python复习7 异常

BaseException是所有异常的基类

错误和异常

错误

错误有语法错误、逻辑错误等，错误是可以人为避免的。

异常

BaseException是所有异常的基类

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，异常并不是一定会发生，例如两数相除时，大部分情况下都能正常执行，但除数为0时将会发生异常。

默认情况下，程序运行中遇到异常时将会终止，并在控制台打印出异常出现的堆栈信息。

Python中，异常是一个类，可以处理和使用，通过异常处理可避免因异常导致的程序终止问题。

raise

强制触发指定的异常

```
try:
    # 一些可能引发异常的代码
    pass
except SomeException as e:
    # 处理异常
    # 然后重新引发它
    raise
```

异常处理

- 如果发生的异常与except子句中的类是同一个类或者是它的基类时，则该类与该异常相兼容。

```
class B(Exception):pass
class C(B):pass
class D(C):pass
for cls in [B,C,D]:
```

```
    try:
        raise cls[]
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
# 输出：D  C  B
如果把except B放在最前面，那么会输出B B B
```

- 异常处理，是在指程序设计时便考虑到可能出现的意外情况，为避免因异常导致程序终止，在程序运行出现错误时，让 Python 解释器执行事先准备好的除错程序，进而尝试恢复程序的执行
- 异常处理，可以有效提升用户体验，甚至在程序崩溃前也可以做一些必要的工作，如将内存中的数据写入到文件、关闭打开的文件、释放分配的内存等
- 异常处理语法框架与逻辑：

```
try:
    pass  # do something
except Exception as error:
    pass  # error handling
else:
    pass  # optional, do sth if there's no error
finally:
    pass  # optional, clean up
```

通常将可能发生异常的代码放在try语句中，如发生异常则通过except语句来捕获并采取一些额外处理，如没有发生异常则执行后面的else语句，最后执行finally语句做一些收尾的操作

- 使用Exception兜底，以拦截其他未可知的异常，是一种不错的处理方式。

Else子句

try...except语句具有可选的else子句，该子句适用于try子句没有引发异常但又必须要执行的代码。**else子句如果存在，必须放在所有except子句之后。**

Finally子句

- 清理异常：无论try语句是否触发异常，都会执行finally子句。

With语句

- **预定义的清理操作**：某些对象定义了不需要该对象时要执行的标准清理操作，无论使用该对象的操作是否成功，都会执行清理操作，如打开文件并输出内容

```
for line in open("myfile.txt"):
    print(line, end="")
```

这段代码的问题在于：执行完代码后，文件在一段不确定的时间内处于打开状态。在简单脚本，中这没有问题，但对于较大的程序来说可能会出问题

- **with语句**保证在其子句执行完毕后，即使处理行时遇到问题，都会关闭文件 f

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

内置异常类层次结构

- BaseException是所有内置异常类的基类
- 直接继承BaseException的内置异常类：

>BaseExceptionGroup

>GeneratorExit

>KeyboardInterrupt

>SystemExit

>Exception

- 除了直接继承BaseExcepion的异常外，包括用户自定义的异常，均继承自Exception