一，在python中，程序在执行的过程中产生的错误称为异常

二，在python中，所有的异常类都是Exception的子类，exception类定义在exceptions模块中
该模块在python的内建命名空间中，不需要导入就可以直接使用
1，常见的异常
1）NameError：尝试访问一个未声明的变量，会引发NameError
2）ZeroDivisionError：当除数为0时，会发生ZeroDivisionError异常
事实上，任何数值被0除都会导致ZeroDivisionError异常
3）SyntaxError：当解释权发生语法错误时，会引发SyntaxError异常
SyntaxError是唯一一个不再运行时=发生的异常，它代表着python代码中有一个不正常的结构，使
得程序无法执行，这些错误一般在编译的时候发生，解释权无法把脚本转换为字节码文件
4）IndexError：当使用序列中不存在的索引时，会引发IndexError异常
5）KeyError：当使用映射中不存在的键时，会引发KeyError异常
6）FileNotFoundError：试图打开不存在的文件，会引发FileNotFoundError（python3.2以前是
IOError）异常，只限于与r有关的打开模式
7）AtrributeError：当试图访问未知的对象属性时，会引发AttributeError异常

三，异常处理：python中使用try-except语句处理异常，其中try用于检测异常，except用于捕获异常
1，捕获简单异常
try：
#语句块
except：
#异常处理块
当try里面的某条语句出现错误的时候，程序就不在继续执行try中的其他语句，而是直接执行except里面的处理异常的语句

2，捕获多个异常
try
#语句块
except 异常处理名称1：

#异常处理代码1
except 异常处理名称2：
#异常处理代码2:
.....
当发现try里面的语句出现错误时，会根据出现异常的类型选择执行except语句

如果一个except字句想要捕捉多个异常，并且使用同一种处理方式，在python2中可以在except的后面使用逗号连接多个异常名称，在python3中必须使用元组来表示，

在python2中：except NameError，FileNotFuondError：

在python3中：except（ NameError，FileNotFuondError）：无论出现任何俩个异常的任何一种，都会执行except里面的语句

3，捕获异常的描述信息：
通过一个except字句可以捕捉多个异常，无论出现任何俩个异常的任何一种，都会执行except里面的语句

5,如果try语句没有捕获到任何错误信息，就不再执行任何except语句
6，通常情况下，finally语句用于释放资源

注意：在python2.5之前，finally字句曾经不能与try-except或try-except-else字句一起使用，只能使用try-finally，但是这并不符合大部分程序员的习惯，在python2.5开始，finally字句可以与except字句和else字句自由结合，与try语句联合使用，python中try/except/finally语句的格式如下：

```
try:
#语句块
except A:
#其他A异常处理代码
except :
#没有异常处理代码
else:
#没有异常处理代码
finally:
#最后必须处理代码
```

注意：1）try/except/else/finally出现的顺序必须是try----->except----->else----->finally即所有的except必须在else和finally之前，else必须在finally之前，而except A必须在except之前，否则会出现语法错误

2）else和finally都是可选的，而不是必须的，如果存在的话，finally必须在整个语句的最后位置

3）else语句的存在必须以except X或except语句为前提，如果在没有except语句的try语句中使用else语句会引发语法错误，也就是说，else不能与try-finally配合使用

四，抛出异常

要想程序中主动抛出异常，可以使用raise和assert语句

1，raise语句（使用raise语句显示的抛出触发的异常）

1）当raise语句指定异常的类名时，会创建该类的实例对象，然后引发异常

raise 异常类/raise 异常类对象/raise

2）通过显示的创建异常类的实例，直接使用该实例对象来引发异常

```
index = IndexError（）
raise index
```

3）传递异常（不带任何参数的raise 语句，可以再次引发刚刚发生过的异常，作用是向外传递异常）

```
try:
raise IndexError
except:
print（“出错了”）
raise
```

4）指定异常的描述信息（当使用raise抛出异常时，还能给异常类指定描述信息）

raise IndexError（“索引下标超出范围”）

5）异常引发异常（如果中抛出另外一个异常，可以使用raise -- from语句实现）

```
try:
number
except Exception as exception:
raise IndexError（“下标超出范围”）from exception
```

2，assert语句

assert语句又称作断言，指的是期望用户满足指定的条件，当用户定义的约束条件不满足的时候，它会触发AssertionError异常，所以assert语句可以当作条件式的raise语句，

assert语句的格式为：

assert 逻辑表达式，data　其中data是可选的，assert后面紧跟一个逻辑表达式，相当于条件，data通常是一个字符串，当表达式的结果为False时，作为异常类型的描述信息的使用

assert语句用来收集用户定义的约束条件，而不是捕捉内在的程序设计错误，因为python会自行收集程序的设计错误，会在遇见错误时自动引发异常

六，预定义清理

1，with语句：python2.5开始，如果要在python2.5中使用with语句，需通过

from _future_ import with_statement导入，从2.6版本正式启用，with语句适用于对资源进行访问

的场合，确保不管使用过程中是否发生异常都会执行必要的清理操作，释放资源

语法格式：

with 上下文表达式[as 资源对象]

对象的操作

1）上下文表达式：返回一个上下文管理器对象，若指定了as字句，该对象并不赋值给资源对象，而是将上下文管理器的_enter_（）方法的返回值给资源对象

2）资源对象：可以是单个变量，也可以是元组

3）对象的操作：with语句包裹的代码块，在执行该代码之前，会调用上下文管理器的_enter_（）方法，在执行代码块之后，会执行_exit_（）方法

2，上下文管理器

（上下文管理器是python2.5开始支持的一种语法，用于规定某个对象的使用范围，一旦进入或者离开使用范围，会有特殊的操作被调用）

1）上下文管理协议

①_enter_（self）进入上下文管理器是会调用此方法其返回值被放入with——as语句中as说明指定的变量中

②_exit_（self，type，value，tb）：离开上下文管理器调用此方法，如果有异常出现，type、value、tb分别为异常类型，值，和追踪信息，如果没有异常，3个参数均设为None，此方法返回值为True或者是False，分别指示被引发的异常得到了还是没有得到处理，如果返回False，引发的异常会被传递出上下文

2）上下文管理器：支持上下文管理协议的对象，用于实现_enter_（）和_exit_（）方法，上下文管理器定义执行with语句时要建立的运行时上下文，负责执行with语句块上下文的进入与退出操作

3）运行时上下文：由上下文管理器创建，通过上下文管理器的_enter_（）和_exit_（）方法实现

4）上下文表达式：with语句中关键字with之后的表达式，该表达式要返回一个支持上下文管理协议的对象