**day09**

day09(上午)：

异常处理涉及的关键字：

**try：**

理解它是扫描器，将可能出现异常的代码放入其中；

如果在执行的过程中出现异常对象了，扫描器会立即察觉到此异常对象，

但是它没有处理它的能力，所以会将异常对象给到except(捕获器)进行处理

**except：**

理解它是捕获器，后面可以定义异常类型，并且和as关键字配合使用；

定义多个except是合法的，但是它的执行顺序是有由上往下，一旦匹配上某个except，之后的就不执行了；

匹配成功就理解将异常对象捕获住并且kill，顺便会执行except内部的代码

**finally：**

将一定需要被执行的代码定义其中，【记住】：finally一定会被执行(核心)

使用场景：像关闭文件、断开数据库连接等行为都可以放入到finally中

**else：**

位置在finally前，except后；效果/作用类似于循环

没有异常对象出现，else就会被执行，反之不会被执行

**raise：**

手动抛出一个异常类型的对象，定义：raise 异常类型(msg)

定义格式：

格式一：

try:

语句块1

except 异常类型1 as e:

语句块2

except 异常类型2 as e:

语句块3

except 异常类型3 as e:

语句块4

finally:

语句块5

【注意事项】：

以上的格式由多个except语句并列存在，每个except包含一种类型

如果它们之间没有子父类的关系(互斥)，那么谁上谁下无所谓

如果它们之间存在子父类的关系(包含)，那么小的在上，大的在下

格式二：

try:

语句块1

except 异常类型1 as e:

语句块2

except 异常类型2 as e:

语句块3

except 异常类型3 as e:

语句块4

except (异常类型4,异常类型5,...,异常类型m) as e:

finally:

语句块5

【格式说明】：

一个except具备多个捕获能力(捕获多种不同类型异常对象)

格式三：

try:

语句块1

except 异常类型1 as e:

语句块2

except 异常类型2 as e:

语句块3

except 异常类型3 as e:

语句块4

except (异常类型4,异常类型5,...,异常类型m) as e:

语句块5

except Exception as e:

语句块6

finally:

语句块7

【类型说明】：

Exception是所有异常类型的父类型，它都是定义在最后一个except的位置

格式三：

try:

语句块1

except 异常类型1 as e:

语句块2

except 异常类型2 as e:

语句块3

except 异常类型3 as e:

语句块4

except (异常类型4,异常类型5,...,异常类型m) as e:

语句块5

except:

语句块6

finally:

语句块7

【Exception的简化版使用】：

如果定义了except，那么它必须是最后一个except，它可以理解为是Exception的简化形式

注意事项：

1).try...finally这种格式是合法的，但是它丧失了解决异常对象的能力 --> 开发不会出现

2).else语句必须配合except使用，出现位置一定是在最后一个except的后面

3).常见的运行时异常类型：

ZeroDivisionError：分母为0的异常

TypeError：类型有误的异常

NameError：没有定义就使用的异常

IndexError：下标越界的异常

FileNotFoundError：文件找不到的异常

4).with open ... as ...语句可以优化原始的open操作！体现：它不需要手动close()文件对象

总结：学习异常对象简单归纳为5个关键字：try except finally else raise

1.异常体系：

**演示自动抛出异常对象***：***def** div(a,b):  
 print(a / b)  
div(10,0)

*演示异常处理的方式一：try ... except ...*

**try**:  
 print(10/0)  
**except** ZeroDivisionError **as** e:  
 print(e)  
**try**:  
 print(**"abc"**+123)  
**except** TypeError **as** e:  
 print(e)  
**try**:  
 print(a)  
**except** NameError **as** e:  
 print(e)  
**try**:  
 lt = [1,2,3,4,5]  
 print(lt[5])  
**except** IndexError **as** e:  
 print(e)  
**try**:  
 fr = open(**r'C:\Users\Administrator\Desktop\kaifanglist1.txt'**,**'r'**,encoding=**'utf-8'**)  
 print(fr.read())  
 fr.close()  
**except** FileNotFoundError **as** e**:**print(e)  
**except**(ZeroDivisionError,TypeError,NameError,IndexError,FileNotFoundError) **as** e:  
 print(**'我能捕获5种类型的异常对象...'**)

**except** Exception **as** e:  
 print(**'我是所有异常类型的父类型，我能解决所有的异常类型对象...'**)

**except**:  
 print(**'我是Exception的简化形式，我只能出现在最后一个except的位置...'**)

*补充：except不能捕获处理类似于语法错误这种情况*

**演示finally语句的使用：  
将一定要被执行的代码定义在finally中，不管之前的代码怎么样(异常是否被解决)，finally一定会被执行，在后期的学习和开发中，我们将关闭文件、关闭数据库连接等操作都定finally中**

**try**:  
 fr = open(**r'C:\Users\Administrator\Desktop\a.txt'**,**'r'**) *#文件能打开* print(fr.read())  
 print(10 / 0)  
 *#fr.close() #因为关闭文件操作是一定要被执行的 但是在try里面也有可能存在风险性，当*print(10 / 0)*行不了，则无法执行close操作，所以关闭文件操作需要放在finally语句当中***except**:  
 print(**'解决异常'**)  
**finally**:  
 print(**'我是finally，我一定会被执行...'**)  
 fr.close()

**但是如果一开始就是一个打不开的文件，则也没有办法关闭文件啦**

fr = 0  
**try**:  
 fr = open(**r'C:\Users\Administrator\Desktop\a111.txt'**,**'r'**) *#没有此文件名 （文件存在则fr不为None值）* print(fr.read())  
**except**:  
 print(**'解决异常'**)  
**finally**:  
 print(**'我是finally，我一定会被执行...'**)  
 **if** fr: *#fr为None值则if为False,否则为True,执行以下指令（为了安全性）*  
 fr.close()

**引入新的语法糖：  
with open...操作：不需要程序员认为的去书写close()函数**

**With** open(**r'C:\Users\Administrator\Desktop\a.txt'**,**'r'**)**as** fr:  
 print(fr.read())

解决异常也是一样

**try**:  
 **with** open(**r'C:\Users\Administrator\Desktop\a.txt'**,**'r'**) **as** fr:  
 print(fr.read())  
**except**:  
 print(**'干掉啦，干掉啦...'**)

**面试题：Finally案例：分别输出了哪些内容，返回值是多少**

**def** func():  
 **try**:  
 print(**'我是try...'**)  
 print(10 / 2)  
 **return** 1 *#执行此步 则函数就结束了，又因为finally一定要被执行，所以return 1不会被执行，直接调到finally* **except**:  
 print(**'我是except...'**)  
 **return** 2  
 **finally**:  
 print(**'我是finally...'**)  
 **return** 3  
num = func()  
print(num)

*#结果得到是 我是try...*

*5.0*

*我是finally...*

*3*

**如果题目改为如下，则**

**def** func():  
 **try**:  
 print(**'我是try...'**)  
 print(10 / 0)  
 **return** 1  
 **except**:  
 print(**'我是except...'**)  
 **return** 2 *#执行此步 则函数就结束了，又因为finally一定要被执行，所以return 2不会被执行，直接调到finally*  
 **finally**:  
 print(**'我是finally...'**)  
 **return** 3  
num = func()  
print(num)

*#结果得到是 我是try...*

*我是except...*

*我是finally...*

*3*

**演示else语句和异常处理机制配合使用：  
特点：  
如果try中没有出现异常对象，那么else语句就一定会被执行  
如果try中出现了异常对象，就算被处理了，else还是不会被执行  
try**:  
 print(**'我是try...'**)  
 print(10 / 0)

**except** Exception **as** e:  
 print(**'我是except...'**)  
**else**:  
 print(**'我是else...'**)  
**finally**:  
 print(**'我是finally...'**)

*#得到 我是try...*

*我是except...*

*我是finally...*

#如果吧print(10 / 0) 改为print(10 / 2)

*#得到 我是try...*

*5.0*

*我是else...*

*我是finally...*

***演示raise的使用：  
作用：手动抛出异常对象***age = 18  
**if** age < 0 **or** age > 130:  
 **try**:  
 **raise** Exception(**'年龄有问题...'**)  
 **except**:  
 print(**'年龄不合法...正在处理'**)  
 **pass  
else**:  
 print(**'年龄为：%d'** %age)

print(**'能走吗？'**)

*#得到结果为 年龄为：18*

*能走吗？*

#如果第一行是age=-18，则  
*#得到结果为 年龄不合法...正在处理*

*能走吗？*

**演示异常处理的方式二：不断往上级传递异常对象，回避问题的方式****def** m1():  
 print(**'我是m1...'**)  
 print(10 / 0)  
**def** m2():  
 print(**'我是m2...'**)  
 *# try:* m1()  
 *# except:  
 # pass***def** m3():  
 print(**'我是m3...'**)  
 **try**:  
 m2()  
 **except**:  
 **pass**m3()

*#得到结果 我是m3...*

*我是m2...*

*我是m1...*

**需求：从键盘读入n个整数，判断整数、负数、0的个数**zs = 0  
fs = 0  
ling = 0  
**while** 1:  
 **try**:  
 num = int(input(**'请输入一个整数：'**))  
 **except**:  
 **break  
 if** num > 0:  
 zs += 1  
 **elif** num < 0:  
 fs += 1  
 **else**:  
 ling += 1  
print(**'正数的个数为：%d'** %zs)  
print(**'负数的个数为：%d'** %fs)  
print(**'零的个数为：%d'** %ling)

2.递归：

递归调用：不断的对于某操作重复调用执行

递归函数：在某个函数中，调用其本身(函数自己调用自己)，这个函数整体我们称为递归函数

递归函数执行，这个过程中只有进栈(开辟空间)，

没有出栈，直到最后一次调用完毕了，才逐个出栈；

所以递归函数在执行的时候非常的占用内存资源；

【注意】：如果执行的次数过多了，会产生内存溢出的现象；

【总结】：递归在之后的学习和开发环境下要慎用！！

如何定义递归函数？

考虑切入点：

1).寻找一个临界点(结束点)

2).本次的函数执行不能满足最终的整体需求，所以还需要再次的去调用此函数；

(找到后者和前者之间的关系，体现：后者的调用结束会将前者需要的数据返回回来)

递归的弊端：

参看老郭代码... *演示递归的弊端：***def** mySum(num):  
 **if** num == 1:  
 **return** 1  
 **return** num + mySum(num - 1)  
mySum(998) *#998是零界点，再大的话就会溢出*

**演示简单递归的定义以及使用  
需求：实现1~5的累加  
发现关系：  
函数名：mySum(num)  
1).找临界点：运算到1(加到1)就结束了  
2).  
代码：mySum(5)  
第一次：5 + mySum(5 - 1) --> return 5 + 10 15  
第二次：4 + mySum(4 - 1) --> return 4 + 6 10  
第三次：3 + mySum(3 - 1) --> return 3 + 3 6  
第四次：2 + mySum(2 - 1) --> return 2 + 1 3  
第五次：1 --> return 1  
def** mySum(num):  
 **if** num == 1:  
 **return** 1  
 **return** num + mySum(num - 1)  
print(mySum(5))  
*#需求：计算1~5的阶乘***def** jiechen(num):  
 **if** num == 1:  
 **return** 1  
 **return** num \* jiechen(num - 1)  
print(jiechen(5))

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
需求：遍历某个路径下面的所有内容(文件和目录，考虑多级的情况)  
import** os*#自定义函数(递归函数)：遍历目录层级(多层)***def** printDirs(path):  
 dirs = os.listdir(path)  
 *#循环处理此列表* **for** d **in** dirs:  
 *#组装d得到其绝对路径形式* fileAbsPath = os.path.join(path,d)  
 *#判断是目录还是文件  
 #如果是文件直接打印即可，如果是目录打印完毕后再次调用此函数* **if** os.path.isfile(fileAbsPath):  
 print(d)  
 **elif** os.path.isdir(fileAbsPath):  
 print(d)  
 printDirs(fileAbsPath)  
path = **r'C:\Users\Administrator\Desktop\a'**printDirs(path)

项目

围绕面向对象思想进行学习：

1).面向过程和面向对象区别

2).类和对象的概念

3).self、构造函数、析构函数...、魔术函数

4).面向对象的特性：封装、继承、多态

5).对象序列化、反序列化机制 --> pickle模块

6).整合案例(项目阶段)：

银行案例：

功能：开户、查询、取款、存款、转账、锁定、解锁、改密...

员工案例：

新增员工、

查询员工信息：

全部查询

按id进行查询

按姓名进行查询

删除员工、

修改员工薪资