for…else和while…else语法

Python中的for、while循环都有一个可选的else分支（类似if语句和try语句那样），在循环迭代正常完成之后执行。
如果循环不是以除正常方式以外的其他任意方式退出循环，那么else分支将被执行。
也就是在循环体内没有break语句、没有return语句，或者没有异常出现。考虑一个简单的（无用的）例子：

```
# -*- coding: utf-8 -*-
if __name__=='__main__':
    sum=0
    for i in range(10):
        sum+=i
    else:
        #以正常的方式退出循环else分支执行
        print "First Loop Run ELSE!"
    for i in range(10):
        if(i==2):
            break
    else:
        #以break方式退出循环else分支不执行
        print "Second Loop Run ELSE!"
    for i in range(1,1):
        sum+=i
    else:
        # 空循环else分支执行
        print "Third Loop Run ELSE!"
```

输出结果：

```
D:\Python27\python.exe D:/Projects/PythonLearn/learn2.py
First Loop Run ELSE!
Third Loop Run ELSE!

Process finished with exit code 0
```

##WITH的语法
with语法旨在对try/finally语法进行增强，使python代码可以进行执行前的准备动作，

以及执行后的收尾动作。
在执行后的收尾动作可以通过try/catch语法实现，执行前的预处理通过try/catch语法无法实现，需要通过with语句。
Python通过上下文管理器（Context Manager）来定义/控制代码块执行前的准备动作，以及执行后的收尾动作。

上下文管理协议（Python中的接口）

enter() 和 **exit**()，协议的对象要实现这两个方法。
enter()
进入上下文管理器的运行时上下文，在语句体执行前调用。with 语句将该方法的返回值赋值给 as 子句中的 target，如果指定了 as 子句的话。
exit(exc_type, exc_value, exc_traceback)
退出与上下文管理器相关的运行时上下文，返回一个布尔值表示是否对发生的异常进行处理。
参数表示引起退出操作的异常，如果退出时没有发生异常，则3个参数都为None。
如果发生异常，返回True 表示不处理异常（可以吃异常！！！），否则会在退出该方法后重新抛出异常以由 with 语句之外的代码逻辑进行处理。要处理异常时，不要显示重新抛出异常，即不能重新抛出通过参数传递进来的异常，只需要将返回值设置为 False 就可以了。之后，上下文管理代码会检测是否 **exit**() 失败来处理异常
如果该方法内部产生异常，则会取代由 statement-body 中语句产生的异常。

使用with/as

下面的代码中with/as语句中，打开了一个文件，open方法返回了file类型，该类型事项上下文管理协议，是file类型对应的文件句柄在使用完之后能够自动关闭。

```
    with open(r'somefileName') as somefile:
        for line in somefile:
            print line
```

自定义支持with/as语句的class

```
class DummyResource:
    def __init__(self, tag):
```

```
            self.tag = tag
            print 'Resource [%s]' % tag
    def __enter__(self):
        print '[Enter %s]: Allocate resource.' % self.tag
        return self      # 可以返回不同的对象
    def __exit__(self, exc_type, exc_value, exc_tb):
        print '[Exit %s]: Free resource.' % self.tag
        if exc_tb is None:
            print '[Exit %s]: Exited without exception.' % self.tag
        else:
            print '[Exit %s]: Exited with exception raised.' % self.tag
            return False   # 可以省略，缺省的None也是被看做是False，True时将
with DummyResource('Normal'):
print "TEST WITH/AS"
```

输出：

```
Resource [Normal]
[Enter Normal]: Allocate resource.
TEST WITH/AS
[Exit Normal]: Free resource.
[Exit Normal]: Exited without exception.
```

__ init __.py存在含义

init.py文件定义了包的属性和方法，假如子目录中也有 **init**.py 那么表示这个包的包含子包。

init.py 文件可以只是一个空文件，但是必须存在。如果 **init**.py 不存在，这个目录就仅仅是一个目录，而不是一个包，它就不能被导入或者包含其它的模块和嵌套包。

当你将一个包作为模块导入时候，实际上导入了它的 **init**.py 文件（可以认为这是模块的构造函数？）。

传递参数

1.代码中不区分穿引用还是传值，对于可变对象及传引用，不可变对象及传值。

2.参数分类：

必备参数：必备参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样，否则将出现语法错误。

缺省参数：调用函数时，缺省参数的值如果没有传入，则被认为是默认值。默认值在函数定义时用等号设置。

不定长参数：函数能处理比当初声明时更多的参数。这些参数叫做不定长参数。加了星号（*）的变量名会存放所有未命名的变量参数。

```
#带星号参数为不定长参数
def printinfo(arg1, *vartuple):
    print arg1
    for var in vartuple:
        print var
printinfo(10)
printinfo(70,60,50)
```

3.传参数的方式

顺序传参。

命名传参：可以乱序，或者跳过某个不传的参数

偏函数functools.partial

函数试编程，目标试简化函数需要传入的参数

假设存在下面函数

```
from datetime import datetime,timedelta
def GetNextDay(baseday,n):
    return str((datetime.strptime(str(baseday),'%Y-%m-%d')+timedelta(days=
```

如果需要进行下面的调用

```
GetNextDay(1,1)
```

```
GetNextDay(1,2)
GetNextDay(1,6)
GetNextDay(1,13)
GetNextDay(1,29)
```

可以简化为

```
import functools
nday = functools.partial(GetNextDay,1)
nday(1)
nday(2)
nday(6)
nday(13)
nday(29)
```

lambda语法

lambda作为一个表达式，定义了一个匿名函数

```
g = lambda x:x+1
等价于:
def g(x):
    return x+1
```

zip

zip是生成并行遍历的方法：

```
L1=[1,3,4]
L2=[3,5.6,5]
for (x,y) in zip(L1,L2):
    print x+y

```

```
#输出:
4
8
10
```

区别：zip根据最短的序列截断，map为短的补none

map

很简单，第一个参数接收一个函数名，第二个参数接收一个可迭代对象

```
ls = [1,2,3]
rs = map(str, ls)
 #打印结果['1', '2', '3']
lt = [1, 2, 3, 4, 5, 6]
def add(num):
    return num + 1
rs = map(add, lt)
print rs #[2,3,4,5,6,7]
```

Class的静态方法、类方法、普通方法

```
@staticmethod
def  staticMethod():
       print("static method")
class MyClass:
    def  method(self):
           print("method")
     @classmethod
     def classMethod(cls):
           print("class method")
```