字符编码

#--coding--utf-8--

Keywords（关键字）

and

del 

from 

not 

while 

as 

如果想要改变被导入模组在当前模组中的名称，而不是sys.modules中的名称。可以使用import as，例如：

import some as other

print(other.name)

elif 

global 

将变量定义为全局变量。可以通过定义为全局变量

>>> def func():

...     global x

...     print 'x is ', x

...     x = 2

...     print 'Change local x to ', x

...

x = 50

>>> func()

x is  50

Change local x to  2

or 

with 

除了有更优雅的语法，with还可以很好的处理上下文环境产生的异常。下面是with版本的代码。

with open("/tmp/foo.txt") as file:

     data = file.read()

while this might look like magic, the way Python handles with is more clever than magic. The basic idea is that the statement after with has to evaluate an object that responds to an \_\_enter\_\_() as well as an \_\_exit\_\_() function. 这看起来充满魔法，但不仅仅是魔法，Python对with的处理还很聪明。基本思想是with所求值的对象必须有一个\_\_enter\_\_()方法，一个\_\_exit\_\_()方法。

After the statement that follows with is evaluated, the \_\_enter\_\_() function on the resulting object is called. The value returned by this function is assigned to the variable following as. After every statement in the block is evaluated, the \_\_exit\_\_() function is called. 紧跟with后面的语句被求值后，返回对象的\_\_enter\_\_()方法被调用，这个方法的返回值将被赋值给as后面的变量。当with后面的代码块全部被执行完之后，将调用前面返回对象的\_\_exit\_\_()方法。

assert 

assert语句用来声明某个条件是真的。

如果你非常确信某个你使用的列表中至少有一个元素，而你想要检验这一点，并且在它非真的时候引发一个错误，那么assert语句是应用在这种情形下的理想语句。

当assert语句失败的时候，会引发一AssertionError。

else 

if 

pass 

空语句 do nothing

保证格式完整

保证语义完整

以if语句为例

if true:

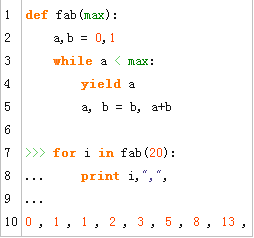
pass # do nothing

else:

print "do something."

yield 

生成器函数在Python中与迭代器协议的概念联系在一起。简而言之，包含yield语句的函数会被特地编译成生成器。**当函数被调用时，他们返回一个生成器对象，这个对象支持迭代器接口。函数也许会有个return语句，但它的作用是用来yield产生值的。**不像一般的函数会生成值后退出，生成器函数在生成值后会自动挂起并暂停他们的执行和状态，他的本地变量将保存状态信息，这些信息在函数恢复时将再度有效。



如果使用生成器就不需要返回整个列表，每次都只是返回一个数据，避免了内存的限制问题。

def get\_primes(number):

while True:

if is\_prime(number):

yield number

number += 1

break 

except 

import

print 

class 

exec 

exec语句用来执行储存在字符串或文件中的Python语句。例如，我们可以在运行时生成一个包含Python代码的字符串，然后使用exec语句执行这些语句。下面是一个简单的例子。

exec 'print "Hello World"'

Hello World

in 

raise 

使用raise抛出异常

当程序出现错误，python会自动引发异常，也可以通过raise显示地引发异常。一旦执行了raise语句，raise后面的语句将不能执行。

演示raise用法

try:

s = None

if s is None:

print "s 是空对象"

raise NameError #如果引发NameError异常，后面的代码将不能执行

print len(s)

except TypeError:

print "空对象没有长度"

continue 

finally 

is 

Python中的对象包含三要素：id、type、value

其中id用来唯一标识一个对象，type标识对象的类型，value是对象的值

is判断的是a对象是否就是b对象，是通过id来判断的

==判断的是a对象的值是否和b对象的值相等，是通过value来判断的

return 

def 

for 

lambda 

lambda [arg1[,arg2,arg3....argN]]:expression

lambda语句中，冒号前是参数，可以有多个，用逗号隔开，冒号右边的返回值。lambda语句构建的其实是一个函数对象。

sum2 = lambda x,y=10:x+y

print sum2(1) #11

print sum2(1,100) #101

try

try...except的使用方法

try...except用于处理问题语句，捕获可能出现的异常。try子句中的代码块放置可能出现异常的语句，except子句中的代码块处理异常。

演示try...except语句捕获IOError异常

try：

file("hello.txt", "r") #如果文件不存在，引发异常

print "读文件"

except IOError: #捕获IO异常

print "文件不存在"

except： #其它异常

print "程序异常"

数据类型

True 

False 

None 

None是一个特殊的常量。

None和False不同。

None不是0。

None不是空字符串。

None和任何其他的数据类型比较永远返回False。

None有自己的数据类型NoneType。

可以将None复制给任何变量，但是你不能创建其他NoneType对象。

strings 

str为字符串

str.isalnum() 所有字符都是数字或者字母

str.isalpha() 所有字符都是字母

str.isdigit() 所有字符都是数字

str.islower() 所有字符都是小写

str.isupper() 所有字符都是大写

str.istitle() 所有单词都是首字母大写，像标题

str.isspace() 所有字符都是空白字符、\t、\n、\r

numbers 

包含int long float complex byte

floats 

lists

L.append(var) #追加元素

L.insert(index,var)

L.pop(var) #返回最后一个元素，并从list中删除之

L.remove(var) #删除第一次出现的该元素

L.count(var) #该元素在列表中出现的个数

L.index(var) #该元素的位置,无则抛异常

L.extend(list) #追加list，即合并list到L上

L.sort() #排序

L.reverse() #倒序

list 操作符:,+,\*，关键字del

a[1:] #片段操作符，用于子list的提取

[1,2]+[3,4] #为[1,2,3,4]。同extend()

[2]\*4 #为[2,2,2,2]

del L[1] #删除指定下标的元素

del L[1:3] #删除指定下标范围的元素

list的复制

L1 = L #L1为L的别名，用C来说就是指针地址相同，对L1操作即对L操作。函数参数就是这样传递的

L1 = L[:] #L1为L的克隆，即另一个拷贝。

list comprehension

[ <expr1> for k in L if <expr2> ]

字符串转义序列(Escape Sequences)

\ \(在行尾时) 续行符

\\  反斜杠符号

\'  单引号

\"  双引号

\a  响铃

\b  退格(Backspace)

\f 换页

\n  换行

\r  回车

\t  横向制表符

\v 纵向制表符

Python中一些基本数据类型：

1. Booleans［布尔型］ 或为 True［真］ 或为 False［假］。
2. Numbers［数值型］ 可以是 Integers［整数］（1 和 2）、Long、Floats［浮点数］（1.1 和 1.2）、Fractions［分数］（1/2 和 2/3）；甚至是 Complex Number［复数］。
3. Strings［字符串型］ 是 Unicode 字符序列，例如： 一份 html 文档。
4. Bytes［字节］ 和 Byte Arrays［字节数组］， 例如: 一份 jpeg 图像文件。
5. Lists［列表］ 是值的有序序列。
6. Tuples［元组］ 是有序而不可变的值序列。
7. Sets［集合］ 是装满无序值的包裹。
8. Dictionaries［字典］ 是键值对的无序包裹。

字符串格式化（String Formats）

%d 有符号整数(十进制)

%i  有符号整数(十进制)

%o  无符号八进制整数

%u  无符号整型（十进制）

%x  无符号十六进制整数

%X  格式化无符号十六进制数（大写）

%e  指数 (基底写为e)

%E  指数 (基底写为E)

%f  浮点数

%F  浮点数，与上相同

%g  指数(e)或浮点数 (根据显示长度)

%G  指数(E)或浮点数 (根据显示长度)

%c  格式化字符及其ASCII码

%r  字符串 (采用repr()的显示)

%s  字符串 (采用str()的显示)

%% 字符"%"

操作符号

+ 

- 

\* 

\*\* 

/ 

// 

%

< 

> 

<= 

>= 

== 

!= 

<> 

( ) 

[ ] 

{ } 

@ 

Decorator,修饰符 <http://www.artima.com/weblogs/viewpost.jsp?thread=240808>

1、Use class as decorator

class entryExit(object):

def \_\_init\_\_(self,f):

print "In the exitExit"

self.f = f

def \_\_call\_\_(self):

print "Entering", self.f.\_\_name\_\_

self.f()

print "Exited", self.f.\_\_name\_\_

@entryExit

def func1():

print "inside func1()"

@entryExit

def func2():

print "inside func2()"

func1()

func2()

2、Use function as decorator

def entryExit(f):

def new\_f():

print "Entering", f.\_\_name\_\_

f()

print "Exited", f.\_\_name\_\_

return new\_f

@entryExit

def func1():

print "inside func1()"

@entryExit

def func2():

print "inside func2()"

func1()

func2()

print func1.\_\_name\_\_

, 

: 

. 

= 

; 

语句结尾一般不需要分号”;” 在Python中，一般来说一条语句占用一行。一条语句结束一般不需要使用”;”，但在Python中也可以使用”;”将两条语句写在一行。当一行很长时，Python可以用 \ 符号折行显示代码。

+= 

-= 

\*= 

/= 

//= 

%= 

\*\*=