Python中的异常处理与C#中的类似。

异常语句：

try:

main-action

except Exception1:

hander1

except Exception2:

hander2

…

else:

else-block

finally:

finally-block

例：程序exception\_detail/exception\_detail1.py

x = 'spam'  
  
  
def fetcher(obj, index):  
 return obj[index]  
  
  
def test():  
 try:  
 fetcher(x, 4)  
 except IndexError:  
 print('got exception')  
 print('test end') # 发生异常也会执行到这里  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test()

输出为：

got exception

test end

try-except-finally

例：程序exception\_detail/exception\_detail2.py

from exception\_detail1 import fetcher  
  
x = 'spam'  
  
  
def after():  
 try:  
 fetcher(x, 4)  
 except IndexError:  
 print('got exception')  
 return # 即使except有return语句，也会执行finally块，与C#类似  
 finally:  
 print('after fetcher')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 after()

输出为：

got exception

after fetcher

with/as环境管理协议

with expression [as variable]:

with-block

with语句：

1. 计算表达式，得到的对象称为环境管理器，必须有\_\_enter\_\_和\_\_exit\_\_。
2. 环境管理器的\_\_enter\_\_方法会被调用。如果as子句存在，返回值赋给as子句的变量，否则直接丢弃
3. 代码块中嵌套的代码会执行
4. 如果with代码块发生异常，\_\_exit\_\_(type, value, traceback)就会被调用。如果此方法返回值为False，异常会重新引发，传递到with语句之外，否则，异常会中止。
5. 如果with代码块没有引发异常，\_\_exit\_\_仍然会被调用，其中type、value和traceback参数都会以None传递。

例：程序exception\_detail/exception\_detail3.py

class TraceBlock:  
 def message(self, arg):  
 print('running', arg)  
  
 def \_\_enter\_\_(self):  
 print('starting with block')  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 if exc\_type is None:  
 print('exited normally\n')  
 else:  
 print('raise an exception!', exc\_type)  
 return False  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with TraceBlock() as action:  
 action.message('test1')  
 print('reached')  
  
 with TraceBlock() as action2:  
 action2.message('test 2')  
 raise TypeError  
 print('not reached')

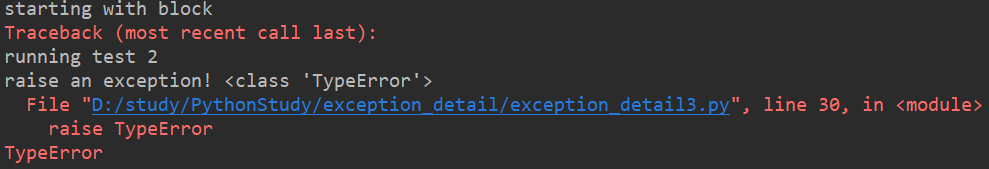
输出为：

starting with block

running test1

reached

exited normally



异常对象：基于类的异常

例：程序exception\_detail/exception\_detail4.py

class General(Exception):  
 pass  
  
  
class Specific1(General):  
 pass  
  
  
class Specific2(General):  
 pass  
  
  
def raiser0():  
 x = General()  
 raise x  
  
  
def raiser1():  
 x = Specific1()  
 raise x  
  
  
def raiser2():  
 x = Specific2()  
 raise x  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for func in (raiser0, raiser1, raiser2):  
 try:  
 func()  
 except General: # 捕获超类异常，可以捕获继承其的所有子类异常  
 import sys  
 print('caught:', sys.exc\_info()[0])

输出为：

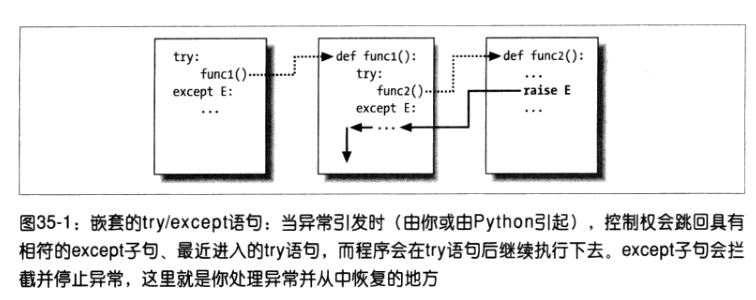
caught: <class '\_\_main\_\_.General'>

caught: <class '\_\_main\_\_.Specific1'>

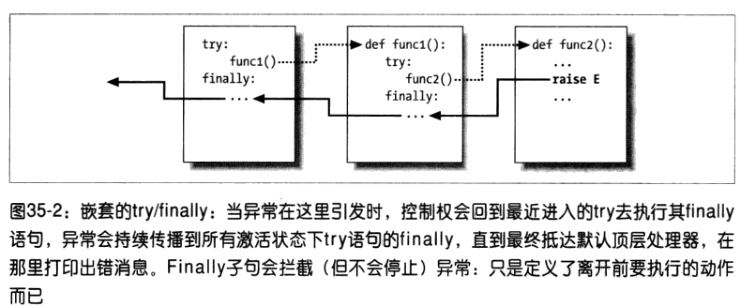
caught: <class '\_\_main\_\_.Specific2'>

异常的嵌套：

try/except语句的嵌套，异常一旦被捕获，其生命就会结束。多个相符（不同种类的异常捕获互不影响）try/except语句，只有第1个会对其进行处理。



try/finally语句嵌套时，当异常发生，每个try/finally的finally块都会执行，Python会把异常逐层上传到其他的try/finally。



例：程序exception\_detail/exception\_detail5.py

def action():  
 print(1 + [])  
  
  
try:  
 try:  
 action()  
 except TypeError:  
 print('inner try')  
except TypeError:  
 print('outer try')  
  
try:  
 try:  
 raise IndexError  
 finally:  
 print('spam')  
finally:  
 print('SPAM')