目录

[第一章：python基础 1](#_Toc350854177)

[1、Python具有很强的面对对象的特性 1](#_Toc350854178)

[2、Python中的注释： 1](#_Toc350854179)

[3、语句的分隔： 1](#_Toc350854180)

[4、局部变量和全局变量 1](#_Toc350854181)

[第三章：python的控制语句 3](#_Toc350854182)

[第五章：模块和函数 4](#_Toc350854183)

[1、包： 4](#_Toc350854184)

[2、模块： 4](#_Toc350854185)

[3、模块的属性： 4](#_Toc350854186)

[4、模块的内置函数： 4](#_Toc350854187)

[5、包： 4](#_Toc350854188)

[6、函数： 5](#_Toc350854189)

[第九章：异常处理和程序调试 7](#_Toc350854190)

[1、Python中的异常： 7](#_Toc350854191)

[2、Try…..except 7](#_Toc350854192)

[3、Python和java处理异常的方式很像，可以嵌套。 8](#_Toc350854193)

[4、使用raise抛出异常 8](#_Toc350854194)

[5、自定义异常： 8](#_Toc350854195)

[6、Assert语句的使用： 9](#_Toc350854196)

[7、异常信息： 9](#_Toc350854197)

## 第一章：python基础

### 1、Python具有很强的面对对象的特性

并且简化了面对对象的实现。它消除了保护类型、抽象类、接口等面对对象的元素，使得面对对象的概念更容易理解。

### 2、Python中的注释：

中文注释：如果需要在代码中使用中文注释，必须在python文件的最前面加上如下注释说明：

#-\*-coding=utf-8-\*-

跨平台注释：如果需要使python程序运行在windows以外的平台，需要在python文件的最前面加上如下的注释说明：

#!/usr/bin/python

### 3、语句的分隔：

;多行放一行 不推荐使用

\一行放多行

### 4、局部变量和全局变量

1）变量

仅仅通过创建一个局部变量来“隐藏“或者覆盖一个全局变量是有可能的。回想一下，局部名字空间是首先被搜索的，存在于其局部作用域。如果找到一个名字，搜索就不会继续去寻找一个全局域的变量，所以在全局或者内建的名字空间内，可以覆盖任何匹配的名字。

使用global语句

如果你想要为一个定义在函数外的变量赋值，那么你就得告诉Python这个变量名不是局部的，而是全局的。我们使用global语句完成这一功能。没有global语句，是不可能为定义在函数外的变量赋值的。

你可以使用定义在函数外的变量的值（假设在函数内没有同名的变量）。然而，我并不鼓励你这样做，并且你应该尽量避免这样做，因为这使得程序的读者会不清楚这个变量是在哪里定义的。使用global语句可以清楚地表明Python函数变量是在外面的块定义的。

例子：

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

#在文件的开头定义全局变量

\_a = 1

\_b = 2

def **add**():

#这里使用了global，用于引用全局变量，如果不这样的话，直接\_a=3,

#相当于又定义了一个局部变量，最终得出来的结果是不对的。

global \_a

\_a = 3

return *'\_a + \_b = '*,\_a+\_b

def **sub**():

global \_b

\_b = 4

return *'\_a - \_b='*,\_a - \_b

print add()

print sub()

#输出的结果为：

#('\_a + \_b = ', 5)

#('\_a - \_b=', -1)

注意：

变量名相同的两个变量可能并不是同一个变量。变量的名称只是起标识的作用，出现的位置不同，变量的含义也不同。

可以把变量放到一个专门的文件中，这样也便于统一管理和修改。如创建一个名为gl.py的文件：

#全局变量

\_a = 1

\_b = 2

再创建一个调用全局变量的文件use\_global.py

#调用全局变量

Import gl

Def fun():

Print gl.\_a

Print gl.\_b

Fun()

不过我们应该避免使用全局变量。

2）常量

Python并没有提供常量的保留字，你可以自定义一个常量类来实现常量的功能。

演示python cookbook一书中定义的常量模块。

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

#ConstError异常继承自typeerror。通过调用类自带的字典\_\_dict\_\_，

#判断定义的常量是否包含在字典中。如果字典中包含此

#常量，将抛出异常，否则，给新创建的常量赋值。

class **\_const**:

class **ConstError**(TypeError):

pass

def **\_\_setattr\_\_**(*self*,name,value):

if *self*.\_\_dict\_\_.has\_key(name):

raise *self*.ConstError,*"can't rebind const(%s)"*%name

*self*.\_\_dict\_\_[name] = value

#将新创建的const类注册到sys.modules这个全局字典中。

import sys

sys.modules[\_\_name\_\_] = \_const()

## 第三章：python的控制语句

1、条件语句和循环语句是不一样的。

2、If\switch等是条件语句，而while、for等则是循环语句。

还有break、continue

3、看一下while else

最后一次循环结束之后将执行else语句。

4、For…..else

最后一次循环结束之后将执行else语句。

For x in 集合（列表、元组、字典）或者range（）、xrange（）、还有一个函数

第四章：内置数据结构

1、内置数据结构是python语言开发的精华

2、全局字典----sys.modules模块

sys.modules是一个全局字典。该字典是python启动后就加载在内存中的。

起到缓存的作用。

## 第五章：模块和函数

Python的程序由包、模块和函数组成。包是又一系列的模块组成的集合。模块是处理某一类问题的函数和类的集合。

### 1、包：

包就是一个完成特定任务的工具箱，python提供了很多有用的工具包，如字符串处理等等。

这些自带的工具包和模块在python的lib文件夹下。

包必须至少含有一个\_\_init\_\_.py的文件，该文件的内容可以为空。\_\_init\_\_.py用于标识当前文件夹的一个包。

### 2、模块：

一个python文件就是一个模块，一个模块由代码、函数、类等组成。

### 3、模块的属性：

模块有一些内置的属性，用于完成特定的任务，如\_\_name\_\_、\_\_doc\_\_。每个模块都有一个名称。例如\_\_name\_\_用于判断当前模块是否是程序的入口。如果当前程序正在被使用，\_\_name\_\_的值为\_\_main\_\_。当他被调用时，值不是\_\_main\_\_.

\_\_name\_\_是指示当前py文件调用方式的方法。如果它等于"\_\_main\_\_"就表示是直接执行，如果不是，则用来被别的文件调用，这个时候if就为False，那么它就不会执行最外层的代码了。

### 4、模块的内置函数：

Python提供了一个内联模块buildin，定义了一些软件开发中常用到的函数。

Apply()

类似于给函数赋值。

Filter（）

过滤作用

Reduce（）

Map()的功能很强大，可以实现对多个序列的每个元素都执行同样的操作。

### 5、包：

包必须至少含有一个\_\_init\_\_.py的文件，该文件的内容可以为空。\_\_init\_\_.py用于标识当前文件夹的一个包。

Python中的包和java中的包的作用是一样的。都是为了实现程序的重用。

包和模块的树形关系：

\_\_init\_\_().py

Mymodule模块

Pack包

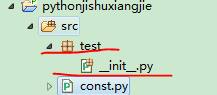
Parent包

\_\_init\_\_().py

Mymode模块

\_\_init\_\_().py

Pack2包



### 6、函数：

1）函数的返回值

Return后面可以是变量或表达式

一个函数中可以使用多个return，但是不推荐。因为return过多往往会造成程序的复杂。这是就需要对代码进行重构了。如果函数有多个return，可以通过增加一个变量来减少。

2）Lambda函数

用于创建一个匿名函数，函数名未和标识符进行绑定。使用lambda函数返回一些简单的运算结果。他的表达式如下：

Lambda 变量1，变量2，变量3…..：表达式

其中，变量列表用于表达式的计算。Lambda中只能使用表达式，不能使用循环等。

把lambda赋值给一个变量，变量就可作为函数使用。如：

#赋值

Func = lambda 。。。。。

#调用

Func（）

例如：

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

sum = lambda x,y:x+y

sub = lambda m,n:m-n

print sum(1,2)

print sub(3,2)

#结果：

#3

#1

3）Generator函数

生成器（generator），的作用是一次产生一个数据项，并把数据项输出。Generator函数可以用在for循环中，generator函数所具有的每次返回一个数据项的功能，使得迭代器的性能更佳。Generator函数的定义如下：

Def 函数名：

。。。。。。

Yield 表达式

Generator函数的定义和普通函数没什么区别，只是在函数体内使用yield生成数据项即可。Generator函数可以被for循环遍历，而且可以通过next()方法获得yield生成的数据项。

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

def **func**(n):

for i in range(n):

yield i

print *'继续执行'*#会打出来，说明是继续执行的

r = func(3)

#print r#<generator object at 0x015E9D78>

print r.next()

print r.next()

print r.next()

print r.next()

print r.next()

#print r.next()#越界了，返回StopIteration这个错误

#返回的结果：

#0

#1

#2

#继续执行

#Traceback (most recent call last):

# File "E:\001workspace\pydev\pythonjishuxiangjie\src\generator\_use.py", line 17, in <module>

# print r.next()

#StopIteration

Yiled保留字和return语句的区别：返回值和执行机理都不相同。Yiled生成值并不会终止程序的执行，返回值后程序继续往后执行，return返回值之后，程序将终止执行。

另外，generator函数会返回元素的值，序列也会，但是两者还是存在很大的区别的。Generator一次只返回一个数据项，占用更少的内存，每次生成数据都要记录当前的状态。序列一次返回所有，还有他们越界产生的异常也不同。

## 第九章：异常处理和程序调试

### 1、Python中的异常：

Python中的异常类定义在exception模块中，并继承自基类BaseException。BaseException下有三个子类，分别是，exception、keyboardinterrupt、systemexit。

注意：

如果在except字句中将异常类型设置为exception，异常处理程序将捕获程序中断外的所有异常。

### 2、Try…..except

可以添加一个else字句，当出现异常时跳到except后，如果不出现异常，则程序执行else。但是else还是和finally不一样的（else只有在不触发异常时才执行，finally什么时候都执行）

### 3、Python和java处理异常的方式很像，可以嵌套。

### 4、使用raise抛出异常

当程序出现错误时，python会自动的引发异常，也可以通过raise语句引发异常，一旦执行了raise语句，raise后面的代码将不被执行，而是直接去执行except后面的语句。

### 5、自定义异常：

Python允许程序员自定义异常类型用于描述python异常体系中没有涉及的异常类型，自定义异常不许继承Exception类，自定义异常按照规范以Error结尾，自定义异常使用raise引发，而且只能通过手工方式触发。

例如：

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

class **DivisionException**(Exception):#定义了自定义异常，该异常继承自Exception

def **\_\_init\_\_**(*self*,x,y):

Exception.\_\_init\_\_(*self*,x,y)#在构造函数中调用基类的构造函数初始化

*self*.x = x

*self*.y = y

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

try:

x = 3

y = 2

if x%y > 0:#判断如果不能整除，则引发异常，手工抛出自定义的DivisionException异常

print x/y

raise DivisionException(x,y)

except DivisionException,div:#div表示DivisionException类的实例对象

print *'DivisionException:x/y = %.2f'*%(div.x/div.y)#调用div的属性

### 6、Assert语句的使用：

Assert语句用于检测某个条件表达式是否为真。Assert语句又称为断言语句，即assert认为检测的表达式永远为真。如果assert语句断言失败，会引发assertionerror异常

注意：

t = (‘helloworld’ , )#python把它作为序列来处理

t = (‘hhfefewfdss’)#python把它作为字符串来处理

### 7、异常信息：

当程序出现错误的时候，python都会输出相关的异常信息，并指向错误的行号和错误的程序代码。

程序执行时，python将产生traceback对象，用于记录异常信息和当期程序的状态。异常信息应该从上往下读。最后一行出现的行号就是异常发生处。

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-12*

*@author: hongtao*

*'''*

import sys

try:

x = 10/0

except Exception,ex:

print ex#输出异常的实例

print sys.exc\_info()#输出一样的类型，traceback对象等信息