Python代码异常处理

在Python中，异常会根据代码的错误自动的被触发，也能由程序员自己触发和截获.

异常由以下四个语句来处理：

1. try/except 代码发生异常，则会被except捕捉
2. try/except/else 代码没有发生异常时执行else下的语句
3. try/except/else/finally 无论代码是否发生异常finally下的语句都会执行
3. raise 手工在代码中触发异常
4. assert 有条件的在代码中触发异常
5. with/as 实现环境管理

一. 异常的角色

1. 错误处理
   每当在运行时检测到程序错误时，Python就会引发异常，可以在程序代码中捕获和响应错误，或者忽略已经发生的异常；Python代码的默认异常处理行为是当代码发生错误的时候，停止程序并打印出错误消息. 如果不想启动这种默认行为，就要写try语句来捕获异常，并且从异常中恢复；当检测到错误的时候，Python会跳到try处理器，而程序会在try之后重新继续执行.

2. 事件通知
   当发生错误的时候，可以在程序之间传递有效状态的信号.

3. 终止行为
   try/finally保证一定会进行指定的操作，无论程序是否异常

In [2]:
```
#示例代码1. 代码发生异常整个程序退出

print 'hello'
print 3 / 0
print 'world'
```

```
hello

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-24594769f156> in <module>()
      2
      3 print 'hello'
----> 4 print 3 / 0
      5 print 'world'
      6

ZeroDivisionError: integer division or modulo by zero
```

In [4]:
```
#示例代码2. 使用try..except..来捕捉异常

try:
    print 3/0    ##这里会报错 因为分母不可以为0
except:
    print 'except'
print 'continue'

#try代码块执行时触发异常,python会跳转到处理器, 执行except下面的语句, try不仅会捕捉异常, 也会从异常中恢复执行.
```

```
except
continue
```

In [6]:
```
#示例代码3. 使用except捕捉特定异常.

try:
    #print 10 / d
    print 10 / 0
except NameError:
    print 'found NameError!'

#except后如果不加异常类型则捕获所有异常, 如果加了异常类型则只捕捉你所指定的异常类型.
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-6-bd1f90b28c87> in <module>()
      3 try:
      4     #print 10 / d
----> 5     print 10 / 0
      6 except NameError:
      7     print 'found NameError!'

ZeroDivisionError: integer division or modulo by zero
```

In [22]:
```
#示例代码4. 代码异常状态打印
try:
    print 10 / d
except NameError as e:
    print e


try:
    print 10 / 0
except Exception as e:
    print e
```

```
name 'd' is not defined
integer division or modulo by zero
```

In [21]:
```
#实例代码5.使用traceback模块, 打印详细的代码错误信息
import traceback

try:
    print 10 / d
except NameError as e:
    traceback.print_exc()
    f = open('123.log', 'w+')
    traceback.print_exc(file=f) #将报错信息写入文件
    f.close()
```

```
Traceback (most recent call last):
  File "<ipython-input-21-4abd9687a2fc>", line 5, in <module>
    print 10 / d
NameError: name 'd' is not defined
```

异常代码编写的细节

语法: try/except/else/finally 其中 else 和 finally是可选的

else: 代码没有发生异常的时候, 执行此处的语句 finally: 不管代码是否发生错误, 此处的语句都执行

In [27]:
```
#示例代码6.
try:
    print 1 / 0
except:
    print 'except'
else:
    print 'this is else!'
print 'after try'
#代码发生除0错误, 该错误被except捕捉, else下的语句不会执行
```

```
except
after try
```

In [28]:
```
#示例代码7.
try:
    print 1 / 1
except:
    print 'except'
else:
    print 'this is else'
finally:
    print 'this is finally!'
print 'after try'

#代码未发生异常, 所以else和finally下的语句都会执行
```

```
1
this is else
this is finally!
after try
```

In [29]:
```
#示例代码8.
try:
    print 1 / 0
except:
    print 'except'
else:
    print 'this is else'
finally:
    print 'this is finally!'
print 'after try'

#代码发生错误, except下的语句和finally下的语句都会执行
```

```
except
this is finally!
after try
```

In [3]:
```
#关于finally的一些思考, 先看下面代码


def func1():
    try:
        print('in func1 try: try statement, will return 1')
        return 1
    finally:
        print('in func1 finally: try statement, will return 2')
        return 2

def func2():
    try:
        print('in func2 try: raise error')
        raise ValueError()
    except:
        print('in func2 except: caught error, will return 1!')
        return 1
    finally:
        print('in func2 finally: will return 3')
        return 3

print(func1())
print(func2())
#这个例子中 func1() 和 func2() 返回什么呢?
```

```
in func1 try: try statement, will return 1
in func1 finally: try statement, will return 2
2
in func2 try: raise error
in func2 except: caught error, will return 1!
in func2 finally: will return 3
3
```

In [6]:
```
def func1():
    try:
        return 1
    finally:
        return 2

def func2():
    try:
        2 / 0
    except:
        return 1
    return 2


def func3():
    try:
        return 1
    except:
        pass
    return 2

print(func1())
print(func2())
print(func3())

#可见 try..except..finally..是一个整体
```

```
2
1
1
```

二. assert断言

语法 assert , , 当test为假的时候, 引发异常 AssertError, 并且data作为错误内容展示出来, 如果test为真则没有发生错误

In [31]:
```
#示例代码9.
assert 1, 'hello!'

assert 0, 'python-training'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-31-99c8646607b3> in <module>()
      2 assert 1, 'hello!'
      3
----> 4 assert 0, 'python-training'

AssertionError: python-training
```

三. raise引发异常

在Python中程序员可以人为的来生成一个异常, 通过raise来生成异常, 默认raise只能触发内置异常, 当然我们也可以定义自己的新的异常, 它特定于我们自己的程序, 用户定义的异常可以通过类编写， 它继承了内置的异常类Exception.

In [32]:
```
#示例代码10. 通过raise人为引发异常
raise ValueError("input value error")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-84d5010572ad> in <module>()
      1 #示例代码10. 通过raise人为引发异常
----> 2 raise ValueError("input value error")

ValueError: input value error
```

In [33]:
```
#示例代码11. 自定义异常

class TaskError(Exception):
    pass

raise TaskError("this task is error!")
```

```
---------------------------------------------------------------------------
TaskError                                 Traceback (most recent call last)
<ipython-input-33-7aedb4c35900> in <module>()
      4     pass
      5
----> 6 raise TaskError("this task is error!")

TaskError: this task is error!
```

四. with/as环境管理器

with/as语句的设计是作为常见的 try/finally 用法模式的替代方案, with/as语句也是用于定义必须执行的终止或清理行为, 无论步骤中是否发生异常.

例如:

通过Python的文件读取

In [34]:
```
#示例代码12.
try:
    f = open('123.log', 'w+')
    for j in f:
        print j
except:
    print 'except..'
finally:
    f.close()
```

In [36]:
```
#示例代码13.

with open('123.log', 'w+') as file:
    for j in file:
        print j

#对open的调用会返回一个简单文件对象, 赋值给变量file, with语句所使用的环境管理协议, 会保证由file所引用的文件对象会自动关闭.
```