

## 7.错误与异常

### 错误和异常的概念：

- 错误 (Error)

```
print(a)                                #NameError: name 'a' is not defined
```

指的是：语法错误，或者解析错误。

- 异常 (Exception)

```
a = input("请输入一个整数")           #输入a
print(int(a))                          #ValueError: invalid literal for int() with base
5: ' '
```

指的是：运行期检测到的错误被称为异常。

### 为什么要进行异常处理？

为了让我们的程序在运行过程中，遇到一些“问题”（可以预知，但不能无视）的时候，仍然能够根据问题进行处理，让程序能够继续的运行下去。

场景：

面对用户输入被除数为0，是程序崩溃、终止？还是给出用户提示，让程序继续运行。

用户输入错误，就是“异常”；我们对这样的情况进行处理，就是“异常处理”。

案例：

```
a = input("请输入一个整数: ")
print(int(a))

n = input("请输入被除数: ")
m = input("请输入除数: ")
res = float(n)/float(m)
print("res:",res)
```

上述代码在用户输入不同内容时，会产生不同的错误提示：

- 情况1：用户输入的不是数字，无法计算，例如：a,b。 错误类型：ValueError: could not convert string to float
- 情况2：用户输入的除数为0。 错误类型：ZeroDivisionError: float division by zero

## 2个处理步骤：

准备：预知错误的情况下，充分的条件判断 【业务逻辑代码】

等待：准备错误预案，出错后进行异常处理 【异常处理代码】

可以通过充分的条件判断，让程序继续运行，例如：

```
n = input("请输入被除数：")
m = input("请输入除数：")
if n.isdigit() and m.isdigit():      #条件判断
    if m != "0":                      #条件判断
        res = float(n)/float(m)      #业务代码
        print("res:", res)
    else:
        print("除数不能为0")         #异常处理代码
else:
    print("请输入数字")
print("程序继续运行并正常结束")
```

**问题：**业务逻辑和异常处理的代码混在一起，不利于代码的维护和表达

## 7.1 异常简介

看如下示例：

```
print '-----test--1---'
open('123.txt','r')
print '-----test--2---'
```

运行结果：

```
code@ubuntu:~/python-test$ python test.py
-----test--1---
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    open('123.txt','r')
IOError: [Errno 2] No such file or directory: '123.txt'
```

说明：

打开一个不存在的文件123.txt，当找不到123.txt文件时，就会抛出给我们一个IOError类型的错误，No such file or directory: 123.txt（没有123.txt这样的文件或目录）

**异常：**

当Python检测到一个错误时，解释器就无法继续执行了，反而出现了一些错误的提示，这就是所谓的“异常”

## 7.2 捕获异常

### 7.2.1 try...except...

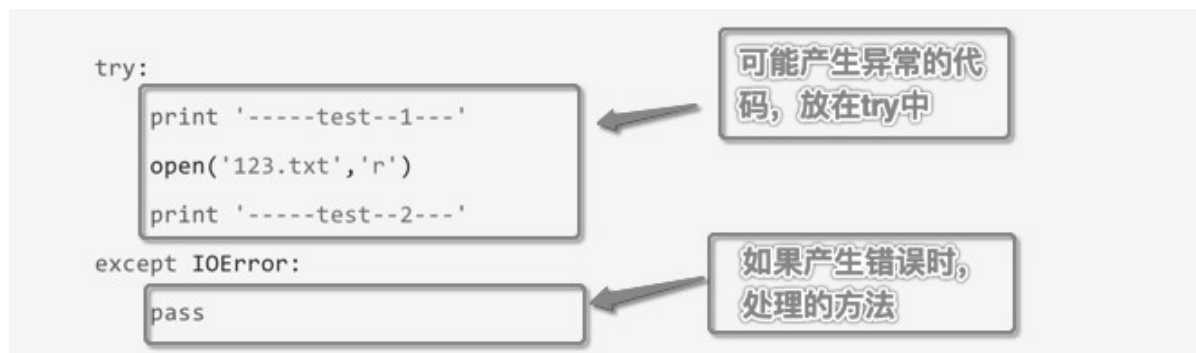
看如下示例:

```
try:
    print('-----test--1---')
    open('123.txt','r')
    print('-----test--2---')
except IOError:
    pass
```

说明:

- 此程序看不到任何错误, 因为用except 捕获到了IOError异常, 并添加了处理的方法
- pass 表示实现了相应的实现, 但什么也不做; 如果把pass改为print语句, 那么就会输出其他信息

小总结:



- 把可能出现问题的代码, 放在try中
- 把处理异常的代码, 放在except中

### 7.2.2 except捕获多个异常

看如下示例:

```
try:
    print num
except IOError:
    print('产生错误了')
```

想一想:

上例程序, 已经使用except来捕获异常了, 为什么还会看到错误的信息提示?

答:

except捕获的错误类型是IOError, 而此时程序产生的异常为 NameError, 所以except没有生效

修改后的代码为:

```
try:
    print num
except NameError:          #异常类型想要被捕获，需要一致
    print('产生错误了')
```

实际开发中，捕获多个异常的方式，如下：

```
#coding=utf-8
try:
    print('-----test--1---')
    open('123.txt','r')      # 如果123.txt文件不存在，那么会产生 IOError 异常
    print('-----test--2---')
    print(num)              # 如果num变量没有定义，那么会产生 NameError 异常

except (IOError,NameError):
    #如果想通过一次except捕获到多个异常可以用一个元组的方式
```

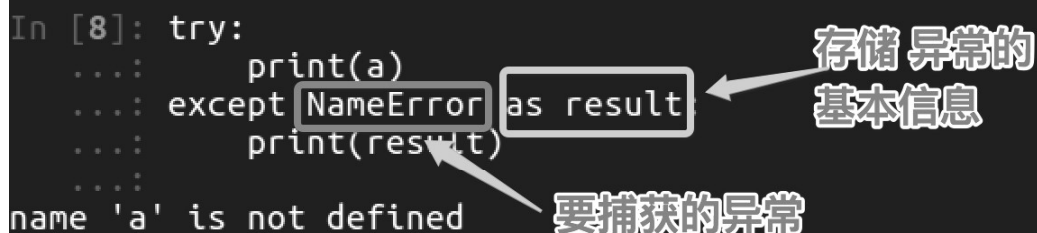
**注意：**

- 当捕获多个异常时，可以把要捕获的异常的名字，放到except 后，并使用元组的方式仅进行存储

如果需要对于不同的异常类型，进行不同的异常处理操作，可以使用多个except：

```
#1.输入不是数字，提示：请输入数字
#2.除数为0，提示：除数不能为0
try:
    n = input("请输入被除数：")
    m = input("请输入除数：")
    res = float(n)/float(m)
    print("res:",res)
except ValueError:
    print("请输入数字")          #数值错误
except ZeroDivisionError:
    print("除数不能为0")        #除零错误
print("程序继续运行并正常结束")
```

### 7.2.3 获取异常的信息描述



```
In [8]: try:
        print(a)
        except NameError as result:
            print(result)

name 'a' is not defined
```

存储异常的基本信息

要捕获的异常

```
In [13]: try:
...:     open("a.txt")
...: except (NameError,IOError) as result:
...:     print("哈哈，捕获到了异常")
...:     print("异常的基本信息是:",result)
```

捕获多个异常，并且存储异常的基本信息

哈哈，捕获到了异常

异常的基本信息是: [Errno 2] No such file or directory: 'a.txt'

```
try:
    n = input("请输入被除数: ")
    m = input("请输入除数: ")
    res = float(n) / float(m)
    print("res:", res)
except Exception as e:
    #print(e,type(e))
    print('str(Exception):\t', str(Exception))
    print('str(e):\t\t', str(e))      #根据错误对象不同，显示其中的错误信息
    print('repr(e):\t', repr(e))      #内置函数repr: 返回一个对象的 string 格式。
```

输出内容为:

```
请输入被除数: 3
请输入除数: 0
str(Exception): <class 'Exception'>
str(e): float division by zero
repr(e): ZeroDivisionError('float division by zero')
```

#### 7.2.4 捕获所有异常

Exception 类型，是异常类型的父类。能够匹配所有异常类型。可以通过as来指向捕获到的异常对象。

```
In [14]: try:
...:     open("a.txt")
...: except:
...:     print("产生了一个异常")
```

没有存储异常的基本信息

产生了一个异常

```
In [15]: try:
...:     open("a.txt")
...: except Exception as result:
...:     print("捕获到了异常")
...:     print(result)
```

捕获所有异常，并且存储异常的基本信息

捕获到了异常

[Errno 2] No such file or directory: 'a.txt'

如果希望根据不同类型的异常进行不同的处理，又希望能够统一处理其他异常的时候，需要将Exception或BaseException放到最后一个。否则优先匹配到最通用的异常类型后，其他（子类）异常类型就不再逐一匹配了。

```

try:
    n = input("请输入被除数: ")
    m = input("请输入除数: ")
    res = float(n)/float(m)
    print("res:",res)

except ValueError:
    print("请输入数字")          #数值错误
except ZeroDivisionError:
    print("除数不能为0")        #除零错误
except BaseException:
    print("出错了")             #注意: 错误类型, 子类在前, 父类在后 (范围小的在前面)

print("程序继续运行并正常结束")

```

### 7.2.5 try... except... else... finally....

try...finally...语句用来表达这样的情况:

在程序中, 如果一个段代码必须要执行, 即无论异常是否产生都要执行, 那么此时就需要使用 finally。比如文件关闭, 释放锁, 把数据库连接返还给连接池等

demo:

```

try:
    n = input("请输入被除数: ")
    m = input("请输入除数: ")
    res = float(n) / float(m)
    #print("res:", res)
except Exception as e:
    print("出错了")          #出错了, 执行
else:
    print("res:",res)        #没出错, 执行
finally:
    print("用户输入完毕, 计算结束")  #不论程序是否正常运行, finally中的代码都会执行

print("程序继续运行并正常结束")

```

### 7.2.6 嵌套的异常处理

异常处理是可以嵌套的。

只要有可能发生异常, 并且不希望异常进行“传导”(被调用的时候才进行处理), 就需要进行异常处理。

及时修正错误, 可以保证后续的代码能够继续运行。

```

try:
    f = open("test11.txt", "r", encoding="utf-8")
    for line in f:
        try:
            3/0
        except ZeroDivisionError:
            print("计算错误")

```

```
        print(line,end="")
except FileNotFoundError:
    print("文件没找到")
finally:
    try:
        f.close()
    except NameError:
        print("文件没有正常打开，不需要关闭")
```

## 7.3 异常类型

异常名称	描述
BaseException	所有异常的基类
SystemExit	解释器请求退出
KeyboardInterrupt	用户中断执行(通常是输入^C)
Exception	常规错误的基类
StopIteration	迭代器没有更多的值
GeneratorExit	生成器(generator)发生异常来通知退出
StandardError	所有的内建标准异常的基类
ArithmeticError	所有数值计算错误的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
<b>ZeroDivisionError</b>	除(或取模)零 (所有数据类型)
AssertionError	断言语句失败
<b>AttributeError</b>	对象没有这个属性
<b>EOFError</b>	没有内建输入,到达EOF 标记
EnvironmentError	操作系统错误的基类
<b>IOError</b>	输入/输出操作失败
OSError	操作系统错误
WindowsError	系统调用失败
ImportError	导入模块/对象失败
LookupError	无效数据查询的基类
IndexError	序列中没有此索引(index)
<b>KeyError</b>	映射中没有这个键
MemoryError	内存溢出错误(对于Python 解释器不是致命的)
NameError	未声明/初始化对象 (没有属性)
UnboundLocalError	访问未初始化的本地变量
ReferenceError	弱引用(Weak reference)试图访问已经垃圾回收了的对象
RuntimeError	一般的运行时错误
NotImplementedError	尚未实现的方法
<b>SyntaxError</b>	Python 语法错误
IndentationError	缩进错误



异常名称	描述
TabError	Tab 和空格混用
SystemError	一般的解释器系统错误
<b>TypeError</b>	对类型无效的操作
<b>ValueError</b>	传入无效的参数
UnicodeError	Unicode 相关的错误
UnicodeDecodeError	Unicode 解码时的错误
UnicodeEncodeError	Unicode 编码时错误
UnicodeTranslateError	Unicode 转换时错误
Warning	警告的基类
DeprecationWarning	关于被弃用的特征的警告
FutureWarning	关于构造将来语义会有改变的警告
OverflowWarning	旧的关于自动提升为长整型(long)的警告
PendingDeprecationWarning	关于特性将会被废弃的警告
RuntimeWarning	可疑的运行时行为(runtime behavior)的警告
SyntaxWarning	可疑的语法的警告
UserWarning	用户代码生成的警告

## 7.4 with语句 (进阶)

with语句，是应用上下文管理器，实现了文件操作的简化：可以省略f.close() 代码,实现自动关闭。

```
#f = open("test.txt","r",encoding="utf-8")
with open("test.txt","r",encoding="utf-8") as f:           #可以使用with语句打开文件，定义变量f
    for line in f:
        print(line,end="")
```

复制图片，使用with语句实现：

```
with open("luoluo.jpg","rb") as fin:
    with open("luoluo_copy3.jpg","wb") as fout:
        for data in fin:
            fout.write(data)

#简单写法： open 之间可以用逗号,隔开
with open("luoluo.jpg","rb") as fin,open("luoluo_copy4.jpg","wb") as fout:
    for data in fin:
        fout.write(data)
```