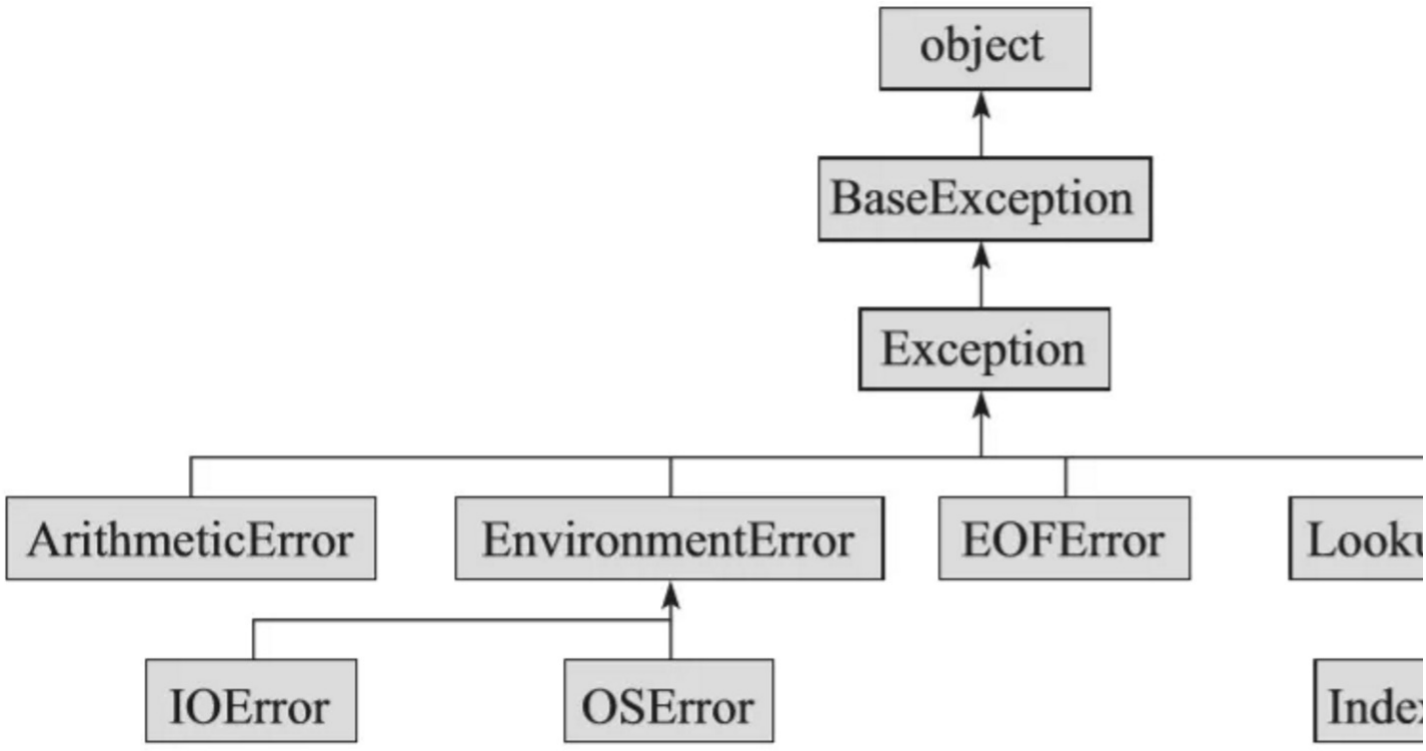
异常就是运行期检测到的错误。计算机语言针对可能出现的错误定义了异常类型，某种错误引发对应的异常时，异常处理程序将被启动，从而恢复程序的正常运行。

1 Python标准异常总结

1. BaseException：所有异常的**基类**
2. Exception：常规异常的**基类**
3. StandardError：所有的内建标准异常的基类
4. ArithmeticError：所有数值计算异常的基类
5. FloatingPointError：浮点计算异常
6. OverflowError：数值运算超出最大限制
7. ZeroDivisionError：除数为零
8. AssertionError：断言语句（assert）失败
9. AttributeError：尝试访问未知的对象属性
10. EOFError：没有内建输入，到达EOF标记
11. EnvironmentError：操作系统异常的基类
12. IOError：输入/输出操作失败
13. OSError：操作系统产生的异常（例如打开一个不存在的文件）
14. WindowsError：系统调用失败
15. ImportError：导入模块失败的时候
16. KeyboardInterrupt：用户中断执行
17. LookupError：无效数据查询的基类
18. IndexError：索引超出序列的范围
19. KeyError：字典中查找一个不存在的关键字
20. MemoryError：内存溢出（可通过删除对象释放内存）
21. NameError：尝试访问一个不存在的变量
22. UnboundLocalError：访问未初始化的本地变量
23. ReferenceError：弱引用试图访问已经垃圾回收了的对象
24. RuntimeError：一般的运行时异常
25. NotImplementedError：尚未实现的方法
26. SyntaxError：语法错误导致的异常
27. IndentationError：缩进错误导致的异常
28. TabError：Tab和空格混用
29. SystemError：一般的解释器系统异常
30. TypeError：不同类型间的无效操作
31. ValueError：传入无效的参数
32. UnicodeError：Unicode相关的异常
33. UnicodeDecodeError：Unicode解码时的异常
34. UnicodeEncodeError：Unicode编码错误导致的异常
35. UnicodeTranslateError：Unicode转换错误导致的异常

异常体系内部有层次关系，Python异常体系中的部分关系如下所示：



2 Python标准警告总结

1. Warning：警告的基类
2. DeprecationWarning：关于被弃用的特征的警告
3. FutureWarning：关于构造将来语义会有改变的警告
4. UserWarning：用户代码生成的警告
5. PendingDeprecationWarning：关于特性将会被废弃的警告
6. RuntimeWarning：可疑的运行时行为(runtime behavior)的警告
7. SyntaxWarning：可疑语法的警告
8. ImportWarning：用于在导入模块过程中触发的警告
9. UnicodeWarning：与Unicode相关的警告
10. BytesWarning：与字节或字节码相关的警告
11. ResourceWarning：与资源使用相关的警告

3 try - except 语句

```
try:
    检测范围
except Exception[as reason]:
    出现异常后的处理代码
```

try 语句按照如下方式工作：

1. 首先，执行try子句（在关键字try和关键字except之间的语句）
2. 如果没有异常发生，忽略except子句，try子句执行后结束。
3. 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。最后执行try语句之后的代码。
4. 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。

【例1】

```
try:
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError:
    print('打开文件出错')

# 打开文件出错
```

【例2】

```
try:
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError as error:
    print('打开文件出错\n原因是：' + str(error))

# 打开文件出错
# 原因是：[Errno 2] No such file or directory: 'test.txt'
```

一个try语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

【例3】

```
try:
    int("abc")
    s = 1 + '1'
    f = open('test.txt')
    print(f.read())
    f.close()
except OSError as error:
    print('打开文件出错\n原因是：' + str(error))
except TypeError as error:
    print('类型出错\n原因是：' + str(error))
except ValueError as error:
    print('数值出错\n原因是：' + str(error))

# 数值出错
# 原因是：invalid literal for int() with base 10: 'abc'
```

【例4】

```
dict1 = {'a': 1, 'b': 2, 'v': 22}
try:
    x = dict1['y']
except LookupError:
    print('查询错误')
except KeyError:
    print('键错误')
else:
    print(x)

# 查询错误
```

try-except-else语句尝试查询不在dict中的键值对，从而引发了异常。这一异常准确地说应属于KeyError，但由于KeyError是LookupError的子类，且将LookupError置于KeyError之前，因此程序优先执行该except代码块。所以，使用多个except代码块时，必须坚持对其规范排序，要从最具针对性的异常到最通用的异常。

【例5】一个 except 子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组。

```
try:
    s = 1 + '1'
    int("abc")
    f = open('test.txt')
    print(f.read())
    f.close()
except (OSError, TypeError, ValueError) as error:
    print('出错了！\n原因是：' + str(error))

# 出错了！
# 原因是：unsupported operand type(s) for +: 'int' and 'str'
```

4 try - except -finally 语句

```
try:
    检测范围
except Exception[as reason]:
    出现异常后的处理代码
finally:
    无论如何都会被执行的代码
```

不管try子句里面有没有发生异常，finally子句都会执行。

如果一个异常在try子句里被抛出，而又没有任何的except把它截住，那么这个异常会在finally子句执行后被抛出。

【例子】

```
def divide(x, y):
    try:
        result = x / y
        print("result is", result)
    except ZeroDivisionError:
        print("division by zero!")
    finally:
        print("executing finally clause")


divide(2, 1)
# result is 2.0
# executing finally clause
divide(2, 0)
# division by zero!
# executing finally clause
divide("2", "1")
# executing finally clause
# TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

5 try - except - else 语句

如果在try子句执行时没有发生异常，Python将执行else语句后的语句。

```
try:
    检测范围
except:
    出现异常后的处理代码
else:
    如果没有异常执行这块代码
```

使用except而不带任何异常类型，这不是一个很好的方式，我们不能通过该程序识别出具体的异常信息，因为它捕获所有的异常。

```
try:
    检测范围
except(Exception1[, Exception2[,...ExceptionN]]]):
   发生以上多个异常中的一个，执行这块代码
else:
    如果没有异常执行这块代码
```

【例子】

```
try:
    fh = open("testfile", "w")
    fh.write("这是一个测试文件，用于测试异常!!")
except IOError:
    print("Error: 没有找到文件或读取文件失败")
else:
    print("内容写入文件成功")
    fh.close()

# 内容写入文件成功
```

【注意】else语句的存在必须以except语句的存在为前提，在没有except语句的try语句中使用else语句，会引发语法错误。

6. raise语句

Python 使用raise语句抛出一个指定的异常。

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')

# An exception flew by!
```

7 习题

1、猜数字游戏

题目描述:

电脑产生一个零到100之间的随机数字，然后让用户来猜，如果用户猜的数字比这个数字大，提示太大，否则提示太小，当用户正好猜中电脑会提示，“恭喜你猜到了这个数是…”。在用户每次猜测之前程序会输出用户是第几次猜测，如果用户输入的根本不是一个数字，程序会告诉用户"输入无效"。

(尝试使用try catch异常处理结构对输入情况进行处理)

获取随机数采用random模块。 


```
import random
num = random.randint(0, 100)
print(num)
print("猜测一个0到100之间的整数.")
for i in range(1, 100):
 try:
  guess = int(input("第" + str(i) + "次猜，请输入一个整形数字:"))
 except ValueError:
  print("输入无效")
  continue
 if guess < num:
  print("太小")
 elif guess > num:
  print("太大")
 else:
  print("恭喜你猜到了这个数是" + str(num))
  break
```