异常与错误

In [2]:
```
#设置全部行输出
```

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

1 错误类型

1.1 语法错误(Syntax Errors)

语法错误, 也就是解析时错误。当我们写出不符合python语法代码时， 在解析时会报SytaxError， 并且会显示出错的哪一行， 并且用小箭头指明最早探测到错误的位置。

程序运行之前就会预先检查语法错误, 因此报出语法错误的时候程序实际上还没有运行.

比如，如果b变量为0，下面的代码就会运行报错:

In [3]:
```
a=2
b=1
if a>b:
    print("a更大")
elif:
    print("b更大")
```

```
  File "<ipython-input-3-c6fe83290c48>", line 5
    elif:
        ^
SyntaxError: invalid syntax
```

1.2 异常 (Exceptions)

即使语句或表达式在语法上是正确的，但在尝试运行时也可能发生错误，运行时错误就叫做异常(Exception)。异常并不是致命问题，因为我们可以在程序运行中对异常进行处理。

（1）除零错误ZeroDivisionError: division by zero

In [4]:
```
10 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-4-a243dfbf119d> in <module>
----> 1 10 / 0

ZeroDivisionError: division by zero
```

（2）命名错误NameError: name 'ok' is not defined

In [5]:
```
ok
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-92a949fd4184> in <module>
----> 1 ok

NameError: name 'ok' is not defined
```

（3）类型错误TypeError

In [6]:
```
2 + '2'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-82150feed70c> in <module>
----> 1 2 + '2'

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

1.2.1 python标准异常

异常名称	描述
BaseException	所有异常的基类

异常名称	描述
SystemExit	解释器请求退出
KeyboardInterrupt	用户中断执行(通常是输入^C)
Exception	常规错误的基类
StopIteration	迭代器没有更多的值
GeneratorExit	生成器(generator)发生异常来通知退出
StandardError	所有的内建标准异常的基类
ArithmeticError	所有数值计算错误的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除(或取模)零 (所有数据类型)
AssertionError	断言语句失败
AttributeError	对象没有这个属性
EOFError	没有内建输入,到达EOF 标记 **好像是读取异常**
EnvironmentError	操作系统错误的基类
IOError	输入/输出操作失败
OSError	操作系统错误
WindowsError	系统调用失败
ImportError	导入模块/对象失败
LookupError	无效数据查询的基类
IndexError	序列中没有此索引(index)
KeyError	映射中没有这个键
MemoryError	内存溢出错误(对于Python 解释器不是致命的)
NameError	未声明/初始化对象 (没有属性)
UnboundLocalError	访问未初始化的本地变量
ReferenceError	弱引用(Weak reference)试图访问已经垃圾回收了的对象
RuntimeError	一般的运行时错误
NotImplementedError	尚未实现的方法
SyntaxError	Python 语法错误
IndentationError	缩进错误
TabError	Tab 和空格混用
SystemError	一般的解释器系统错误
TypeError	对类型无效的操作
ValueError	传入无效的参数
UnicodeError	Unicode 相关的错误
UnicodeDecodeError	Unicode 解码时的错误
UnicodeEncodeError	Unicode 编码时错误
UnicodeTranslateError	Unicode 转换时错误
Warning	警告的基类
DeprecationWarning	关于被弃用的特征的警告

异常名称	描述
FutureWarning	关于构造将来语义会有改变的警告
OverflowWarning	旧的关于自动提升为长整型(long)的警告
PendingDeprecationWarning	关于特性将会被废弃的警告
RuntimeWarning	可疑的运行时行为(runtime behavior)的警告
SyntaxWarning	可疑的语法的警告
UserWarning	用户代码生成的警告

Python所有的错误都是从BaseException类派生的，常见的错误类型和继承关系看这里：

https://docs.python.org/3/library/exceptions.html#exception-hierarchy
(https://docs.python.org/3/library/exceptions.html#exception-hierarchy)

1.3 异常处理语句

当你知道你的代码可能会产生某种异常，但是你却不希望，当这种异常出现的时候导致程序终止，你想要让程序即使出现了异常也能跳过去继续向下运行，这时候你就需要添加try/except 或try/finally语句来处理它。

```
try:
    代码块1(可能会出错的代码块放这里）
except:
    代码块2（如果代码块1出错了，运行代码块2）
else:
    代码块3（如果代码块1没出错了，运行代码块3）
finally:
    代码块4（不论代码块1是否正确，都运行代码块4）
```

1.3.1 try...except语句

```
try:
    代码块1(可能会出错的代码块放这里）
except:
    代码块2（如果代码块1出错了，运行代码块2）
```

In [11]:
```
a=int(input("请输入行驶公里数（km）："))
b=int(input("请输入行驶时间（h）"))
c=int(a/b)
c
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）2
```

Out[11]: 50

In [13]:
```
try:
    a=int(input("请输入行驶公里数（km）："))
    b=int(input("请输入行驶时间（h）"))
    c=int(a/b)
except :        #捕捉所有异常
    print("代码有错")
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）0
代码有错
```

如果我们预先知道代码可能会有哪些错误，可以提前防止好“捕捉器”：

In [15]:
```
try:
    a=int(input("请输入行驶公里数（km）："))
    b=int(input("请输入行驶时间（h）"))
    c=int(a/b)
except ZeroDivisionError :     #只捕捉除数为0的异常
    print("零除错误")
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）a

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-864adc1a5a7e> in <module>
      1 try:
      2     a=int(input("请输入行驶公里数（km）："))
----> 3     b=int(input("请输入行驶时间（h）"))
      4     c=int(a/b)
      5 except ZeroDivisionError :     #只捕捉除数为0的异常

ValueError: invalid literal for int() with base 10: 'a'
```

1.3.2 捕捉异常并打印原因

In [20]:
```
aa=1

try:
    a=int(input("请输入行驶公里数（km）："))
    b=int(input("请输入行驶时间（h）"))
    c=int(a/b)
except ZeroDivisionError as e:        #只捕捉除数为0的异常，并记录异常原因在e
    print("零除错误：",e)  #打印异常原因

aa
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）0
零除错误： division by zero
```

Out[20]:
```
1
```

1.3.3 try...except...else语句

上面我们用try...except语句来捕捉代码块的异常，但是如果代码块没有异常呢？
如果没有代码异常我们想继续运行代码，那么后续的代码我们可以写在else子句中。

In [22]:
```
try:
    a=input("请输入行驶公里数（km）：")
    b=input("请输入行驶时间（h）")
    c=int(a)/int(b)
except (ValueError,ZeroDivisionError) as e_01:          #和int(b)不能为0相比，a和b不能为非数字型字
    print("出现两大错误之一，错误原因是：",e_01)
except Exception as e_02:
    print("出现两大错误以外的错误：",e_02)
else:
    print("\n""代码运行没错")
    print("行驶速度为 %s km/h"%c)
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）1

代码运行没错
行驶速度为 100.0 km/h
```

1.3.4 try...except...else...finally语句

有些语句，无论是否有异常，都需要运行的，我们可以放在finally子句中。

In [24]:
```
try:
    a=input("请输入行驶公里数（km）：")
    b=input("请输入行驶时间（h）")
    c=int(a)/int(b)
except (ValueError,ZeroDivisionError) as e_01:          #和int(b)不能为0相比，a和b不能为非数字型字
    print("出现两大错误之一，错误原因是：",e_01)
except Exception as e_02:
    print("出现两大错误以外的错误：",e_02)
else:
    print("\n""代码运行没错")
    print("行驶速度为 %s km/h"%c)
finally:
    print("\n""全部代码运行完毕")
    print("无论代码有没有错，都会打印上面这句话")
```

```
请输入行驶公里数（km）：100
请输入行驶时间（h）1

代码运行没错
行驶速度为 100.0 km/h

全部代码运行完毕
无论代码有没有错，都会打印上面这句话
```

2 手动引发错误

异常可以用raise语句手动引发，比如：

In [26]:
```
raise ValueError("错了")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-fe568c29f4fd> in <module>
----> 1 raise ValueError("错了")

ValueError: 错了
```

In [102]:
```
try:
    raise ValueError          #将异常ValueError引发
except ValueError:            #然后又将异常ValueError捕捉
    print("成功引发异常")
else:
    print("没成功引发异常")
```

```
成功引发异常
```

为什么要引发异常呢？皮一下就开心了吗？又有什么用呢？

- 其实，有时候我们确实需要把异常引发。
- 比如我们用python模拟抛色子的时候，如果色子点数大于7或者小于1就不对了，这时我们可以设置让它报错：

In [65]:
```
import random
num = random.randint(-10, 10)
print(num)

if not 1<=num<=6:
    raise ValueError('随机生成的骰子不合常理')
else:
    print('您摇出的骰子是', num)
```

```
5
您摇出的骰子是 5
```

3 练习 ¶

既然上面能手动引发ValueError错误，该如何捕捉这类错误并打印异常原因？

In [5]:
```

```

```
8
随机生成的骰子不合常理
程序运行结束，无论是否有报错
```

4 自定义异常类

因为异常其实是一种类（class），捕获一个错误就是捕获到该class的一个实例。因此，错误并不是凭空产生的，而是有意创建并抛出的。Python的内置函数会抛出很多类型的错误，我们自己编写的函数也可以抛出错误。

如果要抛出错误，首先根据需要，可以定义一个错误的class，选择好继承关系，然后，用raise语句抛出一个错误的实例：

In [81]:
```
class dice_error(Exception):
    pass

import random
num = random.randint(-10, 10)
print(num)

try:
    if not 1<=num<=6:
        raise dice_error("骰子点数不合常理")
    else:
        print('您摇出的骰子是', num)
except dice_error as i:
    print("代码报错原因：",i)
```

```
-2
代码报错原因： 骰子点数不合常理
```