
#lambda 函数中的默认函数
func = lambda x,y=2:x+y
print func(1)
#why do we need lambda
#lambda 函数存在的意义
```

```

#a list of functions
#一个函数列表
L = [(lambda x: x**2), (lambda x: x**3), (lambda x: x**4)]
#function dictionary
#函数字典
key = 'got'
print {'already': (lambda: 2 + 2),
'got': (lambda: 2 * 4),
'one': (lambda: 2 ** 6)
}[key]()

```

map 函数

有时候我们可以需要处理一个列表里的所有元素，可以使用一个 for 循环来完成这个工作。但是 Python 内置的 map 函数可以帮我们的忙，它接受函数和列表作为参数，然后返回函数处理之后的列表：

```

#map function
#basic syntax
def mul2(x):
    return x*2
testList = [1,2,3,4]
print map(mul2,testList)
print map(lambda x: x*3,testList)
#map function that has two arguments
def mul(x,y):
    return x*y
print map(mul,[1,2,3,4],testList)

```

generator 函数

定义一个 generator 函数:

```
#basic syntax(a function that yield)
def genMul2(N):
    for i in range(N):
        yield i * 2
for i in genMul2(5):print i
#inside for, the next method is called
#在 for 循环的内部, Python 调用了 next 方法。
#下面的 x 叫做 generator 对象
x = genMul2(2)
#一直调用 next 方法, 最后会抛出一个异常
print x
print x.next()
print x.next()
#print x.next()  !!!error!!!StopIteration
```

迭代器

for 循环、list comprehension 和 map 都使用则个迭代器协议:

```
#iterator
testDict = {'name':'Chen Zhe','gender':'male'}
testIter = iter(testDict)
print testIter.next()
print testIter.next()
#generator expressions
#generator 表达式
testGen = (i*2 for i in range(4))
#testGen is a generator object
```

```
#testGen 是一个 generator 对象
print testGen.next()
```

函数陷阱，局部变量

```
#local variables

def selector():

    print X

X = 88

#selector() !!!!error!!!!assignment inside a function  makes it local
```

以上代码会报错，因为在函数内的任何地方给一个变量赋值都会使它变成局部变量。在上例中，X 编程了局部变量，而 print 语句在赋值之前，所以 Python 解释器认为变量尚未定义。

mutable(可更改的)默认参数

```
#mutable defaults

def saver(x=[]):

    x.append(1)

    print x
```

列表和字典都是可更改的类型，如果我们像上面的例子一样，定义函数的时候使用一个列表类型的默认参数。每次调用 saver 的时候，如果我们不提供一个参数，那么就一直使用相同的 x，也就意味着我们可以像 C 语言中的 static 变量一样使用这个变量，但是并不建议这么做。

函数设计原则

- 使用参数做输入，返回值做输出。仅在必要时使用全局变量。
- 每个函数应该有一个唯一的功能，多个功能的函数不容易重用。

十四、列表推导式

列表推导式(list comprehension)简介

所谓 list comprehension，就是一种很方便的遍历方式。而且除了方便之外，速度通常也会比 for 循环高出许多。

简单示例1:

```
#按行遍历一个文件，大写后输出
print [line.rstrip() for line in open('test.txt')]

#using if
#在 list comprehension 中使用 if 判断
print [line.rstrip() for line in open('test.txt') if line[0]!='n']
```

简单示例2:

```
testList = [1,2,3,4]
def mul2(x):
    print x*2

[mul2(i) for i in testList]

#add some if logic
#仍然是添加 if 判断
print '-----if logic:'

[mul2(i) for i in testList if i%2==0]
```

list comprehension 替代嵌套循环

常规的嵌套循：

```
#nested loop
for x in [1,2,3]:
    for y in [1,2,3]:
        z = x*y
        print str(x)+'*'+str(y)+' is: '+str(z)
```

使用 list comprehension 代替以上代码:

```
print [x*y for x in [1,2,3] for y in [1,2,3]]
```

十五、模块

import 的工作步骤

- 1 在 sys.path 上找到某个类型的 module。
- 2 可能会编译 module
- 3 运行 module

模块 module

```
to see the math namespace
print '-----import and namespace'
import math
print math.__name__ #return a dictionary
```

上面的代码引入了 math 模块，并且打印它的 __name__ 属性。

包 package

包是一个目录（文件夹），package 内必须有一个 __init__.py 文件，当其它程

序 import 这个 package 的时候，这个文件会自动运行。

假设的 dir1 中 __init__.py 内容如下：

```
print 'in dir1 __init__'  
initX = 98712
```

此外，dir1 中还有一个 test.py 文件，内容如下：

```
x = 100  
y = 1000
```

程序如下：

```
#package  
import dir1.test  
#package 的属性 initX  
#它是在 __init__.py 里定义的。  
print dir1.initX
```

其它的引用模块方式：

这样就可以直接使用 test 了，而不用再写 dir1.test

```
#from dir1 import test
```

使用 as 可以讲引用的模块有一个别名，方便记忆。

```
#import dir1.test as test
```

使用字符串动态地引用模块

如果事前不知道模块的名字，或者是需要动态地决定要引用模块的名字，可以使用字符串配合 exec 的方法 import。例如：

```
#import by string  
modname = "string"  
exec "import " + modname
```

reload

- reload 是函数，而不是语句。
- reload 会影响所有使用 import 的客户。
- reload 不会影响已有的 from 客户。

模块的数据隐藏

在一个模块中，`_xyz`、`_val` 等等前面有一个下划线的变量是会被 `from *` 所引入的。

如果从另外一个角度隐藏数据，可以在模块内定义一个 `__all__` 属性，例如：

```
__all__ = ["xyz", "zz", "zzx"]
```

表示只有 `__all__` 这个列表里的属性会被 `from *` 引入程序的命名空间。

`__name__`

当 module 是被当作顶级代码运行的时候，它的 `__name__` 是 `__main__`。如果是被其它代码引用的时候则是它自己的名字。可以利用这一点写测试代码：

```
if __name__ == '__main__':  
#do something
```

这个是很常见的**单元测试**方法之一。

相对引用 (relative import)

```
from .spam import name
```

`spam` 前面的一个点“.”，表示从与当前 module 处于相同 package 的 `spam` module 里 import name 属性。

十六、异常处理

为什么使用异常

错误处理、事件通知、特殊情况处理、退出时的行为、不正常的程序流程。

简单的示例

在没有任何定义 x 变量的时候:

```
print x
print 1
```

将会抛出 `NameError` 异常:

```
NameError: name 'x' is not defined
```

而且1并不会被输出, 也就是说程序将被中断。如果讲代码修改如下:

```
try:
    print x
except NameError:
    print "Something is wrong!"
print 1
```

得到的输出将是:

```
Something is wrong!
1
```

可见, 我们定义的 **except** 会“抓住”**NameError** 类型的语句, 并且执行相应的处理代码, 而且程序不会被中断。

使用 raise

我们可以自己触发异常, 例如:

```
raise IndexError
```

Python 会返回:

```
Traceback (most recent call last):
  File "d:\我的文档\桌面\todo\exep.py", line 1, in <module>
    raise IndexError
IndexError
```

自定义的异常

下面定义了一个 `MyException` 类, 它继承自 Python 内置的 `Exception` 类。

```
class MyException(Exception):pass
try:
```

```
#some code here
raise MyException
except MyException:
    print "MyException encountered"
```

结果为:

```
MyException encountered
```

可以在一个 `except` 内捕获多个异常:

```
except (AttributeError, TypeError, SyntaxError):
```

捕获所有异常

只要在 `except` 后面不添加任何异常类型, 这个 `except` 块就可以捕获所有的异常。

```
except:
```

捕获异常的继承关系

当我们 `except Super` 的时候, 同样会捕获到 `raise Sub` 的异常。

finally

无论 `try` 块是否抛出异常, 永远执行的代码。通常用来执行关闭文件, 断开服务器连接的功能等等。

```
class MyException(Exception):pass
try:
    #some code here
    raise MyException
except MyException:
    print "MyException encountered"
finally:
    print "Arrive finally"
```

结果:

```
MyException encountered
Arrive finally
```

try、except、else

可以在 `try` 块里加入 `else` 块, 代码块将在没有异常被抛出的时候执行:

```
try:
```

```
    print "normal code here"
except MyException:
    print "MyException encountered"
else:
    print "No exception"
finally:
    print "Arrive finally"
```

结果为:

```
normal code here
No exception
Arrive finally
```

raise 异常、同时添加数据

raise 异常的同时，我们可以添加一些额外的数据，就像下面的例子一样：

```
class MyException(Exception):pass
try:
    raise MyException,", and some additional data"
except MyException,data:
    print "MyException encountered"
    print data
```

断言 assert

断言是指期望指定的条件满足，如果不满足则抛出 AssertionError 异常。例如：

```
def positive(x):
    assert x > 0
    print "x"
positive(1)
positive(0)
```

positive(0)一句将会抛出一个异常。

with/as

with/as 语句主要是为了代替 try/finally 语句、通常用来做一些善后工作或者是清理现场的工作。

```
with open("test.txt") as myfile:
    for line in myfile:
        #code here
```

```
#code here
```

当 with 代码块结束之后，文件将会自动关闭。这是因为返回的对象支持 context management protocol。