# Python实验10 异常

## 一、实验介绍

在这个实验我们学习 Python 的异常以及如何在你的代码中处理它们。

知识点

NameError

TypeError

异常处理（try..except）

异常抛出（raise）

finally 子句

## 二、异常

在程序执行过程中发生的任何错误都是异常。每个异常显示一些相关的错误信息，比如你在 Python3 中使用 Python2 独有的语法就会发生 SyntaxError：

不小心在行首多打了一个空格就会产生 IndentationError：

2.1 NameError

当有人试图访问一个未定义的变量则会发生 NameError。

>>> print(kushal)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'kushal' is not defined

最后一行包含了错误的详细信息，其余行显示它是如何发生（或什么引起该异常）的详细信息。

2.2 TypeError

TypeError 也是一种经常出现的异常。当操作或函数应用于不适当类型的对象时引发，一个常见的例子是对整数和字符串做加法。

>>> print(1 + "kushal")

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

2.3 处理异常

我们使用 try...except 块来处理任意异常。基本的语法像这样：

try:

statements to be inside try clause

statement2

statement3

...

except ExceptionName:

statements to evaluated in case of ExceptionName happens

它以如下方式工作：

首先，执行 try 子句 （在 try 和 except 关键字之间的部分）。

如果没有异常发生，except 子句 在 try 语句执行完毕后就被忽略了。

如果在 try 子句执行过程中发生了异常，那么该子句其余的部分就会被忽略。

如果异常匹配于 except 关键字后面指定的异常类型，就执行对应的 except 子句。然后继续执行 try 语句之后的代码。

如果发生了一个异常，在 except 子句中没有与之匹配的分支，它就会传递到上一级 try 语句中。

如果最终仍找不到对应的处理语句，它就成为一个 未处理异常，终止程序运行，显示提示信息。

下面的例子展示了这些情况：

>>> def get\_number():

... "Returns a float number"

... number = float(input("Enter a float number: "))

... return number

...

>>>

>>> while True:

... try:

... print(get\_number())

... except ValueError:

... print("You entered a wrong value.")

...

Enter a float number: 45.0

45.0

Enter a float number: 24,0

You entered a wrong value.

Enter a float number: Traceback (most recent call last):

File "<stdin>", line 3, in <module>

File "<stdin>", line 3, in get\_number

KeyboardInterrupt

首先我输入了一个合适的浮点值，解释器返回输出这个值。

然后我输入以逗号分隔的值，抛出 ValueError 异常，except 子句捕获之，并且打印出错误信息。

第三次我按下 Ctrl + C ，导致了 KeyboardInterrupt 异常发生，这个异常并未在 except 块中捕获，因此程序执行被中止。

一个空的 except 语句能捕获任何异常。阅读下面的代码：

>>> try:

... input() # 输入的时候按下 Ctrl + C 产生 KeyboardInterrupt

... except:

... print("Unknown Exception")

...

Unknown Exception

2.4 抛出异常

使用 raise 语句抛出一个异常。

>>> raise ValueError("A value error happened.")

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: A value error happened.

我们可以捕获这些异常就像任何其它普通异常一样。

>>> try:

... raise ValueError("A value error happened.")

... except ValueError:

... print("ValueError in our code.")

...

ValueError in our code.

2.5 定义清理行为

try 语句还有另一个可选的 finally 子句，目的在于定义在任何情况下都一定要执行的功能。例如:

>>> try:

... raise KeyboardInterrupt

... finally:

... print('Goodbye, world!')

...

Goodbye, world!

KeyboardInterrupt

Traceback (most recent call last):

File "<stdin>", line 2, in ?

不管有没有发生异常，finally 子句 在程序离开 try 后都一定会被执行。当 try 语句中发生了未被 except 捕获的异常（或者它发生在 except 或 else 子句中），在 finally 子句执行完后它会被重新抛出。

在真实场景的应用程序中，finally 子句用于释放外部资源（文件或网络连接之类的），无论它们的使用过程中是否出错。

## 三、总结

本实验我们知道了异常是什么，然后怎样处理异常以及抛出异常。记得在前面说过的 with 语句吧，它是 try-finally 块的简写，使用 with 语句能保证文件始终被关闭。

异常是什么？其实异常是一种 类，而类将会在下一个实验介绍。