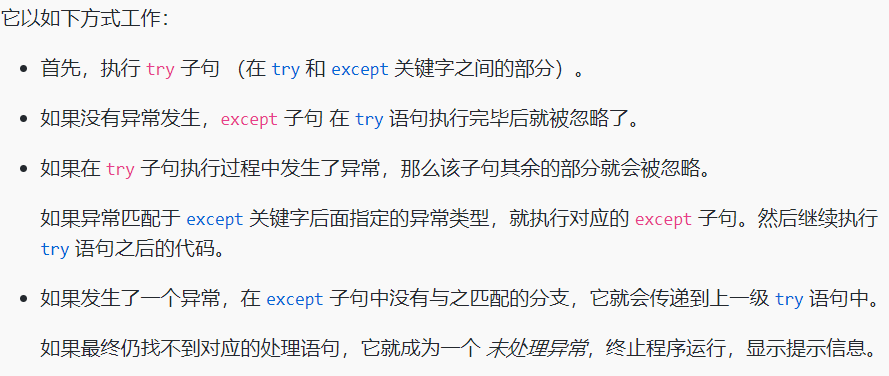
1. if \_\_name\_\_ == "\_\_main\_\_":

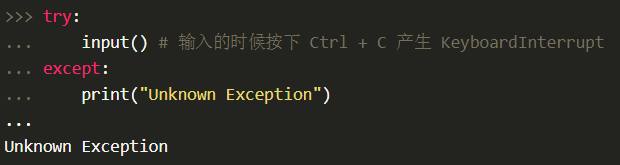
\_\_name\_\_是指示当前py文件调用方式的方法。如果它等于"\_\_main\_\_"就表示是直接执行，如果不是，则用来被别的文件调用，这个时候if就为False，那么它就不会执行最外层的代码了。  
比如你有个Python文件里面  
def XXXX():  
#body  
print "asdf"  
这样的话，就算是别的地方导入这个文件，要调用这个XXXX函数，也会执行print "asdf"，因为他是最外层代码，或者叫做全局代码。但是往往我们希望只有我在执行这个文件的时候才运行一些代码，不是的话（也就是被调用的话）那就不执行这些代码，所以一般改为  
def XXXX():  
#body  
if \_\_name\_\_="\_\_main\_\_":  
print "asdf"

1. 异常处理：

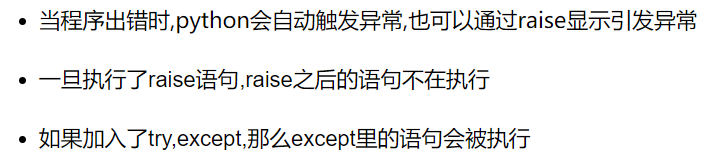
1.Try…except:

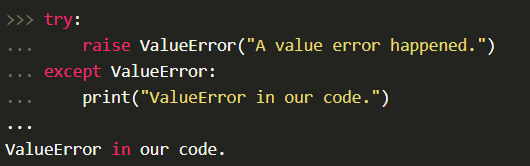


附：空except语句能捕获任何异常



2.抛出异常：





3.定义清理行为：



1. Sys.argv[]:

sys.argv[]说白了就是一个从程序外部获取参数的桥梁，这个“外部”很关键，所以那些试图从代码来说明它作用的解释一直没看明白。因为我们从外部取得的参数可以是多个，所以获得的是一个列表（list)，也就是说sys.argv其实可以看作是一个列表，所以才能用[]提取其中的元素。其第一个元素是程序本身，随后才依次是外部给予的参数。

下面我们通过一个极简单的test.py程序的运行结果来说明它的用法。

1 #test.py

2

3 import sys

4 a=sys.argv[0]

5 print(a)

　将test.py保存在c盘的根目录下。

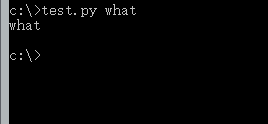
　在程序中找到 ‘运行’->点击->输入"cmd"->回车键   进入控制台命令窗口（如下图），先输入cd c:\   (作用是将命令路径改到c盘根目录)，然后输入test.py运行我们刚刚写的程序：

  得到的结果是C:\test.py，这就是0指代码（即此.py程序）本身的意思。

  然后我们将代码中0改为1 :

  a=sys.argv[1]

  保存后，再从控制台窗口运行，这次我们加上一个参数，输入：test.py what



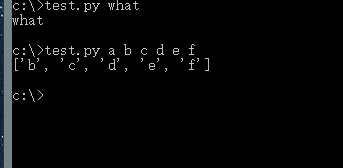
 得到的结果就是我们输入的参数what，看到这里你是不是开始明白了呢。

 那我们再把代码修改一下：

 a=sys.argv[2：]

 保存后，再从控制台窗台运行程序，这次多加几个参数，以空格隔开:

 test.py a b c d e f



 得到的结果为[‘b’, ’c’, ’d’, ’e’, ’f’]

 　　应该大彻大悟了吧。Sys.argv[ ]其实就是一个列表，里边的项为用户输入的参数，关键就是要明白这参数是从程序外部输入的，而非代码本身的什么地方，要想看到它的效果就应该 将程序保存了，从外部来运行程序并给出参数。

1. Del删除对象：