异常处理

- 首先我们要理解什么叫做"**异常**"?
  - 在程序运行过程中，总会遇到各种各样的问题和错误。
  - 有些错误是我们编写代码时自己造成的:
    - 比如语法错误、调用错误，甚至逻辑错误。
  - 还有一些错误，则是不可预料的错误，但是完全有可能发生的:
    - 比如文件不存在、磁盘空间不足、网络堵塞、系统错误等等。
  - 这些导致程序在运行过程中出现异常中断和退出的错误，我们统称为异常。大多数的异常都不会被程序处理，而是以错误信息的形式展现出来。

```
#0不能被作为除数
1/0
异常信息为：ZeroDivisionError: division by zero
```

- 异常的分类:
  - 异常有很多种类型，Python内置了几十种常见的异常，无需特别导入，直接就可使用。
  - 需要注意的是，所有的异常都是异常类，首字母是大写的!

- 异常的危害：
  - 如果程序中一旦出现了异常的语句代码，则该异常就会立即中断程序的运行！
- 因此：
  - 为了保证程序的正常运行，提高程序健壮性和可用性。我们应当尽量考虑全面，将可能出现的异常进行处理，而不是留在那里，任由其发生。
- python处理异常的机制：
  - Python内置了一套try...except...finally...的异常处理机制，来帮助我们进行异常处理。其基本语法是：

```
try:
    pass
except Exception as ex:
    pass
```

  - 机制说明：
    - 首先，执行try子句（在关键字try和关键字except之间的语句）
    - 如果没有异常发生，忽略except子句，try子句执行后结束。
    - 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。

```
try:
    print("发生异常之前的语句正常执行")
    print(1/0)
    print("发生异常之后的语句不会被执行")
except ZeroDivisionError as e:
    print(e)
```

- 如果程序发生的异常不在你的捕获列表中，那么依然会抛出别的异常:

```
# 未捕获到异常，程序直接报错
s1 = 'hello'
try:
    int(s1)
except IndexError as ex:    # 本例为非法值异常，而你只捕获索引异常
    print(ex)
```

- Exception是什么?
  - 在Python的异常中，有一个通用异常：`Exception`，它可以捕获任意异常。
  - 思考：那么既然有这个什么都能管的异常，其他诸如OSError、ZeroDivisionError的异常是不是就可以不需要了?
    - 当然不是！很多时候程序只会弹出那么几个异常，没

有必要针对所有的异常进行捕获，那样的效率会很低。另外，根据不同的异常种类，制定不同的处理措施，用于准确判断错误类型，存储错误日志，都是非常有必要甚至强制的。

- 常见的异常类型：

异常名	解释
AttributeError	试图访问一个对象没有的属性
IOError	输入/输出异常
ImportError	无法引入模块或包；多是路径问题或名称错误
IndentationError	缩进错误
IndexError	下标索引错误
KeyError	试图访问不存在的键
KeyboardInterrupt	Ctrl+C被按下，键盘终止输入
NameError	使用未定义的变量
SyntaxError	语法错误
TypeError	传入对象的类型与要求的不符合
UnboundLocalError	试图访问一个还未被设置的局部变量
ValueError	传入一个调用者不期望的值，即使值的类型是正确的
OSError	操作系统执行错误

- Python的异常机制具有嵌套处理的能力:

- 比如在函数f3()调用f2()，f2()调用f1()，虽然是在f1()出错了，但只需要在f3()进行异常捕获，不需要每一层都捕获异常

```
#函数嵌套出现异常
def f1():
    return 10/0

def f2():
    f1()

def f3():
    f2()

f3()
```

- 函数嵌套处理异常：

```
def f1():
    return 10/0

def f2():
    f1()

def f3():
    f2()

try:
    f3()
except Exception as e:
    print(e)
```

- try...excetion的嵌套
  - 之前我们说过，不是只使用通用的异常类Exception就万事大吉了，为了效率问题，我们需要对常见的异常信息进行精准的捕获，那么如果异常出现在用户层的话，则就需要对用户操作可能会出现的异常进行判断然后精准捕获了，如何操作呢?
    - 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。也就是前面说的嵌套处理能力。直到程序最顶端如果还没有被捕获，那么将弹出异常。

```
try:
    try:
        print("发生异常之前的语句正常执行")
        print(1/0)
        print("发生异常之后的语句不会被执行")
    except ValueError as e:
        print(e)

except ZeroDivisionError as e:
    print("里层没有抓好，只能辛苦我外层了")
```

- 或者使用一个try和多个except的形式：

```
try:
    print("发生异常之前的语句正常执行")
    print(1/0)
    print("发生异常之后的语句不会被执行")
except NameError as e:
    print(e)
except ZeroDivisionError as e:
    print("我是第一个抓取到除零异常的")
except (ValueError,ZeroDivisionError) as
e:
    print("我是备胎")
```

- 或者在except后面跟一个元组，元组中包含多个异常类

```
except (RuntimeError, TypeError,
NameError):
        pass
```

- finally和else子句
  - `try except`语法还有一个可选的else子句，如果使用这个子句，那么必须放在所有的except子句之后。这个子句将在try子句没有发生任何异常的时候执行：

  - 同样的，还有一个可选的finally子句。无论try执行情况和except异常触发情况如何，finally子句都会被执行！

  - 当然，else和finally同时存在时：

```
try:
    pass
except:
    pass
else:
    print("else")
finally:
    print("finally")
```

推导式

- Python语言有一种独特的语法，相当于语法糖的存在，可以帮你在某些场合写出比较精简酷炫的代码。但没有它，也不会有太多的影响。Python语言有几种不同类型的推导式
  - 列表推导式
  - 字典推导式
  - 集合推导式
  - 元组推导式?
- 列表推导式
  - 列表推导式是一种快速生成列表的方式。其形式是用方括号括起来的一段语句，如下例子所示：

```
alist = [x*2 for x in range(1,10)]
print(alist)
############上下两组代码是等效
alist = []
for x in range(1,10):
    alist.append(x*2)
print(alist)
```

- 列表推导式要这么理解，首先执行for循环，对于遍历的每一个x，代入x*x表达式中进行运算，将运算结果逐一添加到一个新列表内，循环结束，得到最终列表。它相当于下面的代码：

```
alist = []
  for x in range(1,10):
    alist.append(x*2)
print(alist)
```

  - 作用：
    - 列表推导式为我们提供了一种在一行内实现较为复杂逻辑的生成列表的方法。其核心语法是用中括号[]将生成逻辑封装起来。当然列表推导式也有多样用法

- 增加条件语句

```
alist = [x * x for x in range(1,11) if x % 2 == 0]
print(alist)

###############相当于如下代码
alist_1 = []
for x in range(1,11):
    if x % 2 == 0:
        alist_1.append(x*x)
print(alist_1)
```

- 多重循环

```
re = [a+b for a in '123' for b in 'abc']
print(re)

############
alist = []
for a in '123':
    for b in 'abc':
        alist.append(a+b)
print(alist)
```

- 字典推导式
  - 既然使用中括号[]可以编写列表推导式，那么使用大括号呢？你猜对了！使用大括号{}可以制造字典推导式！

```
dic = {x:x**2 for x in [2,4,6]}
print(dic)

'''
dic = {}
for x in [2,4,6]:
    dic[x] = x**2
'''
```

  - 注意x: x**2的写法，中间的冒号，表示左边的是key右边的是value。
- 集合推导式

- 大括号除了能用作字典推导式，还可以用作集合推导式，两者仅仅在细微处有差别。

```
a = {x for x in 'aabbccddeeff'}
print(a)
```