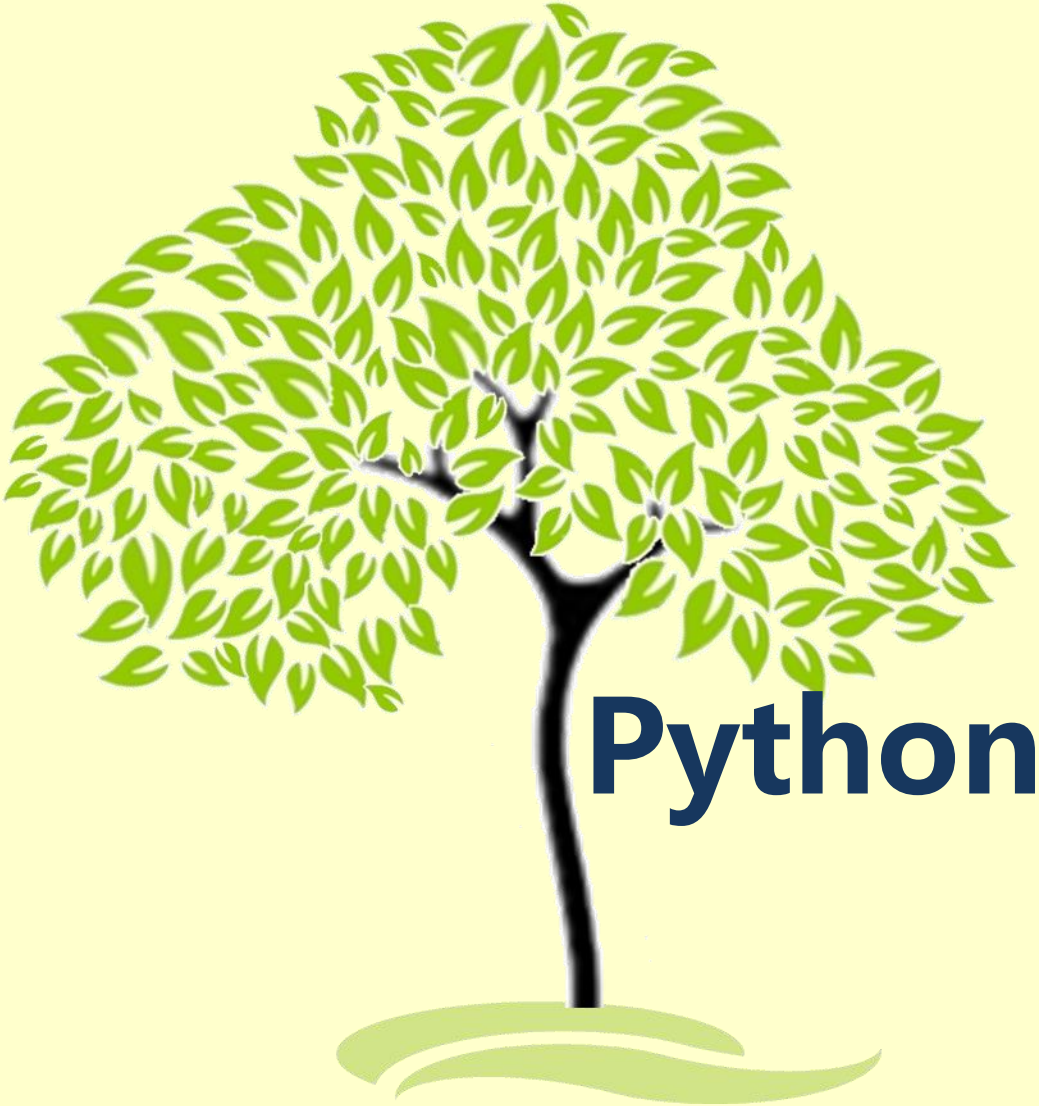

Python 程序设计基础

江西财经大学-朱文强

第七章

异常处理

异常处理

对于程序开发人员来说，程序出错是不可避免的。**当Python程序报错时，程序会强制退出，并且会在控制台打印错误信息和提示信息**。不同错误的解决方案是不一样的，本章将介绍Python中常见的错误、异常以及异常处理机制。

1 错误和异常

所谓的程序错误主要是指语法错误，而异常是指语法正确但是在程序运行过程中出现的一些问题(面向对象不同)。对于语法错误，必须在程序执行前人工进行修改，否则程序无法运行；而对于异常，则可以通过程序进行控制。

1.1 错误

语法错误是指源代码中的拼写不符合解释器和编译器所要求的语法规则，一般集成开发工具中都会直接提示语法错误，编译时提示SyntaxError。语法错误必须在程序执行前改正，否则程序无法运行。**逻辑错误是程序代码可执行，但执行结果不符合要求**。例如求两个数中的最大数，返回的结果却是最小数。

1.1 错误

常见的语法错误有:

- 需要使用英文半角符号的地方用了中文全角符号;
- 变量、函数等命名不符合Python命名规范;
- 条件语句、循环语句、函数定义后面忘了写冒号;
- 位于同一层级的语句缩进不一致;
- 判断两个对象相等时, 使用一个等号而不是两个等号;
- 语句较为复杂时，括号的嵌套层次错误，少了或多了左/右括号;
- 函数定义时, 不同类型参数之间的顺序不符合要求。

1.2 异常

异常是指程序语法正确，但执行中因一些意外而导致的错误，异常并一定总会发生。 例如两个数相除，只有当除数为0时才会发生异常。

每当程序运行发生异常时，它都会创建一个异常类实例。如果你编写了处理该异常的代码，程序将继续运行；**如果你未对异常进行处理，程序将停止，并显示一个Traceback，其中包含有关异常的报告(包括函数调用出错反向跟踪序列，也称为调用栈)**。

2 异常处理机制

2.1 异常处理结构

异常处理是指程序设计时，就考虑到了可能出现的意外情况，为了避免因异常而导致程序终止给用户带来不好的体验，**而做的一些额外处理操作。异常处理使得异常出现后，程序仍然可以执行。**

Python提供了的3种异常处理结构，分别为**try…except，try…except…else，try…except…else…finally**结构。接下来将一一介绍它们。

2.1 异常处理结构

Python**异常处理最基本的结构是try...except...**, 语法如下:

```
try:
    代码块
except 异常名称1:
    处理异常的代码块
……
except 异常名称n:
    处理异常的代码块
……
```

2.1 异常处理结构(示例)

```
print(“两个整数相除的程序”)  # 产生异常的程序
print("按'q'健退出程序")
while True:
    num1 = input("请输入被除数:")
    num2 = input("请输入除数:")
    if num1=='q' or num2=='q':
        break
```

2.1 异常处理结构(示例)

```
    result = int(num1) / int(num2)

    print(f"两数相除结果:{result}")

print("程序运行结束")
```

2.1 异常处理结构(示例)

 # try...except...结构示例(进行异常处理, 接前例)

```
.....
    try:
        result = int(num1) / int(num2)
    except ZeroDivisionError:
        print("除数不能为0, 请重新输入")
    except ValueError as v:
        print("输入必须为数字, 请重新输入")
```

2.1 异常处理结构(示例)

```
    print(f"两数相除结果:{result}")

print("程序运行结束")
```

2.1 异常处理结构

对try...except...异常处理结构的说明。

(1)try子句中可含多条语句，**如果未发生异常，语句依次执行**。如果发生异常，则忽略异常发生处后面的语句，转向except语句执行。

(2)try子句后可接多个except语句，分别用来处理不同类型的异常。**发生异常时，按照except语句顺序从上到下依次匹配**，如果匹配到所捕获的异常类，则执行该except子句中的代码块。异常处理结束后，不再匹配后面的except语句，即**最多只有一个except子句会执行**。

2.1 异常处理结构

(3) except子句的顺序会影响到程序的执行结果, 如果异常之间存在包含关系，通常会**将范围小的异常放在前面，范围大的异常放在后面**。

(4) **一个except子句可同时处理多个异常**，这时可将多个异常名称放在一个元组中。例如except (异常名称1，异常名称2, ...)。

2.1 异常处理结构

(5)如果所有except语句都不匹配，那就相当于未捕获异常，此时将采用默认处理方式，程序终止，打印异常堆栈信息。因此，**通常会让最后一个except不指定异常名称，或指定异常名称为Exception**，Exception是所有异常类的基类。此时，程序可处理所有的异常。

2.1 异常处理结构

try...except...else…结构, 语法如下:

try:

    代码块

except 异常名称1:

……

else:

    try语句成功后的代码块

……

2.1 异常处理结构

 # try...except...else…结构示例, 接前例

```
……
    else:
        print(f"两数相除结果:{result}")
print("程序运行结束")
```

2.1 异常处理结构

(1)与try...except...结构相比，该结构多了一个else子句，表示如果**try子句中没有出现异常且没有return语句，则执行else子句**;

(2)else语句不能独立存在，必须在except语句之后，且**与except语句互斥执行**;

(3)else语句中的内容也可以直接放在try语句块的最后，效果是等价的。

2.1 异常处理结构

try...except...else…finally结构, 语法如下:

```
try:
    代码块
except 异常名称1:
……
finally:
    必须执行的代码块    # print("数据库连接关闭。")
……
```

2.1 异常处理结构

 # try...except...else…finally结构示例, 接前例

```
……
total = 0
while True:
    ……
    finally:
        total += 1
        print(f"程序执行了{total}次")
print("程序运行结束")
```

2.1 异常处理结构

(1)与try...except...else…结构相比，该结构多了一个finally子句，无论是否发生异常都会执行finally子句。**它一般用来做收尾工作，如关闭之前打开的文件或关闭数据库连接**;

(2)try子句不能独立存在，后面必须要有except子句或finally子句;

(3)若try或except中存在break、continue、return语句，finally子句将在这些语句执行前执行;

(4)**如果finally子句包含return语句，则函数返回值为finally子句中return语句的返回值**。

2.1 异常处理结构(示例)

```
def words_count(filename):  # 也可用正则表达式实现
    try:
        with open(filename) as fb:
            content = fb.read()
    except (FileNotFoundError, FileExistsError):
        print(f"{filename}文件读取异常")
    else:
        all_words = content.split()
```

2.1 异常处理结构(示例)

```
        total = len(all_words)
        print(f"{filename}拥有{total}个单词.")


files = ["alice.txt", "little_women.txt",
         "big_shark.txt", "moby_dick.txt"]


for file in files:
    words_count(file)
```

2.2 抛出自定义异常(常见异常类)

AssertError	断言语句(assert)失败
AttributeError	尝试访问未知的对象属性
FloatingPointError	浮点计算错误
FileNotFoundError	文件不存在错误
ImportError	导入模块失败
IndexError	索引超出序列的范围
KeyError	在字典中查找一个不存在的键
KeyboardInterrupt	用户输入中断键(Ctrl+c)

2.2 抛出自定义异常(常见异常类)

MemoryError 内存溢出(可通过删除对象释放内存)

NameError 尝试访问一个不存在的变量

NotImplementedError 调用尚未实现的方法

OSError 操作系统产生的异常(如打开一个不存在的文件)

OverflowError 数值运算超出最大限制

ReferenceError 弱引用(weak reference)试图访问一个已经被垃圾回收机制回收了的对象

RuntimeError 一般的运行时错误

2.2 抛出自定义异常(常见异常类)

IndentationError	缩进错误
SyntaxError	Python的语法错误
TabError	Tab和空格混合使用
SystemExit	Python编译器进程被关闭
TypeError	类型错误(如：不同类型数据的相互操作)
UnboundLocalError	访问一个未初始化的本地变量
UnicodeError	Unicode相关的错误(ValueError的子类)
UnicodeEncodeError	Unicode编码时的错误

2.2 抛出自定义异常(常见异常类)

UnicodeDecodeError	Unicode解码时的错误
UnicodeTranslateError	Unicode转换时的错误
ValueError	无效的参数或值
ZeroDivisionError	除数为零的错误

2.2 抛出自定义异常

创建异常类和创建其他类一样，必须直接或间接的继承Exception(该类是所有异常类的基类)。自定义异常类的代码类似如下。

```
class CustomizeException(Exception):

    …… # or pass
```

2.2 抛出自定义异常(raise语句)

要抛出异常，可直接使用raise语句，并将一个异常类或实例作为参数(结合第1个例子使用, 负数)。

```
if age < 0 or age > 120:
    raise Exception(年龄不真实)   #等价于 raise(Exception)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception
# 等价于raise(Exception(‘内存溢出'))
>>> raise Exception('内存溢出')
```

2.2 抛出自定义异常(pass语句)

有时, **我们希望发生异常时, 程序暂时什么都不处理, 这时可以使用pass语句来完成**。

pass语句还可以充当占位符, 用于其他的程序代码块中(如if、for、 while循环及函数), 它提醒你在程序的某个地方什么也没有做, 但以后可以加上适当的功能。

3 断言与警告

对于程序测试和调试来讲，有时候会需要个性化的抛出异常或警示信息，这时使用断言和警告，会比用户自定义异常会更加高效。

3.1 断言

Python使用assert语句来申明断言，用来判断语句是否满足特定的条件，否则抛出异常，这在程序测试和调试中添加检查点时很有帮助。

断言一般包含以下部分内容:

assert 条件表达式, 当结果为False时显示的字符串

注意：断言是针对程序员的错误设计的，对于那些可恢复的错误(如文件不存在，用户输入无效数据等)，则应抛出异常。

3.1 断言

```
>>>age = 10

>>>assert 0 <= age <= 120 #  前面的if语句

>>>age = -1

>>> assert 0 <= age <= 120, ‘年龄不真实’
```

注意：在使用Python命令运行.py程序时，**可使用-O参数，以禁用程序中的断言**(如开发已经完成，所有断言都已经通过)，从而提高程序运行性能。

```
# python –O as.py
```

3.2 警告

如果你只想发出**警告**(通常只是打印一条警告消息，一般不影响程序运行)，指出情况偏离了正轨，**可使用模块warnings中的函数warn**。

```
>>> from warnings import warn
>>> warn("年龄可能不真实")
__main__:1: UserWarning:年龄可能不真实
```

3.2 警告

还可以使用模块warnings中的函数**filterwarnings来抑制警告, 并指定抑制类型, 如参数'error'或'ignore'**。

```
>>> from warnings import filterwarnings
>>> filterwarnings("ignore")    #将警告设置为ignore级别
>>> warn("结果有点偏离正常区域")
>>> filterwarnings("error")     # 将警告设置为error级别
>>> warn("结果有点偏离正常区域")
Traceback (most recent call last):
  ……
```

练习

01 完成教材和PPT的练习和例子

02 完成课后编程题2

谢 谢