基本功

**Python语言中没有采用花括号来分割代码块，而是使用了冒号和代码缩进来区分代码之间的层次。**

Ex：

if \_\_name\_\_ == *'\_\_main\_\_'*:

time = 12

if(time==12):

print *'12'*

else:

print *'13'*

**函数与函数之间或者类与类之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。**

Ex：

class **Myclass**:

def **MyFirstClass**(*self*):

print *'MyFirstClass'*

def **MySecondFun**(*self*):

print *'MySecondFun'*

if \_\_name\_\_ == *'\_\_main\_\_'*:

myClass = Myclass()

myClass.MyFirstClass()

myClass.MySecondFun()

**函数命名规则：**

函数名必须以下划线或字母开头，可以包含任意字母、数字或下划线的组合。

函数名是区分大小写的

函数名不能是保留字。（其他命名规则与java类似）

**Java采用//或/\*…\*/作为注释标记，Python中**

1. 单行注释：#
2. 行内注释：行内注释至少用两个空格和语句分开，以#和单个空格开始
3. 注释块：也是用#
4. 中文注释：如果需要在代码中加入中文注释，必须在文件最前端添加 #\_\*\_ coding:utf-8 \_\*\_,否则整的会报错。
5. 跨平台注释：如果程序在windows以外的平台上运行，在文件最前端加!#/user/bin/python

**Python一般用换行作为语句结束的标识，也可以像Java一样用分号，但是不推荐使用。如果一行中写多条语句，则用分号作为结束标识。（不赞成使用）**

**数值类型：**

1. 整型：
2. 布尔型：TRUE为1，FALSE为0，为整型的子类，不能被继承。
3. 标准整数类型：
4. 长整型：从Python2.3开始，就不会出现整型溢出的错误，结果会被自动转换成长整型，即在数字后边加一个L。
5. 双精度浮点型：类似于C语言中的double，可以用十进制或科学计数法表示。每个浮点数占8个字节，64比特。浮点数的表示方法通常使用一个小数点和一个可选的E(大小写都可以)，E和指数之间可以用+或-表示指数的正负。（+可以省略） 1.09E-12
6. 复数：一个实数和一个虚数的组合构成一个复数
7. 十进制浮点数：该类型不是内建类型，必须先导入decimal模块才可以使用这种数值类型。

Ex：

if \_\_name\_\_ == *'\_\_main\_\_'*:

from decimal import Decimal

dec = Decimal(1.1)

print dec

print 1.1

**标识符的命名：**

1. 标识符的第一个字符必须是字母或者下划线
2. 以下划线开头的标识符室友特殊意义的，其中：
3. 以单下划线开头的（\_foo）代表不能直接访问的类的属性，需通过类提供的接口进行访问，也不能用from xxx import \*导入
4. 以双下划线开头的（\_\_foo）代表类的私有成员
5. 以双下划线开头和结尾的（\_\_foo\_\_）代表Python中特殊的方法专用的标识，例如\_\_init\_\_()代表类的构造方法。
6. 标识符的其他部分也可以由字母、下划线、数字组成。
7. 标识符的名称大小写敏感。

**Python是动态类型的语言，不需要预先声明变量的类型。变量的类型和值在赋值的那一刻就被初始化了。且不支持i++，i--，++i，--i。支持i\*=10，i= i\*10。**

**Python创建的变量就是一个对象，Python会管理变量的生命周期。Python对变量的回收也是采用垃圾回收机制。**

**在文件的开头定义全局变量，在函数中引用时，用global，否则会出现不正常的结果。**

Ex

\_num = 12

if \_\_name\_\_ == *'\_\_main\_\_'*:

global \_num

print \_num

**单引号、双引号和三引号字符串**

单引号：使用单引号表示普通的字符串，表示普通的字符串。（不带任何引号的字符串）

if \_\_name\_\_ == *'\_\_main\_\_'*:

print *'大家好，这是 HelloWorld！ '*

大家好，这是 HelloWorld！

双引号：

1. 与单引号中使用方法相同

print *"your name"*

your name

1. 若字符串中使用了单引号，就不能使用单引号包围字符串

print *"What's your name"*

What's your name

1. 当字符串中要出现双引号时，可以在外边加上单引号。

print *'"What is your name?" I say'*

"What is your name?" I say

三引号：可以指示一个多行字符串，并且三引号中的字符串可以自由的使用单引号和双引号。

print *'''"HelloWorld",*

*HiWord*

*'hi' '''*

"HelloWorld",

HiWord

'hi'

**str()：把值转换为合理的形式的字符串，以便用户理解**

repr()：创建一个字符串，以合法的Python表达式的形式表达出来。

**input()是把读入的用户输入默认为Python表达式。**

name = input(*'what\'s your name?'*)

print *'my name is '* + name

what's your name?'1'

my name is 1

raw\_input()是把读入的数据转换为字符串。

name = raw\_input(*'what\'s your name?'*)

print *'my name is '* + name

what's your name?1

my name is 1

**\*\*，x\*\*y求幂运算，2\*\*3=8**

**if语句**

name = *'Happy'*

if name == *'Happy1'*:

print *'111'*

elif name == *"Happy2"*:

print *'222'*

elif name == *'Happy3'*:

print *'333'*

else:

print *'Happy'*

**while循环，当while指定条件为true时，循环执行while块中的语句，否则执行else中的语句。**

i = 1

while i < 5:

print *'我是第'* + str(i) + *'次输出'*

i = i + 1

else:

print *'输出结果完毕'*

**for循环，**

meinv = {

*'赵飞燕'*:*'水色萧前流玉霜，赵家飞燕待昭阳，掌中舞罢箫声绝，三十六宫秋夜长。'*,

*'王昭君'*:*'罗燕北飞情未消，折戟沉沙和两朝，可怜琵琶空对月，万般愁情镜中描。'*

}

for key in meinv:

print key,*'被形容的诗句是：'*,meinv[key]

**迭代器readline和next**

Readline：一次从文件中读取一行文本，当调用readline方法是，就会前进到下一列，知道文件末尾，返回一个空字符串。

Next：当方法执行到文件末尾是，不是返回空字符串，而是印发内置的StopIteration异常。

iter()函数返回一个迭代器，然后使用迭代器的next方法降至输出。

**并行迭代，程序可以同时迭代两个程序。**

names = ['A','B','C','D']

ages = ['21','22','23','24']

for i in range(len(names)):

print names[i],'的年龄是：',ages[i]

names = [*'A'*,*'B'*,*'C'*,*'D'*]

ages = [*'21'*,*'22'*,*'23'*,*'24'*]

for name,age in zip(names,ages):

print name,*'的年龄是：'*,age

**编号迭代，我们使用了内建的enumerate函数来进行编号迭代，其主要作用是在提供索引的地方迭代索引值对。**

strs = [*'星桥鹊起'*,*'经年才见'*,*'想离情、'*,*'别恨难穷'*]

for index,s in enumerate(strs):

if *'星桥鹊起'* in s:

strs[index] = *'金风玉露一相逢，便胜却人间无数'*

for s in strs:

print s

2

fin = raw\_input(*'请输入：'*)

flo = [*'111'*,*'222'*,*'333'*,*'444'*,*'555'*,*'666'*]

for index,f in enumerate(flo):

if fin in f:

print *'find '*+f

**break语句在符合条件的情况下跳出整个循环，continue语句则是跳过符合条件的内容，然后转到循环的顶端继续执行循环。**

在循环中可以使用else子句。

**pass在程序中什么事情都不需要做，类似于一个占位符。**

name = raw\_input(*'输入用户名：'*)

if name == *'111'*:

print *'you are manager:'* + name

else:

pass

**del,一般来说，Python会删除那些不再使用的对象，因为使用者不会再通过任何变量或者数据结果来引用他们。Del语句可以用来删除序列和字典元素，不仅会移除对一个对象的引用，还会移除这个名字本身。**

name = *'HelloWorld'*

del name

print name

（NameError: name 'name' is not defined）

**Python程序其实是有包、模块和函数三者组成的。**

包是由一系列模块组成的集合。模块是处理某一类问题的函数和类的集合。包中必须至少包含一个\_init\_.py文件，该文件的内容可以为空，用于标识当前文件夹是一个包。

**Python可以在函数中默认参数值。**

def **login**(username=*'haha'*,password = *'haha'*):

if (username==*'admin'*) and (password==*'admin'*):

print *'登陆成功！'*

else:

print *'登陆失败！'*

if \_\_name\_\_ == *'\_\_main\_\_'*:

login()

login(username=*'admin'*,password=*'admin'*)

登陆失败！

登陆成功！

**函数的参数可以使变量，也可以是数组、列表等内置数据结构。**

def **login**(usernames = [],password = *'haha'*):

username = usernames[0]

if (username==*'admin'*) and (password==*'admin'*):

print *'登陆成功！'*

else:

print *'登陆失败！'*

if \_\_name\_\_ == *'\_\_main\_\_'*:

login([*'admin'*,*'123'*],*'admin'*)

登陆成功！

**可变长度的参数值。**

在程序开发过程中，常常需要传递可变长度的参数。在函数的参数前使用标识符\*，可以实现这个要求。\*可以引用元祖，将多个参数组合在一个数组中。

def **login**(\* userpwds):

username = userpwds[0]

password = userpwds[1]

if(username==*'admin'*) and (password == *'admin'*):

print *'登陆成功！'*

else:

print *'登陆失败！'*

if \_\_name\_\_ == *'\_\_main\_\_'*:

login(*'admin'*,*'admin'*,*'123'*)

登陆成功！

**字典类型的参数值。**

Python还提供了另一种标识符\*\*，在形式参数前面添加\*\*，可以引用夜歌字典作为参数。根据实际参数的赋值表达式生成字典。

def **login**(\*\*userpwds):

keys=userpwds.keys()

username=*''*

password=*''*

for key in keys:

if *'username'*==key:

username=userpwds[key]

if *'password'*==key:

password=userpwds[key]

if(username==*'admin'*) and (password==*'admin'*):

print *'登陆成功！'*

else:

print *'登陆失败！'*

if \_\_name\_\_ == *'\_\_main\_\_'*:

login(username=*'admin'*,password=*'admin'*)

登陆成功！

**函数返回值，如果需要返回多个值**

如果需要返回多个值，可以将需要返回的值全部打包到元组中。在调用时，对返回的元组解包即可。

def **operat**(x=1,y=1,z=1):

x = 1

y = 2

z = 3

oper=[x,y,z]

numbers = tuple(oper)

return numbers

if \_\_name\_\_ == *'\_\_main\_\_'*:

x,y,z=operat()

print x,y,z

**Python中的from。。。import。。。语句可以将模块中的类或函数导入，从而不需要使用模块名作为前缀。**

from module\_name import function\_name/class\_name

但是不能直接引入某个类下的function

可能会产生代码可读性差的问题。

**\_\_name\_\_属性**

\_\_name\_\_属性用户判断当前模块是不是程序的入口，如果当前程序正在被使用，\_\_name\_\_的值为\_\_main\_\_。

If \_\_name\_\_ == ‘\_\_main\_\_’:

…

else:

…

**\_\_doc\_\_属性**

每个对象都会有\_\_doc\_\_属性，该属性值用于描述该对象的作用。

**33、apply()**

可以实现调用可变参数列表的功能，把函数的参数存放到一个元组或序列中。

**列表（List）**

添加元素：

append(value)

insert(index,value)

扩展元素：

extend()

a.extend(b),a和b为列表

元素赋值：

list[index] = value

删除元素：

List.remove(value)，删除的元素必须存在，否则报错

del list[index]，删除指定索引的元素

分片赋值：

List[m:n]=

list = [1,2,3]

list[2:2] = [4,5,6]

for a in list:

print a

1 2 4 5 6 3

列表的链接:

extend（） list1.estend(list2)

+ list3 = list1+list2

+= list1 += list2

\* list2 = list1\*2(将list1中的内容复制一次)

列表的查找、排序、反转：

list1.index(value) 返回值为value的序号

list1.sort()列表的排序，默认为升序，会改变原来的列表

list1.reverse()列表的反转，会改变原来的列表

pop()

list.pop()，移除最后一个元素，并返回

list.pop(0)，移除第一个元素，并返回

**元组**

元组中的元素被包含在一对圆括号中，元素之间用逗号隔开。

如果元组只包含一个元素，那么该元素后面的逗号是不可忽略的。array = (a,)

创建元组后，其内部的值不能被修改，否则运行时将报错。

Tuple以一个序列作为参数，并把它转换为元组。

元组的遍历：

range()函数返回一个递增或递减的数字列表。

range(start,stop,step)

**字典**

dictionary\_name{key:value,key:value…}

其中key代表键值，value代表value值。字典由多个键值对组成，每个键和其对应的值用冒号：隔开，项与项之间用逗号隔开，整个字典用一对**大括号**括起来。

setdefault()方法可以用来想字典中添加元素。但是如果key已经存在，则其value为原有值。

userDic={*'01'*:*'xiaowan'*,*'02'*:*'xiaoli'*}

print userDic

userDic.setdefault(*'03'*,*'xiaopei'*)

print userDic

userDic.setdefault(*'03'*,*'xiaoma'*)

print userDic

{'02': 'xiaoli', '01': 'xiaowan'}

{'02': 'xiaoli', '03': 'xiaopei', '01': 'xiaowan'}

{'02': 'xiaoli', '03': 'xiaopei', '01': 'xiaowan'}

还可以用以下方式

userDic={*'01'*:*'xiaowan'*,*'02'*:*'xiaoli'*}

print userDic

userDic[*'03'*] = *'xiaoma'*

print userDic

userDic[*'03'*] = *'xiaoli'*

print userDic

这种方式可以修改已经存在key的value

**字典的删除元素**

1、del(userDic[*'01'*]) 或 del userDic[*'02'*]

2、userDic.pop(*'01'*)

**38、字典的遍历**

1、for…in

userDic={*'01'*:*'xiaowan'*,*'02'*:*'xiaoli'*}

userDic[*'03'*] = *'xiaoma'*

for key in userDic:

print *'userDic[%s]='* %key ,userDic[key]

2、items()

userDic={*'01'*:*'xiaowan'*,*'02'*:*'xiaoli'*}

userDic[*'03'*] = *'xiaoma'*

print userDic.items()

3、iteritems(),iterkeys(),itervalues()

iteritems()返回键值序列,相比items（），更加高效。

iterkey()返回所有key

itervalues()返回所有value

zip(userDic.iterkeys(),userDic.itervalues()) == iteritems()

**字典的其他方法**

clear()清楚字典中的所有项

copy()返回值为一个具有相同键值对的新字典

fromkeys()使用给定的键来建立新的字典

userDic={*'01'*:*'xiaowan'*,*'02'*:*'xiaoli'*}

userDic[*'03'*] = *'xiaoma'*

userDic1 = userDic.fromkeys([*'01'*,*'02'*],[*'ma'*,*'li'*])

print userDic1

{'02': ['ma', 'li'], '01': ['ma', 'li']}

get()用于返回字典中的某个元素，如果访问的元素不存在，则返回none

has\_key()可以检查字典中是否含有指定的键，返回值为true或false

popitem()第一次调用会返回第一个元素，第二次调用会返回第二个元素。。。

update()

pop()参数为键，用于获取相应参数的值，然后将这个键值对从字典中删除。

formkeys（）用给定的键建立新的字典，每个键都对应一个默认的None，参数为列表[‘’,’’,’’,’’]

**eval（）**

运行一个包含Python表达式的字符串

num1 = *'1'*

num2 = *'2'*

print eval(num1+*'+'*+num2)

print eval(*'1+2'*)

**ord() and chr()**

Ord()将单个字符转换成ASCII数值

Chr将整数转换成对应的字符。

**42、在程序前加from \_\_future\_\_ import division ，可执行普通除法。用双斜线//可实现整除。**

**43、字符串格式化**

format = *"Hello,%s %s enouth for ya?"*

value=(*"world"*,*"Hot"*)

print format % value

Hello,world Hot enouth for ya?

只有元组和字典可以格式化一个以上的值。其他会被解释为一个值。

substitute()

**字符串常用方法**

find(),查找字串，如果存在返回该子串最左端的索引，否则返回-1。可加入非别为起始点和结束点的参数。

join()用来链接序列中的元素，且要求序列元素都为字符串。

seq = [*'1'*,*'2'*,*'3'*,*'4'*]

sep = *'!'*

print sep.join(seq)

1!2!3!4

表示用sep链接seq。

lower()返回字符串的小写字母

upper（）返回字符串的大写字母

replace（）查找并替换，将字符串中的第一个参数替换成第二个参数。

split（），是join的逆方法，用来将字符串分割为序列。如果不提供任何分隔符（参数），程序会把空格作为分隔符。

strip()返回出去两侧空格的字符串，不包括内部空格

translate（）可以替换字符中的某些部分，但只能处理单个字符。可以同时进行多行替换。

makestrans()函数接收两个参数，两个等长的字符串，表示第一个字符串中的每个字符都用第二个字符串中相同位置的字符替换。

table = maketrans(*'cs'*,*'kz'*)

print len(table)

print *'cscscscsascbbbbbc'*.translate(table);

256

Kzkzkzkzazkbbbbbk

**赋值**

**序列解包**

x,y,z = *'1'*,2,*'b'*

print x,y,z

交换两个值

x,y,z = *'a'*,2,3

x,y = y,x

print x,y

2 a

序列

values=[1,2,3]

x,y,z = values

print x,y,z

**下面的值在作为布尔表达式时，会被解释器看做假（false）**

False, none, 0, ””, (), [], {}

**bool（）函数可以将其他值转换为布尔值。**

**cmp（x,y）用于比较两个对象是否相同，若相同则返回0，否则返回-1.**

**is运算符是判断同一性，而不是相等性，==用于判断相等性。**

**字符串、序列可以按照字母顺序排列进行比较**

print *'abcdef'* < *'bacdef'*

*True*

**列表推导式，利用其他列表创建新列表**

print [x\*x for x in range(10)]

print [x\*x for x in range(10) if x%3==0]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

[0, 9, 36, 81]

**内建的callable函数可以用来判断函数是否可调用。**

**收集参数**

函数中的参数带\*时，意思是手机其余的位置参数，

有\*\*时，表示收集其余的关键字参数。

def **story**(\*\*kwds):

print *'Once upon a time, there was a %(job)s called %(name)s'* %kwds

def **power**(x,y,\*others):

if others:

print *'Received redundant parameters:'*,others

return pow(x,y)

def **interval**(start,stop=None,step=1):

if stop is None:

start, stop = 0, start

re = []

i = start

while i<stop:

re.append(i)

i += step

print re

story(job=*'king'*,name=*'Pei'*)

params = {*'job'*:*'Co'*, *'name'*:*'Pei'*}

story(\*\*params)

del params[*'job'*]

story(job=*'coco'*,\*\*params)

power(3,3,*'Hello,world'*)

interval(10)

interval(1,5)

Once upon a time, there was a Co called Pei

Once upon a time, there was a coco called Pei

Received redundant parameters: ('Hello,world',)

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

[1, 2, 3, 4]

**Choice**

From random import choice

X = choice([‘’,’’,’’,,])

X为choice参数（列表）中的某一项

**Python不需要对类写get & set 方法，属性可以在外部直接访问**

**Python为了让方法或者特性（属性）变为私有，从外部不能访问，只要再它的名字前面加上双下划线\_\_。**

在累的内部定义中，所有以双下划线开始的名字都被“翻译”成前面加上单下划线和类名的形式。所以可以这样继续访问，s=\_ClassName\_\_method()

\_\_metaclass\_\_=type

class **Person**:

\_\_name=*''*

def **setName**(*self*,name):

*self*.\_\_name = name

def **getName**(*self*):

return *self*.\_\_name

def **greet**(*self*):

print *'Hello, world, I am %s'* %*self*.\_\_name

foo = Person()

bar = Person()

foo.setName(*'Pei'*)

bar.setName(*'Wang'*)

print foo.getName()

print bar.getName()

foo.greet()

**print foo.\_Person\_\_name**

**print foo.\_\_name**

Pei

Wang

Hello, world, I am Pei

Pei

AttributeError: 'Person' object has no attribute '\_\_name'

如果不需要使用这种方法，又想让其他对象不访问内部数据，那么可以使用单下划线。例如，前面有下划线的名字都不会被带星号的import语句（from module import \*）导入。

**指定父类**

子类可以扩展父类的定义。将其他类名写在class语句后的圆括号内可以指定父类。

Class subclass（superclass1，superclass2。。。）：

**检查继承**

Issubclass(subclass,superclass)

如果向知道已知类的基类们，

可以Subclass.\_\_bases\_\_

**实例检查**

Isinstance（）

**多个父类时，同名方法的顺序**

如果一个方法从多个父类继承（父类们中有两个具有相同名字的不同方法），先继承的类中的方法会重写后继承的类中的方法。

**hasattr(),getattr(),setattr()，callable（），type（）**

hasattr(object，‘attr’)用于检查对象中是否含有属性（方法）attr

getattr(object,’attr’,None)如果object不含属性（方法）attr。

Callable（method）判断方法是否可用

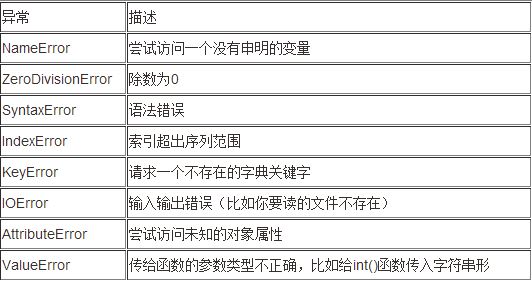
Setattr(object,’name’,’value’) object.name=value

Type(object)返回对象的类型

**dir（）**

dir() 函数可能是 Python 自省机制中最著名的部分了。它返回传递给它的任何对象的属性名称经过排序的列表。如果不指定对象，则 dir() 返回当前作用域中的名称

**python的一些常见异常**



**创建自己的异常类**

class UrExceptionName(Exception): …

**Try/except捕获异常**

try:

x = input('Enter a number')

y = input('Enter a number')

print x/y

except ZeroDivisionError:

print 'The second number can not be zero!'

**如果捕捉到异常，但又不想处理，而是启发它**

可用raise

try:

x = input(*'Enter a number'*)

y = input(*'Enter a number'*)

print x/y

except ZeroDivisionError:

raise

**可以由多个except，一个except中也可以有多个异常，如果想获得异常对象：**

Try:

…

Except (E1,E2…), e(2.0)

Except (E1,E2…) as e(3.0)

…

**如果想捕获所有异常，使没有漏掉的异常，可以在except子句中忽略所有异常类。**

Except:

…

while TRUE:

try:

x = input(*'Enter a number'*)

y = input(*'Enter a number'*)

print x/y

except:

print *'Wrong,try again'*

else:

break;

**Finally**

不管try中是否发生异常，finally中的内容肯定会被执行。

**构造函数**

Python中只能有一个构造函数，def **\_\_init\_\_**(*self, attr, attr, …*):attr为参数，可以没有。实例化一个子类的时候，并不会调用他的父类构造方法，因此也得不到父类实例化时初始化的属性，例如hungr，因此在子类的构造房子法中，还要调用父类的构造方法。可用super函数。

class **Bird**():

def **\_\_init\_\_**(*self*):

*self*.hungry = True

def **eat**(*self*):

if *self*.hungry:

*self*.hungry = False

print *'I am eating...'*

else :

print *'I am not hungry, thanks...'*

class **SongBird**(Bird):

# hungry = True

def **\_\_init\_\_**(*self*):

# Bird.\_\_init\_\_(self)

super(SongBird,*self*).\_\_init\_\_()

*self*.song = *'Hello...'*

def **sing**(*self*):

print *self*.song

sb = SongBird()

sb.sing()

sb.eat()

sb.eat()

**迭代器**

\_\_iter\_\_方法，返回一个迭代器，就是具有next方法（不需要任何参数）的对象。调用next方法时，迭代器会返回它的下一个值。如果next方法被调用，但是迭代器没有值可返回，就睡引发一个StopIteration异常。

**class** Fibs:  
 **def** \_\_init\_\_(self):  
 self.a = 0  
 self.b = 1  
 **def** next(self):  
 self.a,self.b = self.b,self.a + self.b  
 **return** self.a  
 **def** \_\_iter\_\_(self):  
 **return** self  
fibs = Fibs()  
**for** f **in** fibs:  
 **print** f  
 **if** f>1500:  
 **break**

一个实现了\_\_inter\_\_方法的对象是可迭代的，一个实现了next方法的对象是迭代器。

**从迭代器得到序列**

**class** TestIterator:  
 value = 0  
 **def** next(self):  
 self.value +=1  
 **if** self.value >10: **raise** StopIteration  
 **return** self.value  
 **def** \_\_iter\_\_(self):  
 **return** self  
  
li = TestIterator()  
**print** list(li)

**生成器**

Yield。任何包含yield的语句的函数都称为生成器。Yield每次返回多个值。每次产生一个值（使用yield语句），函数就会被冻结，即函数停在那点等待被重新唤醒，函数重新唤醒后，就从停止的那点开始执行。

nested = [[1,2],[3,4],[5,6]]  
**def** flatten(nested):  
 **for** sublist **in** nested:  
 **for** e **in** sublist:  
 **yield** e  
  
**for** num **in** flatten(nested):  
 **print** num

**Python中任何程序都可以作为模块导入。**

Hello.py

**import** sys  
sys.path.append(**'D:/Ainy幸福/PythonTesst'**) *#中文不可识别*

**import** hello

**导入模块的时候，代码别执行了。但再次导入该模块，就什么都不会发生。**

导入模块并不意味着再导入的时候执行某些操作（譬如print）。它们主要用于定义变量、函数、类等。只需要定义这些东西一次，导入模块多次和导入一次效果一样。

**Pprint**

如果数据结构过大，不能在一行打印完成，可以使用pprint模块中的pprint函数代替普通的print语句，能够更加智能的打印输出。

pprint.pprint(sys.path)

D:\Python2.7.8\python.exe D:/Ainy/PythonTest/P1/hello.py

['D:\\Ainy\\PythonTest\\P1',

'D:\\Ainy\\PythonTest',

'C:\\Windows\\system32\\python27.zip',

'D:\\Python2.7.8\\DLLs',

'D:\\Python2.7.8\\lib',

'D:\\Python2.7.8\\lib\\plat-win',

'D:\\Python2.7.8\\lib\\lib-tk',

'D:\\Python2.7.8',

'D:\\Python2.7.8\\lib\\site-packages']

**Python解释器可以从上述目录中找到所需模块。**

site-packages是最佳选择。

**77所示方法并不通用，标准方法是在PYTHONPATH环境变量包含模块所在的目录。**

**为了让Python将其作为包对待，它必须包含一个命名为\_\_init\_\_.py的文件（模块）。**

**可以将包作为普通模块导入**

\_\_init\_\_.py中的内容就只所导入包中的内容。

若P1/\_\_init\_\_.py中包含语句，PI= 3.14。可在其他模块中这么做：

**import** P1  
**print** P1.PI

**探究模块**

模块中有什么

**import** copy  
**print** dir(copy)  
**print** copy.\_\_all\_\_

help获取帮助

help(copy.copy)

=

**print** copy.copy.\_\_doc\_\_

**标准库**

1. os

*environ*用于对环境变量 *进行映射*

*print os.environ['PYTHONPATH']*

D:\Ainy\PythonTest

os.sep路径中的分隔符，pathsep，分隔路径的分隔符（;,::）

**print** os.sep

\

os.linesep文本文件的字符串的分隔符

**print '11111'**+os.linesep+**'222'**

11111

222

os.system(**r'I:\text1.txt'**)用于打开外部程序  
os.startfile(**r'I:\text1.txt'**)win平台下专用打开外部程序函数。

webbrowser模块的open函数，可以打开浏览器，并打开参数总的网址。

1. fileinput
2. 集合、堆、双端队列

集合（Set），Set类位于模块总，但不需要导入sets模块即可使用。

集合是有序列构成的，主要用于检查成员资格，因此副本是呗忽略的：

n = set([1,1,2,2,3,3,4,4])  
**print** n

set([1, 2, 3, 4])

set和字典一样，集合的顺序是随意的。

n = set([**'e'**,**'d'**,**'c'**,**'b'**,**'a'**])  
**print** n

set(['a', 'c', 'b', 'e', 'd'])

其他方法：

与（OR）：c = a.union(b,n)

c = a | b | n

a = set([1,2,3])  
b = set([2,3,4])  
c = a.union(b,n)  
c = a | b | n  
c = a & b  
**print** c.issubset(a)  
**print** c.issuperset(a) #包含  
**print** c.intersection(a) #找出交叉，返回set  
**print** c<b  
**print** c<=b  
**print** a.difference(b)  
**print** c.issuperset(a)  
**print** a.copy()  
**print** c

堆，优先队列的一种，能够以任意顺序增加对象，能在任意时间找到最小元素，比列表的min方法要高效的多。

Heappush：用于增加堆的项，

**import** heapq  
**import** random  
  
data = range(10)  
**print** data  
random.shuffle(data)  
heap = []  
**for** n **in** data:  
 heapq.heappush(heap,n)  
**print** heap  
heapq.heappush(heap, 0.5)  
**print** heap

元素的顺序虽然看起来有些随意，但是也有规则：位于i位置上的元素总比i/2位置上的元素大，也叫堆属性。

Heappop:用于弹出最小元素。

**mport** heapq  
**import** random  
  
data = range(10)  
**print** data  
random.shuffle(data)  
heap = []  
**for** n **in** data:  
 heapq.heappush(heap,n)  
**print** heap  
heapq.heappush(heap, 0.5)  
**print** heap  
  
n = heapq.heappop(heap)  
**print** n  
n = heapq.heappop(heap)  
**print** n  
n = heapq.heappop(heap)  
**print** n

**print** heap

：

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

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

[0, 0.5, 4, 2, 1, 8, 5, 9, 6, 7, 3]

0

0.5

1

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

该函数操作过程中，会进行位移操作。

Heapify：该函数参数为任意列表，并且尽可能少的位移，将其转换为合法的堆（符合对属性）。如果没有heappush建立堆，那么在使用heappush和heappop前，应该使用该函数。

Heapreplace不像其他函数那么常用。它弹出堆的最小元素，并将新元素推入堆中。Heappop+heappush。

n = heapq.heapreplace(heap,100)  
**print** n  
**print** heap

双端队列，在需要按照元素增加的顺序来移除元素时非常有用。

**from** collections **import** deque  
q = deque(range(5))  
**print** q  
q.append(6)  
**print** q  
q.appendleft(5)  
**print** q  
q.pop()  
**print** q  
q.popleft()  
**print** q  
q.rotate(3)  
**print** q

：

deque([0, 1, 2, 3, 4])

deque([0, 1, 2, 3, 4, 6])

deque([5, 0, 1, 2, 3, 4, 6])

deque([5, 0, 1, 2, 3, 4])

deque([0, 1, 2, 3, 4])

deque([2, 3, 4, 0, 1])

1. Time

**import** time  
**print** time.asctime() *#将当前时间格式化为字符串***print** time.localtime(0) *#将秒数转换为日期元组，以本地时间为准***print** time.gmtime() *#获得当前全球统一时间*time.sleep(10) *#让解释器等待给定的秒数***print** time.strptime(**'Tue Mar 31 15:25:53 2015'**) *#将asctime格式化过的字符串转换为日期元组。***print** time.time() *#新纪元开始后的秒数*

：

Tue Mar 31 15:28:52 2015

time.struct\_time(tm\_year=1970, tm\_mon=1, tm\_mday=1, tm\_hour=8, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=1, tm\_isdst=0)

time.struct\_time(tm\_year=2015, tm\_mon=3, tm\_mday=31, tm\_hour=7, tm\_min=28, tm\_sec=52, tm\_wday=1, tm\_yday=90, tm\_isdst=0)

time.struct\_time(tm\_year=2015, tm\_mon=3, tm\_mday=31, tm\_hour=15, tm\_min=25, tm\_sec=53, tm\_wday=1, tm\_yday=90, tm\_isdst=-1)

1427786942.94

1. Random

**import** random  
**print** random.random() *#返回0~1之间的随即实数***print** random.getrandbits(20) *#以长整型形式返回指定字节数的随机实数***print** random.uniform(0,360) *#随即返回所提供的参数a和参数b之间的实数***print** random.randrange(1,100,2) *#返回rangge（start,stop,step）中的随机数***print** random.choice([**'w'**,**'e'**,**'r'**,**'c'**,**'q'**]) *#从给定序列中选择随机元素*s =[**'w'**,**'e'**,**'r'**,**'c'**,**'q'**]  
random.shuffle(s) *#将给定序列元素进行随即移位***print** s  
**print** random.sample(s,3) *#从序列中选择n个随机且独立的元素*

：

0.316138956569

954699

90.7243118022

39

c

['q', 'e', 'c', 'r', 'w']

['e', 'q', 'w']

**import** random  
**import** time  
deta1 = (2008,1,1,0,0,0,-1,-1,-1)  
time1 = time.mktime(deta1)  
deta2 = (2009,1,1,0,0,0,-1,-1,-1)  
time2 = time.mktime(deta2)  
t = random.uniform(time1,time2)  
**print** t  
**print** time.asctime(time.localtime(t))

:

1226603929.68

Fri Nov 14 03:18:49 2008

1. Shelve
2. Re

该模块包含对正则表达式的支持。

**re.compile(r'[a-z]'),**根据包含正则表达式的字符串创建模式对象。

**re.search(r'p','peihongyue')**，在字符串中寻找模式，如果找到返回MatchObject（值为True）,否则返回None（值为False）

**re.match(r'p','app')**在给定字符串的开头匹配正则表达式。如果找到返回MatchObject（值为True）,否则返回None（值为**False**）

**re.split(r’p’,’rpei’)**根据模式的匹配来分割字符串。其中maxsplit参数为指定最多可以分割的次数。

**re.findall(r'[a-z]+','hello, world')**

**re.sub(r'name','Mr Pei','name, how are you!')**将字符串中多有pat的匹配项用第二个参数替换。

**打开文件**

open函数可弄过来打开文件，open(name, mode, buffering)

第一个参数为文件的路径，如’C:\text\somefile.txt’

第二个参数为模式，有读模式（r），写模式（w），追加模式（a），二进制模式（b），读/写模式（+）。如果不指定模式参数，则默认为读模式。

第三个参数控制着文件的缓冲。如果参数是0（或者False），I/O就是无缓冲（所有读写操作都直接针对硬盘），如果是1（或者是True），I/O就是有缓冲的（Python会用内存代替硬盘，让程序更快，只有flush或close是，才会更新到硬盘上的数据）大于1的数字代表缓冲区的大小（单位是字节），-1（或者其他负数）代表使用默认的缓冲区大小。

**Python对文本的操作**

唯一要用到的技巧就是标准化换行符。(\n)表示结束一行并另起一行。

**读和写**

文件最重要的能力就是提供或者接受数据。

f = open(**r'D:\1s\test.txt'**,**'w'**)  
f.write(**'hello, world'**)  
f.write(**'. '**)  
f.close()

f = open(**r'D:\1s\test.txt'**,**'r'**)  
**print** f.read()  
f.close()

read函数可以指定读多少字符（字节）。

**管道符号（|）**

管道符号将一个命令的标准输出和下一个命令的标准输入连在一起。

**随机访问**

使用类文件对象的方法seek和tell来直接访问感兴趣的部分（这种做法称为随机访问）。

f.seek(5,0)

第一个参数表示偏移量。第二个参数，默认是0，表示偏移量是从文件头开始计算的(偏移量必须是非负的)，若设置为1，表示相对于当前位置的移动，此时偏移量可以为负值，若为2，表示相对于文件结尾移动。

f = open(**r'D:\1s\test.txt'**,**'w'**)  
f.write(**'012345678901234567890'**)  
f.seek(5,0)  
f.write(**'H0'**)  
f.seek(5,1)  
f.write(**'H1'**)  
f.seek(5,2)  
f.write(**'H2'**)  
f.close()  
f = open(**r'D:\1s\test.txt'**,**'r'**)  
**print** f.read()

：

01234H078901H14567890 H2

tell方法韩慧当前文件的位置。

**读写行**

**print** f.readline(2)读取单独一样，参数为可读取字符的最大值。

**print** f.readlines()可以读取一个文件中的所有行，并将其作为列表返回。

f.writelines（）和readlines方法相反，参数为一个字符串列表（其他序列或者可迭代的列表也行），它会把所有字符串写入文件。

**关闭文件**

为了确保文件被关闭，应该使用try/finally语句，并且在finally子句中调入close方法。

**with** open(**r'D:\1s\test.txt'**,**'r'**) **as** test:  
 **print** test.readlines()

同样可以可是在语句结束的时候，关闭文件。

如果想在关闭之前，将更新写到磁盘上，（数据可能被缓存），可用文件对象的flush方法。

**数据库编程接口（API）**

* 全局变量

任何支持2.0版本DB API的数据库模块都必须定义3个描述模块特性的全局变量。这样做的原因是API设计得很灵活，以支持不同的底层机制、避免过多的包装。

apilevel，API级别，字符串常量，提供正在使用的API版本号。

threadsafety线程安全等级。取值范围0~3的整数。0表示线程完全不共享。1表示线程本身可以共享，但不对链接共享。3表示模块是完全线程安全的。

paramstyle，参数风格，表示在执行多次类似查询的时候，参数是如何被拼接到SQL查询中的。值’format’表示标准的字符串格式化（使用基本的格式代码），可以在参数中进行拼接的地方插入%S。值’pyformat’表示扩展的格式代码，用于字典拼接中，比如%(foo)。’qmark’的意思是使用问好，而’numeric’表示使用:1或者:2格式的字段（数字表示参数的序号）。’named’表示:foobar这样的字段，其中foobar为参数名。

|  |  |
| --- | --- |
| qmark | 表示question-mark风格。指令字符串中的数据的每一位都被用一个问号替换，参数以list或tuple的形式给出。例如：INSERT INTO ch14 VALUES (?, ?)。 |
| format | 使用和printf()一样的类型格式，不支持对于指定参数Python的扩展名。它带一个list或tuple来转换。例如：INSERT INTO ch14 VALUES(%d, %s) |
| numeric | 表示numeric风格。指令字符串中的数据的每一位都被一个后面是数字的冒号替换（数字以1开始），参数以list或tuple的形式给出。例如：INSERT INTO ch14 VALUES(:1, :2) |
| named | 表示named风格。和numeric类似，但是在冒号后面用名称取代数字。带一个dictionary用来转换。例如：INSERT INTO ch14 VALUES(:number, :text) |
| pyformat | 支持Python风格的参数，带dictionary用来转换。例如：INSERT INTO ch14 VALUES(%(number)d, %(text)s)。 |

* 异常

=========================================================

异常 超类 描述

=========================================================

StandarError 所有异常的泛型基类

Waring StandarError 在非致命错误发生时引发

Error StandarError 所有错误条件的泛型超类

InterfaceError Error 关于接口而非数据库的错误

DatabaseError Error 与数据库相关的错误的基类

DataError DatabaseError 与数据库相关的问题，比如值超出范围

OperationalError DatabaseError 数据库内部操作错误

IntegrityError DatabaseError 关系完整性受到影响，比如键检查失败

InternalError DatabaseError 数据库内部错误，比如非法游标

ProgramningError DatabaseError 用户编程错误，比如未找到表

NotSupportedError DatabaseError 请求不支持的特性(比如回滚)

* 连接和游标

在适当的环境下使用具名函数connect，可以连接数据库。该函数有多个参数，而具体使用哪个参数取决于底层数据库的类型。

|  |  |
| --- | --- |
| 参数 | 说明 |
| user | 用户名，默认为当前登录用户。 |
| passwd | 用户密码，没有默认的。 |
| db | 连接的数据库名。 |
| host | 数据库主机名。 |
| port | TCP端口，默认是3306。 |

connect函数返回连接对象。这个对象表示目前和数据库的绘画。该对象支持的方法：

|  |  |
| --- | --- |
| close() | 关闭连接之后，连接对象和它的游标均不可用 |
| commit() | 如果支持的话，就提交挂起的事务，否则不做任何事 |
| rollback() | 回滚挂起的事务（可能不可用） |
| cursor() | 返回链接的游标对象 |

cursor方法返回一个游标对象。通过游标执行SQL查询并检查结果。游标对象的方法有：

callproc(name[, params]) 使用给定的名称和参数(可选)调用以命名的数据库程序 close() 关闭游标后，游标不可用

execute(oper[, params]) 执行SQL操作，可选的参数

executemany(oper, pseq) 对序列中的每个参数执行SQL操作

fetchone() 把查询结果集中的下一行保存为序列或者None

fetchmany([size]) 获取查询的结果集中的多行，默认尺寸为arraysize

fetchall() 将所有(剩余)的行作为序列的序列

nextset() 跳至下一个可用的结果集(可选)

setinputsizes(sizes) 为参数预先定义的内存区域

setoutputsize(size[, col]) 为获取的大数据库值设定缓冲区尺寸

游标对象的特性

Description，结果列描述的序列，只读

Roucount，结果中的行数，只读

Arraysize，fetchmany中返回的行数，默认为1。

* 类型

DB API 定义了Python中的类型和数据库类型的对应关系。通过特殊的类型和值的构造函数以及常量(单例模式)，如下(一些模块可能并不是完全按照这些定义)：

=========================================================================

名称 描述

========================================================================

Date(year, month, day) 创建保存日期值的对象

Time(hour, minute, second) 创建保存时间值的对象

Timestamp(y, mon, d, h, min, s) 创建保存时间戳的对象

DateFromTicks(ticks) 创建保存自新纪元以来的秒数的对象

TimeFromTicks(ticks) 创建保存来自秒数的时间值的对象

TimestampFromTicks(ticks) 创建保存来自秒数的时间戳值的对象

Binary(string) 创建保存二进制字符串值的对象

STRING 描述二进制列(比如LONG或RAW)

BINARY 描述数字列

MUMBER 描述数字列

DATETIME 描述日期/时间列

ROWID 描述行ID列

* 举例

**def** convert(value):  
 **if** value.startswith(**r'~'**):  
 **return** value.strip(**r'~'**)  
 **if not** value:  
 **return '0'  
 return** float(value)  
  
conn = sqlite3.connect(**'food.db'**)  
curs = conn.cursor()  
curs.execute(**'''  
 create table food(  
 id text PRIMARY KEY,  
 desc text,  
 water FLOAT,  
 kcal FLOAT ,  
 protein FLOAT ,  
 fat FLOAT ,  
 ash FLOAT,  
 carbs FLOAT,  
 fiber FLOAT,  
 sugar FLOAT  
 )  
'''**)  
  
query = **'INSERT INTO food VALUES (?,?,?,?,?,?,?,?,?,?)'  
  
for** line **in** open(**r'D:\1s\ABBREV.txt'**):  
 fields = line.split(**'^'**)  
 values = [convert(f) **for** f **in** fields[:10]]  
 curs.execute(query,values)  
query = **'select \* from food'**curs.execute(query)  
names = [f[0] **for** f **in** curs.description]  
**for** row **in** curs.fetchall():  
 **for** pair **in** zip(names,row):  
 **print** pair  
  
conn.commit()  
conn.close()

**Socket模块**

在网络编程中，一个基本的组件就是套接字（socket）。套接字基本上是两个端点的程序之间的信息通道。

socket的实例化需要3个参数：第一个参数是地址族（默认为socket.AF\_INEF），第二个参数是流（默认值是socket.SOCK\_STREAM）或数据报（默认socket.SOCK\_DGRAM）套接字，第三个参数是使用的协议（默认是0）。

服务器端套接字使用blind方法后，再调用listen方法去监听某个特定的地址。客户端套接字使用connect方法连接到服务器，在connect方法中使用的地址与服务器在blind方法中的地址相同（在服务器端，能实现很多功能，比如使用函数socket.gethostname得到当前主机名）。在这种情况下，一个地址就是一个格式为（host， port）的元组，host为主机名，port是端口号。listen方法只有一个参数，即服务器未处理的连接长度（允许排队等待的连接数目，这些连接在禁用之前等待）。

服务器点套接字在开始监听后，就可以接受客户端的连接。这个步骤使用accept方法来完成。这个方法会阻塞，知道客户端连接，然后该方法就返回一个格式为（client, address）的元组，client是一个客户端套接字，address是一个前面解释过的地址。服务器在处理完与该客户端的连接后，在调用accept方法开始等待下一个连接(在无限循环中实现)。

**服务器端：**

**import** socket  
s = socket.socket()  
host = socket.gethostname()  
port = 8888  
s.bind((host, port))  
  
s.listen(5)  
**while** True:  
 c, addr = s.accept()  
 **print 'Got connection from '**, addr  
 c.send(**'Thank you for connectiong'**)  
 c.close()

**客户端：**

**import** socket  
s = socket.socket()  
s.connect((socket.gethostname(), 8888))  
**print** s.recv(1024)

**urllib和urllib2**

urllib2更好一下，如果只是简单的下载，urllib就足够了。如果使用http验证或cookie，或者要为自己的协议编写扩展程序，urllib2会更好一些。

**import** urllib  
webpage = urllib.urlopen(**r'http://www.baidu.com'**)  
urllib.urlretrieve(**r'http://www.baidu.com'**, **r'D:\1s\baidu.html'**)  
urllib.urlcleanup()

urlopen参数可以为file： D:\，可以打开本地文件。

Urlretrieve返回一个元组（filename，headers），filename是本地文件名（由urllib自动创建），header包含一些远程文件的信息。第二个参数为要下载的副本的指定文件名。如果没有指定，文件就会放在临时位置，用open函数打开它，如果完成了对它的操作，就可以删除它以节省磁盘空间。可调用urlcleanup()。

**使用SocketServer进行分叉和线程处理**

分叉：

**import** SocketServer  
**class** Server(SocketServer.ForkingMixIn, SocketServer.TCPServer):**pass  
  
class** Handler(SocketServer.StreamRequestHandler):  
  
 **def** handle(self):  
 addr = self.request.getpeername()  
 **print 'Gog connection from '**,addr  
 self.wfile.write(**'Thank you for connectiong'**)  
  
server = Server((**''**,8888), Handler)  
server.serve\_forever()

线程：

**import** SocketServer  
**class** Server(SocketServer.ThreadingMixIn, SocketServer.TCPServer): **pass  
  
class** Handler(SocketServer.StreamRequestHandler):  
  
 **def** handle(self):  
 addr = self.request.getpeername()  
 **print 'Got connection from '**,addr  
 self.wfile.write(**'Thank you for connecting'**)  
  
server = Server((**''**,8888),Handler)  
server.serve\_forever()

**Tidy和XHTML解析**

Tidy是用来修复不规范且有些随意的HTML文档的工具。

**测试工具doctest**

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **import** doctest, hello  
 doctest.testmod(hello)

hello为另一模块

**Unittest**

Unittest.main函数负责实际运行测试。它会实例化所有TestCase的子类，**运行所有名字以test开头的方法**。另外，如果定义了叫做setUp和tearDown的方法，它们就会在运行每个测试方法之前和之后执行，这样就可以用这些方法为所有的测试提供一般的初始化和清理代码。

**import** unittest,my\_math  
  
**class** ProductTestCase(unittest.TestCase):  
  
 **def** testIntegers(self):  
 **for** x **in** xrange(-10, 10):  
 **for** y **in** xrange(-10, 10):  
 p = my\_math.product(x, y)  
 self.failUnless(p == x\*y,**'wrong'**)  
 **def** testFloats(self):  
 **for** x **in** xrange(-10, 10):  
 **for** y **in** xrange(-10, 10):  
 x = x/10  
 y = y/10  
 p = my\_math.product(x, y)  
 self.failUnless(p == x\*y, **'error'**)  
  
 **def** setUp(self):  
 **print**(**'begin'**)  
 **def** tearDown(self):  
 **print**(**'end'**)  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

#my\_math.py

**def** product(x, y):  
 **return** x\*y  
 *# pass*

**性能测试profile**

只需要运行profile模块的run方法即可。

**import** profile, my\_math  
profile.run(**'my\_math.square(100)'**)

结果如下：

D:\Python2.7.8\python.exe D:/Ainy/PythonTest/P1/T1.py

4 function calls in 0.001 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.001 0.001 0.001 0.001 :0(setprofile)

1 0.000 0.000 0.000 0.000 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 my\_math.py:3(square)

1 0.000 0.000 0.001 0.001 profile:0(my\_math.square(100))

0 0.000 0.000 profile:0(profiler)

Process finished with exit code 0

**扩展python**

Jython & IronPython (fail)

**程序打包**

Distutils的安装脚本（setup.py）

**from** distutils.core **import** setup  
setup(  
 name='Hello',  
 version='1.0',  
 description='A simple example',  
 author='Pei',  
 py\_modules=['my\_math']  
)

要确保在同一目录下存在名为my\_math.py的文件。

在dos中输入python setup.py，运行该文件，会得到类似下面的输出：

D:\Ainy\PythonTest\P1>python T1.py

usage: T1.py [global\_opts] cmd1 [cmd1\_opts] [cmd2 [cmd2\_opts] ...]

or: T1.py --help [cmd1 cmd2 ...]

or: T1.py --help-commands

or: T1.py cmd –help

输入python setup.py build，会得到类似下面的输出：

D:\Ainy\PythonTest\P1>python T1.py build

running build

running build\_py

copying my\_math.py -> build\lib

会在该目录下自动创建一个名为build的子文件夹。

在安装的时候不需要运行build命令，如果需要的话，运行install命令的时候他就自动运行了。

**打包**

建立存档文件：运行python setup.py sdist，可用于源代码发布。如果运行脚本，会得到一堆输出，其中还有一些警告。我们可以忽略这些警告。然后会生成MANIFEST的文件，并且声称build和dist子文件。

创建windows安装程序或RPM包：使用bdist命令可以创建单一的windows安装程序和RPM包，一般可以用它来发布二进制程序包，其中的扩展已经由特定的结构而被编译。输入

D:\Ainy\PythonTest\P1>python T1.py bdist --formats=wininst即可在dist文件夹内生成exe文件。

还可以使用py2exe创建可执行文件。

**使用ConfigParser读取配置文件。**

**from** ConfigParser **import** ConfigParser  
CONFIGFILE = 'test1.txt'  
config = ConfigParser()  
config.read(CONFIGFILE)  
**print** config.get('message', 'greeting')  
radius = raw\_input(config.get('message', 'question') + '')  
**print** config.get('message', 'result\_message')  
**print** config.getfloat('numbers', 'pi')

其中test1.txt中内容为：

[numbers]

pi: 1234567890

[message]

greeting: Welcome to China

question: Where are you come from

result\_message: US

**使用logging记录日志**

**import** logging  
logging.basicConfig(level=logging.INFO, filename='mylog.phy')  
logging.info('Starting program')  
logging.info('Try to divide 1 by 0')  
**print**(1/0)  
logging.info('The division succeeded')  
logging.info('Ending program')

可生成mylog.phy文本文件，用于记录日志。