异常 Exception

错误 Error
逻辑错误：算法写错了，例如加法写成了减法
笔误：例如变量名写错了，语法错误
函数或类使用错误，其实这也属于逻辑错误
总之，错误是可以避免的

异常 Exception
本意就是意外情况
这有个前提，没有出现上面说的错误，也就是说程序写的没有问题，但是在某些情况下，会出现一些意外，导致程序无法正常的执行下去。
例如open函数操作一个文件，文件不存在，或者创建一个文件时已经存在了，或者访问一个网络文件，突然断网了，这就是异常，是个意外的情况。
异常不可能避免

错误和异常
在高级编程语言中，一般都有错误和异常的概念，异常是可以捕获，并被处理的，但是错误是不能被捕获的。

举例
对比异常和错误

```
with open('testabc') as f:
    pass

# 异常
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    with open('testabc') as f:
FileNotFoundError: [Errno 2] No such file or directory: 'testabc'

def 0A():
    pass

# 错误
  File "test.py", line 3
    def 0A():
        ^
SyntaxError: invalid syntax
```

一个健壮的程序

- 尽可能的避免错误
- 尽可能的捕获、处理各种异常

产生异常

产生：

- raise 语句显式的抛出异常
- Python解释器自己检测到异常并引发它

```
def foo():
    print('before')
    1/0 # 除零异常
    raise Exception('my exception') # raise主动抛出异常
    print('after')
bar()
```

程序会在异常抛出的地方中断执行，如果不捕获，就会提前结束程序（其实是终止当前线程的执行）

异常的捕获

```
try:
    待捕获异常的代码块
except [异常类型]:
    异常的处理代码块
```

```
def foo():
    try:
        print('before')
        c = 1/0
        print('after')
    except:
        print('catch u')
    print('finished')

foo()
print('==== end ====')
```

上例执行到 `c = 1/0` 时产生异常并抛出，由于使用了try...except语句块则捕捉到了这个异常，异常生成位置之后语句将不再执行，转而执行对应的except部分的语句，最后执行try...except语句块之外的语句。

捕获指定类型的异常

```
def foo():
    try:
        print('before')
        c = 1/0
        print('after')
    except ArithmeticError: # 指定捕获的类型
        print('catch u')
    print('finished')

foo()
print('==== end ====')
```

异常类及继承层次

```
# Python异常的继承

BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- RuntimeError
      |    +-- RecursionError
      +-- MemoryError
      +-- NameError
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- SyntaxError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError

```

BaseException及子类

BaseException

所有内建异常类的基类是BaseException

SystemExit

sys.exit()函数引发的异常，异常不捕获处理，就直接交给Python解释器，解释器退出。

```python
import sys

print('before')
sys.exit(1)
print('SysExit')
print('after') # 是否执行?
```

```python
# 捕获这个异常
import sys
try:
    print('before')
    sys.exit(1)
    print('after')
except SystemExit: # 换成Exception能否捕获
    print('SysExit')
print('outer') # 是否执行?
```

如果except语句捕获了该异常，则继续向后面执行，如果没有捕获住该异常SystemExit，解释器直接退出程序。

注意捕获前后程序退出状态码的变化。

KeyboardInterrupt

对应的捕获用户中断行为Ctrl + C

```python
import time

try:
    while True:
        time.sleep(1)
        print('running')
except KeyboardInterrupt:
    print("Ctrl + c")
print('=' * 30)
```

Exception及子类

Exception是所有内建的、非系统退出的异常的基类，自定义异常类应该继承自它

SyntaxError 语法错误
Python将这种错误也归到异常类下面的Exception下的子类，但是这种错误是不可捕获的

```python
def a():
    try:
        0a = 5
    except:
        pass

# 错误
  File "test2.py", line 3
    0a = 5
     ^
SyntaxError: invalid syntax
```

ArithmeticError
所有算术计算引发的异常，其子类有除零异常等

LookupError
使用映射的键或序列的索引无效时引发的异常的基类：IndexError, KeyError

自定义异常类
从Exception继承的类

```
class MyException(Exception):
    pass

try:
    raise MyException()
except MyException: # 捕获自定义异常
    print('catch u')
```

多种捕获

except可以指定捕获的类型，捕获多种异常

```
import sys
class MyException(Exception):
    pass

try:
    a = 1/0
    raise MyException()
    open('t')
    sys.exit(1)
except ZeroDivisionError:
    print('zero')
except ArithmeticError:
    print('arith')
except MyException: # 捕获自定义异常
    print('catch u')
except Exception:
    print('exception')
except: # 写在最后，缺省捕获
    print('error')

print('====end====')
```

捕获规则

- 捕获是从上到下依次比较，如果匹配，则执行匹配的except语句块
- 如果被一个except语句捕获，其他except语句就不会再次捕获了
- 如果没有任何一个except语句捕获到这个异常，则该异常向外抛出
- `except:`称为缺省捕获，缺省捕获必须是最后一个捕获语句

捕获的原则

- 从小到大，从具体到宽泛

as子句

先看一个例子

```
# raise 能抛出什么样的异常？
class A: pass

try:
    # 1/0
    raise 1
    # raise "abc"
    # raise A
    # raise A()
    # raise {}
except: # 写在最后，缺省捕获
    print('catch u')

print('====end====')
```

raise真的什么类型都能抛出吗？

被抛出的异常，应该是异常类的实例，如何获得这个对象呢？使用as子句

```
# raise 能抛出什么样的异常？
class A: pass

try:
    # 1/0
    raise 1
    # raise "abc"
    # raise A
    # raise A()
    # raise {}
except Exception as e: # 写在最后，缺省捕获
    print(type(e), e) # 抛出TypeError类型异常实例

print('====end====')
```

raise语句

- raise后要求应该是BaseException类的**子类或实例**，如果是类，将被**无参实例化**。自定义应该是Exception子类
- raise后什么都没有，表示抛出最近一个被激活的异常，如果没有被激活的异常，则抛类型异常。这种方式较少用，它用在except中

finally子句

finally最终，即最后一定要执行的，try...finally语句块中，不管是否发生了异常，都要执行finally的部分

```
try:
    f = open('test.txt')
except FileNotFoundError as e:
    print('{} {} {}'.format(e.__class__, e.errno, e.strerror))
finally:
    print('清理工作')
    f.close() #
```

注意上例中的f的作用域，解决的办法是在外部定义f

finally中一般放置资源的清理、释放工作的语句

```
f = None
try:
    f = open('test.txt')
except Exception as e:
    print('{}'.format(e))
finally:
    print('清理工作')
    if f:
        f.close()
```

也可以在finally中再次捕捉异常

```
try:
    f = open('test.txt')
except Exception as e:
    print('{}'.format(e))
finally:
    print('清理工作')
    try:
        f.close()
    except Exception as e:
        print(e)
```

语句嵌套和捕获

- 异常语句内部可以嵌入到try块、except块、finally块中
- 异常在内部产生后，如果没有捕获到，就会继续向外部抛出
- 如果外部也没能捕获，将继续再向外部抛出，直至异常代码所在线程，导致线程崩溃
- finally中有return、break语句，则异常就不会继续向外抛出

```
try:
    try:
        1/0
    except KeyError as e:
        print(1, e)
    finally:
        print(2, 'inner fin')
except FileNotFoundError as e:
    print(3, e)
finally:
    print(4, 'outer fin')
```

else子句

```
try:
    ret = 1 * 0
except ArithmeticError as e:
    print(e)
else:
    print('OK')
finally:
    print('fin')
```

else子句，**没有任何异常发生**，则执行

总结

```
try:
    <语句>    #运行别的代码
except <异常类>:
    <语句>    # 捕获某种类型的异常
except <异常类> as <变量名>:
    <语句>    # 捕获某种类型的异常并获得对象
else:
    <语句>    #如果没有异常发生
finally:
    <语句>    #退出try时总会执行
```

1. 如果try中语句执行时发生异常，搜索except子句，并执行第一个匹配该异常的except子句
2. 如果try中语句执行时发生异常，却没有匹配的except子句，异常将被递交到外层的try，如果外层不处理这个异常，异常将继续向外层传递。如果都不处理该异常，则会传递到最外层，如果还没有处理，就终止异常所在的线程
3. 如果在try执行时没有发生异常，如有else子句，可执行else子句中的语句
4. 无论try中是否发生异常，finally子句最终都会执行