第二章　流程控制

本章导读

类与对象是兵器，流程控制语句是兵法。兵法控制兵器，兵器固然重要，但兵法更加重要。兵器有限，而兵法变化万千，但万变不离其宗，顺序、分支和循环。平面几何虽千变万化，但基于5条公理。各种算法异彩纷呈，同样如此。

顺序语句，根据语句出现的先后，依次执行。分支语句和循环语句都依赖于逻辑判断，或走不同的路即分支语句，或重复某条路即循环语句。

Python的流程控制语句简洁优雅。分支语句有if-elif-else，其中elif和else子句不是必须。循环语句有for-in和while，都可以有else子句。在循环语句中，还可以用break提前终止循环，或用continue提前进入下一轮循环。

流程控制语句可以多层嵌套。分支语句中可以嵌套分支语句也可以嵌套循环语句；循环语句中可以嵌套循环语句也可以嵌套分支语句。嵌套层数过多，将增加相关人理解难度。

学习目标：

1. 掌握while与for-in循环的不同应用场景；
2. 掌握循环的else子句执行条件；
3. 掌握break和continue在循环中的功能；
4. 掌握if-elif-else语句中条件选择规则；
5. 掌握占位pass的机构意义以及在开发中的应用；

本章目录

第一节　循环语句

在Python中，循环有while和for-in两种形式。for-in用于遍历成员，while常用于已知循环终点或已知循环条件。for-in循环在上一章已经有所讲解，下面讲述while循环。

1、while循环

while循环的语法如下，其中else子句可选。当条件表达式为真(True)时执行语句体1(又称循环体，可以一句或多句)，当循环正常结束（没执行过break语句）执行语句体2。**注意：else子句不是条件表达式结果为假（False）时执行，与if语句的else子句很不同。**另外，while和else必须以英文冒号结尾，被控制的语句也必须缩进。

```
while 条件表达式:
    语句体1
else:
    语句体2
```

例程2-1是while循环的一个示例，图2-1是其执行效果。例程第5行while i<=10:表示循环体（第6-8行）被执行的条件是i<=10，由于刚到循环时，i的值为1因此满足条件，将第1次执行循环体语句，将i累加到Sum，Sum从0变成1，print()函数输出i值并以逗号结尾，然后执行第8行，i累加1编程2，这时不会执行第9行，而是回到while循环位置，继续判断i值是否满足i<=10。如果满足继续执行循环体，否则终止循环，执行循环体后的语句。由于i值为2，满足i<=10，因此继续执行循环体。

例程2-1

```
第1行  #while循环示例
第2行
第3行  i=1
第4行  Sum=0
第5行  while i<=10:
第6行      Sum+=i
第7行      print(i,end=",")
第8行      i+=1
第9行  else:
第10行     print(Sum)
第11行     print(i)
```

```
1,2,3,4,5,6,7,8,9,10,55
11
```

图2-1　例程2-1执行结果

由于while循环体内没有break语句（严格来说是没有执行break语句），因此else子句将被执行。第10行输出Sum的值，第11行输出i的值。由于while循环结束，因此该值必须是让i<=10不成立的第一个值，此处为11，如图2-1第2行所示。

初学时，很容易少写例程第8行i+=1。如果没有该行代码，i值不会被修改，因此i<=10总是满足，**循环将无限制被执行，成为死循环**。如果循环长时间不能执行结束，如果不是有意为之，很有可能就是死循环，一般可以通过Ctrl+C终止循环，查看代码中循环变量的控制是否合适。

上例可以用for-in循环实现，其效果相同，如例程2-2所示。由于for-in循环是变量in后数据的每一个成员，因此无需控制循环变量，也就没有类似上例的第8行的循环变量控制。另外，for-in循环的最后一个值是符合规则左闭右开区间的最后一个值，因此，else子句输出的i值也不相同，此处是10，而不是如上例一样是11，如图2-2第2行所示。

例程2-2

```
第1行  #用for-in等效实现上例
第2行
第3行  Sum=0
第4行  for i in range(1,11):
第5行      Sum+=i
第6行      print(i,end=",")
第7行  else:
第8行      print(Sum)
第9行      print(i)
```

```
1,2,3,4,5,6,7,8,9,10,55
10
```

图2-2　例程2-2执行结果

有些while循环难以用for-in实现，如例程2-3所示。由于3X+1猜想循环次数不可预期，不像for-in一样由成员数量决定，但知道循环的结束条件(此处是N!=1)，因此难以用for-in实现但可以用while循环实现。

例程2-3

```
第1行  """编程验证3X+1猜想.
第2行  对于一个正整数，如果是偶数就除以2，如果是奇数就乘以3再加上1。
第3行  对得到的数字，重复之前的操作，无论最早的初始正整数是多少，
第4行  这一串数列最终都会进入4,2,1,4,2,1,....这样的循环。
第5行  编写程序，按3X+1猜想，肯定都能演变到1。
```

```
第6行  """
第7行
第8行  N=int(input("请输入N的值:"))
第9行  print(N,end=",")
第10行
第11行 while N!=1:
第12行     if N%2==0: #奇数时
第13行         N=N//2
第14行     else:
第15行         N=N*3+1
第16行     print(N,end=",")
```

```
= RESTART: D:/PKU/课堂教学/2020Spring/FCA/学生问题/1.py
请输入N的值:100
100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,4
0,20,10,5,16,8,4,2,1,
>>>
= RESTART: D:/PKU/课堂教学/2020Spring/FCA/学生问题/1.py
请输入N的值:256
256,128,64,32,16,8,4,2,1,
```

图2-3　例程2-3执行结果

例程2-4是while循环的又一示例。由于不知道循环次数，宜采用while循环。注意第10行没有缩进，如果与第8行对齐，则同样接受while控制，则每次循环都将输出成绩，而现在是全部输入完毕后，一次性输出全部成绩。

例程2-4

```
第1行  #不知学生数量多少。请录入成绩，直到录入负数为止。
第2行
第3行  scoreList=[] #scoreList是List类型，保存成绩
第4行
第5行  score=int(input("请输入成绩："))
第6行  while score>=0:
第7行      scoreList.append(score) #循环内都符合条件，追加
第8行      score=int(input("请输入成绩："))
第9行
第10行 print(score) #输出全部合规成绩
```

2、break语句

break语句用于终止循环，适用于for-in和while循环。当执行到break语句时，当前循环将终止并转到循环体后执行。**注意：break只能终止当前循环。如果有多层循环，则将只能终止break所在的循环。**一般说来，break常与if语句搭档使用。例程2-5是对例程2-4的改造。

在例程中，代码while True:中True是一个布尔型直接量，表示真。对于while循环，只要条件表达式位置的值为真，则循环，因此while True:将永远循环除非循环体内有break。例程第6行代码用于从键盘输入并转换为整数，如果score小于0，则执行break，即循环终止，循环体内break后的语句将不会被执行转到整个循环体后执行。对于例程，只有score不小于0即大于等于0，才会执行第10行。

例程2-5

```
第1行  #不知学生数量多少。请录入成绩，直到录入负数为止。
第2行
第3行  scoreList=[] #scoreList是List类型，保存成绩
第4行
```

```
while True:
    score=int(input("请输入成绩："))
    if score<0:
        break #终止循环
    else:
        scoreList.append(score) #循环内都符合条件，追加

print(score) #输出全部合规成绩
```

如果循环有else子句，且执行到break，则else子句不会被执行。例程2-6由于没有break语句，else子句肯定会被执行，而例程2-7则肯定不会被执行。

例程2-6

```
#不知学生数量多少。请录入成绩，直到录入负数为止。

scoreList=[] #scoreList是List类型，保存成绩
score=int(input("请输入成绩："))

while while score>=0::
    scoreList.append(score) #循环内都符合条件，追加
    score=int(input("请输入成绩："))
else:
    print(score) #输出全部合规成绩
```

在例程2-7中，由于不输入小于0的数，则将永远循环，且学生数量也不可能是无限多。如果要终止循环，在本例则必须执行break语句，因此其else子句将不会被执行。为了能输出合规的成绩，可采用例程2-5，将else子句放在while循环外。

例程2-7

```
#不知学生数量多少。请录入成绩，直到录入负数为止。

scoreList=[] #scoreList是List类型，保存成绩

while True:
    score=int(input("请输入成绩："))
    if score<0:
        break #score小于0时，终止循环
    else:
        score.append(score) #循环内都符合条件，追加
else:
    print(score) #输出全部合规成绩
```

例程2-8用于判断键盘输入数是否为素数，是break的又一示例。当N%i==0即余数为0整除时，则执行第9行并中断循环，此时不会执行第11、12行；当一直没有被整除，此时为素数，则第10行会被执行，此时将执行第11、12行，输出N为素数。如果输入2，即N为2，则第7行相当于for i in range(2,2):，range(2,2)的结果是空，因为没有间隔值，默认为1，相当从2开始，每次增1，由于结束是2，因此为空。此时，while后面的循环体不会被执行，但else子句仍然会被执行。

例程2-8

```
"""判断一个数N是否素数，即只能被1和N本身整除的数。
或者说只要被2到N-1的数整除，即不是素数。
```

```
第3行  """
第4行
第5行  N=int(input("请输入一个正整数："))
第6行
第7行  for i in range(2,N):
第8行      if N%i==0:
第9行          print(f"{N}不是素数")
第10行         break
第11行 else:
第12行     print(f"{N}是素数")
```

break与从函数返回值的return有些相似。但break仅用于终止循环且是当前循环，如果用在函数中，不会结束函数的继续执行；而return则是结束函数的执行，且不管是否在循环中，都将从函数中退出。如例程2-9所示。

自定义函数isPrime()用于判断参数N是否为素数，如果是素数则返回值为True，否则False。第7、8行则N小于2时被执行，满足该条件，则直接返回False，即1、0或者负数，直接返回False，不是素数。第10-13行与例程2-4相似。当N被2到N-1之间任何一个数整除，则直接返False结束函数运行。当循环结束，将执行第13行，能执行到的条件时，是循环没有被break即没有整除过一次，因此此时为True即是素数。第16-18行是isPrime()函数的应用。

例程2-9

```
第1行  """判断一个数N是否素数，即只能被1和N本身整除的数。
第2行  或者说只要被2到N-1的数整除，即不是素数。
第3行  """
第4行
第5行  def isPrime(N): #判断参数N是否素数，如是返回True，否则False
第6行
第7行      if N<2:#如果小于2，直接返回False并结束函数执行
第8行          return False
第9行
第10行     for i in range(2,N):
第11行         if N%i==0:
第12行             return False
第13行     return True
第14行 #------isPrime()函数定义结束------
第15行
第16行 for i in range(1,100):
第17行     if isPrime(i)==True: #输出1-100间的素数
第18行         print(i,end=",")
```

如果将例程2-9第12行改成break，则不会离开函数，且仍然会执行第13行，结果返回值True，即是否素数，都将返回True(N<2除外)，与预期不符。

初学程序设计容易在函数中print()是否为素数，这不是一个好习惯。在该函数中，仅判断是否质数即可，至于如何应用该函数由调用确定为好。同时，定义函数时，目标单纯有利于提高函数质量以及更多应用可能。

3、continue语句

continue用于在while或for-in循环中跳出本次循环，而break是跳出当前整个循环。当执行到continue时，将跳过当前循环中的剩余语句，提前进入下一轮循环。

在例程2-10中，第4、5行代码的含义是当i除以5或3余数为0（即整除）时，即执行continue，即不执行尚未循环的第6行语句，而是提前进入下一轮循环。由于第6行没有执行也就没有输出，因此能被5或3整除的数都没有输出，如第6行注释所示。

例程2-10

```
第1行  #continue示例
第2行
第3行  for i in range(1,20):
第4行      if i%5==0 or i%3==0:
第5行          continue
第6行      print(i,end=",")#输出：1,2,4,7,8,11,13,14,16,17,19,
```

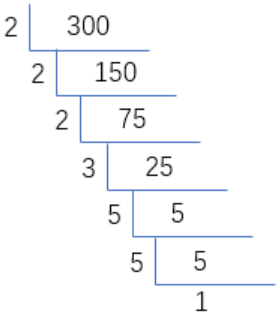

图2-4　分解质因数示例

图2-4是分解质因数示例。从图中可以看出，质因数是从小到大，但某些质因数可能会重复。从图中可以看出，起点是300，终点是1，中间的质因数逐步变大，某些质因数有可能重复。例程2-11是代码实现。

在例程中，第4行申明runResult为list类型，用于保存分解后的质因数。在第7行代码的功能是N>1则循环。i的初始值为2（第6行赋值），第8行根据N%i的状态分支。如果为0即整除时，说明该i值是N的质因数，追加到runResult之中，此时N的值变为N整除以i的值，且i值保持不变进入下一轮循环，继续试探i值能否被继续整除，直到不能整除进入else分支。如果不能整除，则i值增加1（第13行），判断i的下一个整数（比i值大1）能否整除。

例程2-11

```
第1行  #continue示例。分解质因数
第2行
第3行  N=int(input("请输入正整数:"))
第4行  runResult=[] #保存分解后的质因数
第5行
第6行  i=2
第7行  while N>1:
第8行      if N%i==0:#整除则意味着i是其质因数
第9行          runResult.append(i)
第10行         N//=i #N变为除以i后的值
第11行         continue #可能多次整除，继续原质因数试探
第12行     else:
第13行         i+=1
第14行
第15行 print(runResult) #如输入300，输出为：[2, 2, 3, 5, 5]
```

第二节　分支语句

分支语句比较简单，前面已经涉及并多次应用。Python分支语句只有一种形式，即if-elif-else，其语法如下，其中elif和else子句可以有也可以没有，但else子句如果有也只能有一个，而elif子句可以0个、1个或多个。在程序执行过程中，首先判断if后的条件表达式，如果满足，则执行紧随的语句体，否则将查找elif或else子句；如果存在elif且满足条件，则执行紧随的语句体，其他elif不再判断；如果都不满足，则执行else子句(如果存在)。在if-elif-else中，elif比较适用平行条件，即多条件平行对等罗列。另外，if、elif以及else语句都必须以英文冒号结尾。

```
if 条件表达式1:
    语句体1
elif 条件表达式2:
    语句体2
elif 条件表达式3:
    语句体3
elif 条件表达式N-1:
```

```
    语句体N-1
else:
    语句体N
```

例程2-12是if-elif-else的示例，程序比较简单，其中elif罗列每种等级对应的涨工资金额，else列出不在上述之列的金额。例程2-13是例程2-12的等效实现，从中可以看出，如果全部采用if-else，则嵌套很深，让人难以理解，代码也不直观优雅。

例程2-12

```
"""涨工资。初级涨500元，中级涨700元
副高涨1200，正高涨2000元，其他300元"""

titleLevel=input("请输入职称等级，没有等级输入其他。")

if titleLevel=="正高":
    print("涨工资2000元")
elif titleLevel=="副高":
    print("涨工资1200元")
elif titleLevel=="中级":
    print("涨工资700元")
elif titleLevel=="初级":
    print("涨工资500元")
else:
    print("涨工资300元")
```

例程2-13

```
#涨工资。初级涨500元，中级涨700元，副高涨1200，正高涨2000元，其他300元

titleLevel=input("请输入职称等级，没有等级输入其他。")

if titleLevel=="正高":
    print("涨工资2000元")
else:
    if titleLevel=="副高":
        print("涨工资1200元")
    else:
        if titleLevel=="中级":
            print("涨工资700元")
        else:
            if titleLevel=="初级":
                print("涨工资500元")
            else:
                print("涨工资300元")
```

例程2-14用于“列出能被3、5以及3和5同时整除的数”，但运行时第8、9两行永远不会被执行，虽然15能被3和5整除，但由于if和elif只能被执行一个，而当i为15能被3整除，执行第4、5行后，其他elif以及else都不会被执行，在例程即进入下一轮循环，即转到if-elif-else之后执行，在本例则进入下一轮循环。

例程2-14

```
#列出能被3、5以及3和5同时整除的数

for i in range(1,20):
    if i%3==0:
        print(f"{i}能被3整除")
    elif i%5==0:
        print(f"{i}能被5整除")
    elif i%3==0 and i%5==0:
        print(f"{i}能被3和5同时整除")
    else:
        print(f"{i}不能被3或5整除")
```

在使用elif子句时，要注意条件是否独立，是否有交叉，如果不是刻意为之以达到某种效果，通常将带来错误，宜谨慎，例程2-14可修改如例程2-15、例程2-16、例程2-17。

例程2-15只是例程2-14的顺序调整，但因为先判断能否被3和5整除，所以不会有问题，逻辑清晰，没有嵌套，易于理解。例程2-16按能否被3或5整除，分为两种情况，然后分类处理，逻辑也比较清晰，有一层嵌套。例程2-17有多层嵌套，虽然逻辑清晰，但相对难以理解。平衡逻辑清晰和代码简洁，非常重要。

例程2-15

```
#列出能3、5以及3和5同时整除的数

for i in range(1,20):
    if i%3==0 and i%5==0:
        print(f"{i}能被3和5同时整除")
    elif i%5==0:
        print(f"{i}能被5整除")
    elif i%3==0:
        print(f"{i}能被3整除")
    else:
        print(f"{i}不能被3或5整除")
```

例程2-16

```
#列出能3、5以及3和5同时整除的数

for i in range(1,20):
    if i%3==0 or i%5==0:
        if i%3==0 and i%5==0:
            print(f"{i}能被3和5同时整除")
        elif i%5==0:
            print(f"{i}能被5整除")
        elif i%3==0:
            print(f"{i}能被3整除")
    else:
        print(f"{i}不能被3或5整除")
```

例程2-17

```
#列出能3、5以及3和5同时整除的数

```

```
第3行  for i in range(1,20):
第4行      if i%3==0 or i%5==0:
第5行          if i%3==0 and i%5==0:
第6行              print(f"{i}能被3和5同时整除")
第7行          else
第8行              if i%5==0:
第9行                  print(f"{i}能被5整除")
第10行             else:
第11行                 print(f"{i}能被3整除")
第12行     else:
第13行         print(f"{i}不能被3或5整除")
```

当选择语句如果仅控制一条语句，则可以同行书写，如例程2-18所示，代码第2行return -v受到if v<0控制，因为仅有一行，所以可以写在同行。当然一般说来，写在两行并缩进是更好选择。

例程2-18

```
第1行  def ABS(v) #求v的绝对值
第2行      if v<0:return -v
第3行      return v
```

Python还提供了一种if-else辩题，或可以称为条件运算符，其语法格式如下，如果conditionalExpression(条件表达式)的值为True时，该表达式的值为trueStatement，否则为falseStatement，注意if或else之后都没有英文冒号。在例程2-19中，代码中的v if v>0 else -v即为条件运算符，其含义是当if后的条件表达式v>0为真时，其值为if前的值v，否则为else后的值-v。

```
trueStatement if conditionalExpression else falseStatement
```

例程2-19

```
第1行  def ABS(v):
第2行      return v if v>0 else -v
```

第三节　pass占位

观察例程2-20会发现有多处使用了pass，但该程序能运行，没有语法错误，但也没有什么功能作用，但如果删除pass，将导致错误。pass在其中仅仅是占位，确保结构完整，否则将导致错误。

例程2-20

```
第1行  #pass语句示例
第2行  def IamFx():
第3行      pass
第4行
第5行  for i in range(1,100):
第6行      pass
第7行
第8行  if 5>2:
第9行      pass
第10行 else:
第11行     pass
```

pass虽然没有具体功能仅具结构意义，但善用pass颇有意义。如例程2-21所示，在求1-100之间的素数时，虽然没有实现具体代码，但已有架构，在此基础上可以根据实际，逐步完善代码。完成其中一块代码时，可以不考虑其他部分，每完成一部分，即会

更接近目标。

例程2-21

```
第1行  #1-100之间的素数
第2行  def isPrime(N):
第3行      pass
第4行
第5行  print(isPrime(13)) #测试isPrime()函数是否正确，测试完毕后删除或注释掉
第6行
第7行  for i in range(1,101):
第8行      pass
```

由于例程2-21的isPrime()函数没有返回值，所以第5行将输出None。如果将isPrime()函数的返回值修改为True，则输出将是True。例程2-22是对例程2-21的升级，运行该程序，会发现第5行输出True，第7、8、9行将把所有数都输出为素数。通过观察会发现，只要写好isPrime()函数，则整个程序就算完成，如果存在错误，也仅仅是isPrime()函数部分存在问题，利于定位错误而不是漫无目的。

例程2-22

```
第1行  #1-100之间的素数
第2行  def isPrime(N):
第3行      return True
第4行
第5行  print(isPrime(13)) #测试isPrime()函数是否正确，测试完毕后删除或注释掉
第6行
第7行  for i in range(1,101):
第8行      if isPrime(i)==True:
第9行          print(f"{i}是素数!")
```

例程2-23是实现isPrime()函数后的代码，在开发过程中，可以修改第11行isPrime(13)中的13，测试不同参数的效果，测试完毕后，注释掉(行首加#即可)或删除。

例程2-23

```
第1行  #1-100之间的素数
第2行  def isPrime(N):
第3行      if N<2:
第4行          return False
第5行
第6行      for i in range(2,N):
第7行          if N%i==0:
第8行              return False
第9行      return True
第10行
第11行 print(isPrime(13)) #测试isPrime()函数是否正确，测试完毕后删除或注释掉
第12行
第13行 for i in range(1,101):
第14行     if isPrime(i)==True:
第15行         print(f"{i}是素数!")
```

善于利用占位pass，有利于规划开发过程。

第四节　异常处理

异常处理是特殊的流程控制语句，是发生异常(Exception)时的流程控制。注意：其他程序设计语言一般不把异常处理归在流程控制语句之中。例程2-24是触发异常的一个示例。如图2-5所示，当除数为0时，将触发名为“ZeroDivisionError”的异常。如果是重要的执行过程，如果仅仅是小小的失误就导致错误或异常而导致程序运行中断，将是非常可惜甚至严重后果。Python提供了较好的异常处理机制，如例程2-25所示。

例程2-24

```
第1行  #异常处理示例，触发异常
第2行
第3行  M=int(input("请输入整数："))
第4行  N=int(input("请输入整数："))
第5行  print(f"{M}/{N}={M/N}")
```

```
请输入整数：100
请输入整数：0
Traceback (most recent call last):
  File "C:/PythonCode/Ex00.py", line 5, in <module>
    print(f"{M}/{N}={M/N}")
ZeroDivisionError: division by zero
```

图2-5　例程2-24执行结果

在例程2-25中，try-except就是异常处理语句，和其他流程控制语句一样，语句结尾必须有英文冒号。在Python异常处理中，将可能发生异常的语句放在try之后，如果发生异常将执行except后的语句。在例程中，虽然仅仅是print()出提示信息，但既然能执行该语句，则说明已能捕获异常，且提供了能处理异常的代码。

例程2-25

```
第1行  #异常处理示例，捕获异常
第2行
第3行  M=int(input("请输入整数："))
第4行  N=int(input("请输入整数："))
第5行  try:
第6行      print(f"{M}/{N}={M/N}")
第7行  except ZeroDivisionError:
第8行      print("Warning：除数不能为0!")
```

```
请输入整数：100
请输入整数：0
Warning：除数不能为0!
```

图2-6　例程2-25执行结果

例程2-26相对于例程2-25增加了else子句，当没有异常发生时，将执行else子句，其执行效果如图2-7所示。

例程2-26

```
第1行  #异常处理示例，捕获异常
第2行
第3行  M=int(input("请输入整数："))
第4行  N=int(input("请输入整数："))
第5行  try:
第6行      Result=M/N
第7行  except ZeroDivisionError:
第8行      print("Warning：除数不能为0!")
```

```
第9行    else:
第10行       print(f"{M}/{N}={Result}")
```

```
请输入整数：100
请输入整数：20
100/20=5.0
>>>
============== RESTART:
请输入整数：20
请输入整数：0
Warning：除数不能为0!
```

图2-7　例程2-26执行结果

例程2-27的异常处理有finally子句，无论是否发生异常，该语句都将被最后执行。在例程中，如果文件打开（第4行代码）失败，将执行第10-11行。如果文件正常打开，将执行第5-9行。即便文件写入（第6行）成功或失败，但7-9行代码都会被执行，因为文件已经被打开。在文件操作中，加入异常处理语句，是良好的工程习惯。

例程2-27

```
第1行    #异常处理示例，异常嵌套
第2行
第3行    try:
第4行        myFile=open("testfile","w")
第5行        try:
第6行            print("这是一个测试文件，用于测试异常!!",file=myFile)
第7行        finally:
第8行            print("关闭文件!")
第9行            myFile.close()
第10行   except IOError:
第11行       print("Error: 没有找到文件或读取文件失败")
```

在Python的异常处理中，except子句可以有多个，但else和finally子句如果有，只能有一个，如例程2-28所示。在例程中，有多个except子句，每个子句处理不同类型的错误。第11行的except子句后没有具体的错误，则如果发生异常但如果不是OSError或ValueError则将会被执行。多个异常也可以写在一个Except子句之后，如例程2-29所示。

例程2-28

```
第1行    #异常处理示例，异常嵌套
第2行
第3行    try:
第4行        f = open('myfile.txt')
第5行        s = f.readline()
第6行        i = int(s.strip())
第7行    except OSError:
第8行        print("操作系统错误!")
第9行    except ValueError:
第10行       print("参数错误!")
第11行   except:
第12行       print("其他错误!")
```

例程2-29

```
第1行    #异常处理示例，异常嵌套
第2行
```

```
第3行  try:
第4行      f = open('myfile.txt')
第5行      s = f.readline()
第6行      i = int(s.strip())
第7行  except (OSError,ValueError):
第8行      print("操作系统或参数错误!")
第9行  except:
第10行     print("其他错误!")
```

可以捕获异常，也可以用raise语句触发异常，如例程2-30所示，第4行的raise语句将触发一个ZeroDivisionError异常。在开发实践中，定义函数时如果预计到可能产生异常即终止程序的运行，非常不符合工程需要。因为函数执行时，可能大量的其他工作已经进行，如果仅调用某个函数即终止程序运行，将可能带来或大或小的风险，但触发异常符合工程规范，将异常发生后的处理交予函数调用处理更加合理。

例程2-30

```
第1行  #异常处理示例，触发异常
第2行
第3行  def Div(M,N):
第4行      if N==0:raise ZeroDivisionError
第5行      return M/N
第6行
第7行  try:
第8行      print(Div(10,2))
第9行      print(Div(10,0))
第10行 except ZeroDivisionError:
第11行     print("除数不能为0!")
```