# 第3章 Python程序控制结构

**学习目标**

- 理解表达式的逻辑值。
- 熟练掌握选择结构的语法及应用。
- 熟练掌握循环结构的语法及应用。
- 熟练掌握异常处理结构的语法及应用。

写计算机程序代码类似于用自然语言写文章，对特定的事物运动状态及规律进行描述，只不过计算机程序是送给计算机阅读执行的一个任务书。计算机程序无须描写与修饰，单纯按照设定的逻辑顺序执行人们事先编写好的动作语句序列，完成整个程序工作。任何编程语言为了描述语句的执行过程，都会提供一套描述的机制，这种机制称为“控制结构”，用来控制语句的执行过程。

Python 提供的控制结构有顺序结构、选择结构和循环结构 3 种。顺序结构是最基本的执行流程，也是默认的程序执行流程，即在一个没有其他控制结构的程序中，语句的执行顺序为从第一条语句依次执行到最后一条语句，这种结构很好理解，但是如果是复杂的问题，这种结构就不能很好地处理了。选择结构是程序的执行流程分为多条路径，程序运行时会根据条件判断执行哪一条路径下面的程序代码。循环结构多用于在满足条件时反复执行某项任务，有了这种结构计算机就能反复、快速地执行某些计算任务，但要注意的是，每次循环过后，循环主体变量会发生变化，程序会根据循环条件判断是否继续执行循环语句。

顺序结构是构成程序的主要结构，Python 中描述顺序结构的语句包括输入语句、计算语句和输出语句三种，这在后续的内容中都有涉及，这里不再进行单独描述。

## 3.1 选择结构

生活中处处充满选择，如大学生毕业是选择继续深造考取研究生还是直接就业；期末考试有多个科目，如何确定适合自己的科目复习顺序等。编写程序时，选择结构是根据不同条件来决定是否执行某些特定的代码，依据解决问题逻辑不同，选择结构分为单分支结构、双分支结构、多分支结构以及嵌套结构。

在介绍选择结构之前，有一个内容需要了解，那就是条件表达式，也称为条件语句。在 Python 中，条件语句是经常用到的语句，如在选择结构、循环结构中，经常会根据条件表达式的值来确定下一步的动作（或执行流程），选择结构会根据条件的不同而执行不同的代码，

循环结构则根据条件的真假判断是退出当前的循环，还是继续执行循环体代码。Python 中所有的合法表达式都可以作为条件表达式。

### 3.1.1 单分支选择结构

单分支选择结构是用来描述只有一个条件来决定程序是否执行。语法格式如下。

```
if <条件表达式>:
   语句块
```

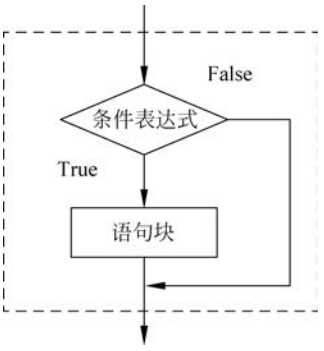


图 3-1 单分支选择结构

在这个语法中，if 是关键字，用“< >”括起来的条件表达式是必有项，为一个逻辑表达式或者其值可以转换为逻辑值的其他类型表达式。语句块由一条或多条语句组成，如果是多条语句，则使用回车进行语句分隔，代表当条件表达式为真时要执行的程序内容，应注意语句块与 if 的语法缩进（标准为 4 个字符）。另外，条件表达式后面的“:”是必需的。

单分支选择结构语句执行流程如图 3-1 所示。如果条件表达式值为 True，则执行语句块内容；否则即为 False，不执行语句块中的任何内容，跳过语句块继续执行下一条语句。

**【例 3-1】** 根据输入数值判断，如果输入的是偶数则输出该偶数。

```
a = input("请输入一个整数")
if int(a) % 2 == 0:
    print("您输入的数是偶数",a)
```

根据输入的整数值判断是奇数还是偶数，当输入 34 时，结果为：

```
请输入一个整数 34
您输入的数是偶数 34
```

当输入 3 时，结果为：

```
请输入一个整数 3
```

可见，当输入 3 时，程序后续没有任何结果提示，原因是不满足 if 条件表达式为真值的条件，不会执行语句块里的 print()函数内容，语句块后面也没有其他语句，因此不会有其他输出内容。

注意：input 语句输入的内容都是字符串类型，进行数值运算时需要转换为所需的数值类型，本例中将输入的整数赋给 a，运算时使用内置函数 int()变为整型后除以 2 取余数判断是否与 0 相等。

### 3.1.2 双分支选择结构

双分支选择结构也是描述只有一个条件的情况，与单分支选择结构不同的是，其程序的执行流程包含当条件表达式为 False 时要执行的语句内容。语法格式如下。

```
if <条件表达式>:
   语句块 1(条件表达式为 True 时执行的语句块)
else:
   语句块 2(条件表达式为 False 时执行的语句块)
```

双分支选择结构语句执行流程如图 3-2 所示。if 和 else 都是关键字，后面都带有“:”，根据条件表达式的真假判断来执行语句块 1 还是语句块 2，两个语句块必定有一个会被执行。

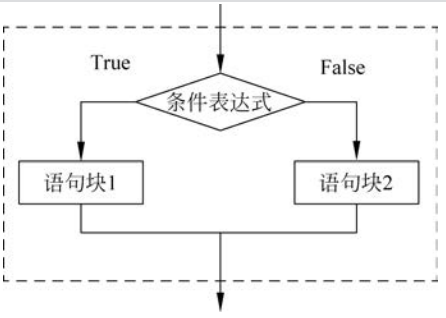

图 3-2 双分支选择结构

**【例 3-2】** 根据输入数值判断，如果输入的是奇数则打印“您输入的数是奇数”，否则打印“您输入的数是偶数”，同时无论奇偶，都将该数显示出来。

```
a = input("请输入一个整数")
if int(a) % 2 == 0:
    print("您输入的数是偶数",a)
else:
    print("您输入的数是奇数",a)
```

当输入 3 时，结果为：

```
请输入一个整数 3
您输入的数是奇数 3
```

当输入 2 时，结果为：

```
请输入一个整数 2
您输入的数是偶数 2
```

### 3.1.3 多分支选择结构

多分支选择结构是为了解决多条件判断而设计的，适用于多情况讨论问题。语法格式如下。

```
if <条件表达式 1>:
   语句块 1(条件表达式 1 为 True 时执行的语句块)
elif  <条件表达式 2>:
     语句块 2(条件表达式 2 为 True 时执行的语句块)
…
elif  <条件表达式 n>:
     语句块 n(条件表达式 n 为 True 时执行的语句块)
[else:
     语句块 n + 1(不满足上述任何条件时执行的语句块)]
```

多分支选择结构如图 3-3 所示。当程序执行 if…elif…else…语句的时候，首先判断条件表达式 1 是否为 True，如果为 True 则执行语句块 1，然后结束整个选择结构，如果为 False 则继续判断条件表达式 2 是否为 True，如果条件表达式 2 为 True，则执行语句块 2，然后结束整个选择结构，如果为 False 则继续判断条件表达式 3 是否为 True，以此类推，直

到判断所有的条件表达式都不满足，则执行 else 里的语句块 $n+1$，然后结束整个选择结构。注意 else 语句不是必需的，可根据实际需要决定是否有这个语句。另外，所有的条件最多只执行一次，不要将 elif 写成 else 或者 elseif。

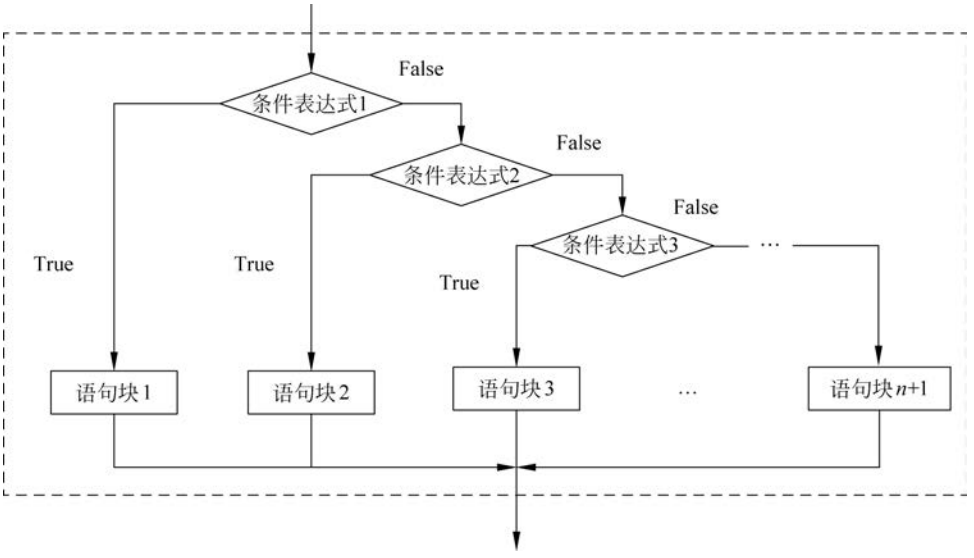


图 3-3　多分支选择结构

**【例 3-3】** 根据输入的 0～100 的整型数，给出五级分制的结果输出。

```
score = eval(input("请输入课程的成绩数值"))
if score >= 90:
    print("优秀")
elif score >= 80:
    print("良好")
elif score >= 70:
    print("中等")
elif score >= 60:
    print("及格")
else:
    print("不及格")
```

这个例子是先从大于或等于 90 分开始判断的，当然也可以先从大于或等于 60 分开始判断，也可以从小于 60 分判断，方法可以任选。此问题相当于对 0～100 的数进行归类，共分成优秀、良好、中等、及格和不及格 5 类。注意：此例题中用到了内置 eval()函数，确保 score 为数值数据。

### 3.1.4　嵌套选择结构

现实中往往存在着条件中又带有条件、决策判断中又有决策判断的情况，Python 提供了嵌套选择结构来描述此种情况，前面介绍的多分支选择结构可以理解为简单的嵌套选择。嵌套选择结构的语法格式如下。

```
if  <条件表达式 1>:                    # 第 1 层判断
    语句块 1
```

```
    if  <条件表达式 2>:                    # 第 2 层判断
        语句块 2
        if  <条件表达式 3>                 # 第 3 层判断
            语句块 3
        else:                              # 第 3 层判断
            语句块 4
    elif <条件表达式 4>:                   # 第 2 层判断
        语句块 5
        if  <条件表达式 5>:                # 第 3 层判断
            语句块 6
        elif  <条件表达式 6>:              # 第 3 层判断
            语句块 7
        else:                              # 第 3 层判断
            语句块 8
else:                                      # 第 1 层判断
    语句块 9
    if  <条件表达式 7>:                    # 第 2 层判断
        语句块 10
```

## 3.2 循环结构

循环结构是在一定条件下,重复执行某段代码的一种编码结构。Python 提供的循环结构有两种形式:一种是事先知道循环次数的编码结构,使用 for 循环;另一种是事先不知道循环次数,只能等到条件达到退出循环时才停止循环的编码结构,使用 while 循环。

### 3.2.1 for 循环

for 循环是一种遍历循环(也称迭代循环),重复执行相同的操作。此种循环需要事先知道循环的次数,因此也称为记数循环。经常用于遍历字符串、列表、字典等数据结构时。基本语法如下。

```
for  <循环变量>  in  <遍历结构>:
    语句块
```

for 和 in 是遍历循环的关键字,< >中的部分是必有项,不可省略,彼此间使用空格进行分隔。循环变量可以有一个,也可以有多个(彼此有关联,同时递进),但必须是 Python 的合法标识符。for 循环流程图如图 3-4 所示。

首先确定循环变量,为循环变量指定遍历的范围,将遍历指针指向第一个元素,之后判断循环变量是否存在于遍历结构中(也可以理解为是否超出遍历范围),如果在,就执行循环体语句块,同时推进循环变量向前移动;否则结束循环,继续执行循环结构后面的语句。

**【例 3-4】** 计算 1～100 所有数的和。

```
sum = 0
for i in range(1,101):
    sum + = i
print('1～100 所有数的和为: ',sum)
```

结果为:

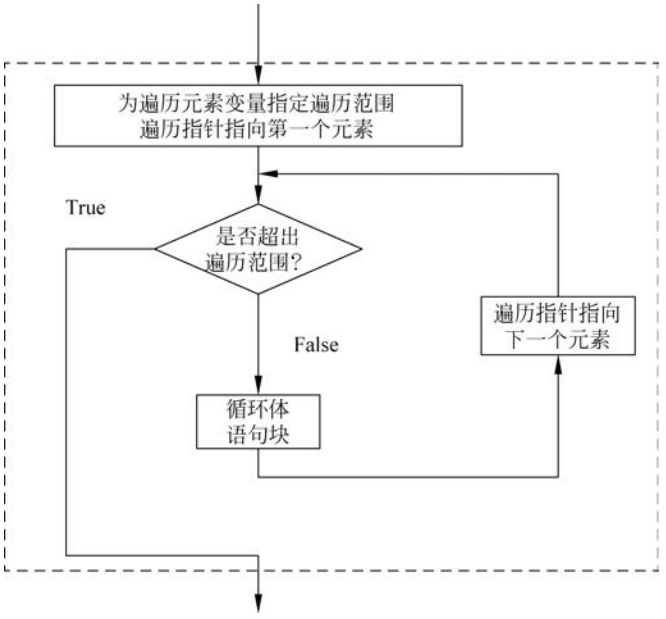


图 3-4　for 循环流程图

```
1～100 所有数的和为: 5050
```

例 3-4 用到了 range()函数，要注意这个函数的右侧边界问题，计算 1～100 所有数的和，因此右侧的边界设置为 101，sum＋＝i 为循环体。

**【例 3-5】** 打印输出下面的图形。

```
    &
   & &
  & & &
 & & & &
& & & & &
```

分析要求：图案主要是由 & 和空格组成，总共 5 行，所以要执行 5 次循环，每一次要确定打印几个 & 和几个空格。细心观察发现，第一行有四个空格，一个 &；第二行有三个空格，两个 &；第三行有两个空格，三个 &；第四行有一个空格，四个 &；第五行无空格，五个 &。设计程序代码如下。

```
for i in range(1,6,1):                  # 确定遍历范围 1～5,步长为 1(默认),可不写
    print(" " * (6-i),end = "")         # 确定每行空格数
    print("& " * i,end = "")            # 确定每行 & 个数
    print()
```

## 3.2.2　while 循环

while 循环是一种事先不知道循环次数的无限循环，但知道循环的终止条件。语法格式如下。

```
while <循环条件>:
    <语句块>
```

while 是关键字，循环条件和语句块都是必有项，while 和循环条件之间有一个空格，循环语句执行时，首先判断循环条件，当条件为 True 时执行语句块；当条件为 False 时，循环终止。while 循环流程图如图 3-5 所示。

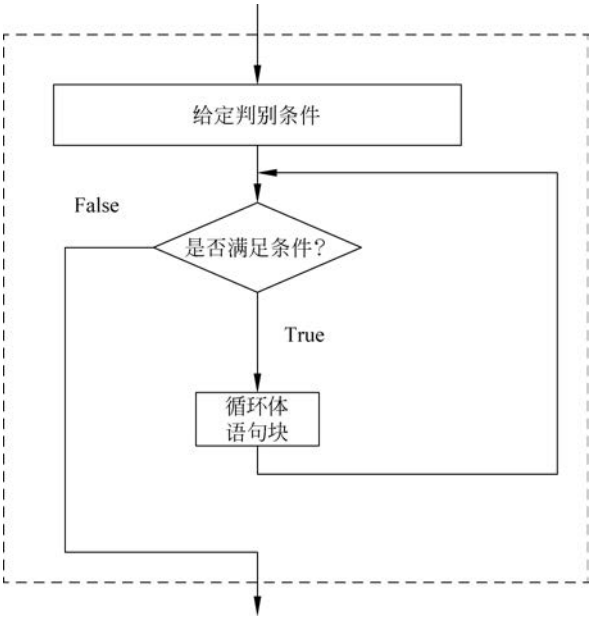


图 3-5 while 循环流程图

从图 3-5 可以看出 while 循环的流程，首先需要给出判别条件，然后进行条件判断，如果满足判别条件就执行循环体语句，否则就跳出循环，执行后面的语句。注意：如果循环条件一直为 True，那么就会陷入死循环，可以使用 Ctrl+C 组合键来终止死循环或者关闭开发环境。

**【例 3-6】** 使用 while 循环实现例 3-4。

```
sum = 0
i = 1
while i <= 100:
    sum += i
    i += 1
print('1～100 数的和为: ',sum)
```

从程序可以看出，首先对循环变量 $i$、求和变量 sum 赋初值，之后判断 while 循环条件是否为 True，如果为 True 则执行循环体语句，更新变量 sum 的值，同时推进循环变量 $i$ 前进(加 1)，当 $i$ 为 101 时，循环条件为 False，则退出循环，执行 print()函数，将 1～100 的和显示出来。

while 循环还有一种扩展模式。语法格式如下。

```
while <循环条件>:
    <语句块 1>
else:
    <语句块 2>
```

此种模式较前一种多了一个 else 语句，当 while 循环正常执行之后，程序会继续执行 else 语句中的内容。

**【例 3-7】** while 循环扩展模式计算 1～10 的和。

```
sum = 0
i = 1
while i <= 10:
    sum += i
    i += 1
    print('1 到 %d 之间数的和为：%d' %(i-1,sum))
else:
    print('求和完毕!')
```

结果为：

```
1 到 1 之间数的和为：1
1 到 2 之间数的和为：3
1 到 3 之间数的和为：6
1 到 4 之间数的和为：10
1 到 5 之间数的和为：15
1 到 6 之间数的和为：21
1 到 7 之间数的和为：28
1 到 8 之间数的和为：36
1 到 9 之间数的和为：45
1 到 10 之间数的和为：55
求和完毕!
```

本例中用到了 Python 格式化输出功能，除了使用%外，还可以使用 format 或者 f 方法来实现格式化输出，读者可自行学习相关知识。

### 3.2.3 嵌套循环

嵌套循环是指在一个循环中往往会因程序需要嵌入另外一个或多个完整的循环结构，这样就形成了一个多层循环。Python 也支持循环嵌套机制，for 循环和 while 循环可以互相嵌套，两种循环的嵌套组合的四种形式如图 3-6 所示。

| | |
|---|---|
| for <循环变量> in <遍历结构>：<br>…<br>for <循环变量> in <遍历结构>：<br>… | for <循环变量> in <遍历结构>：<br>…<br>while <循环变量>：<br>… |
| while <条件>：<br>…<br>for <循环变量> in <遍历结构>：<br>… | while <条件>：<br>…<br>while <条件>：<br>… |

图 3-6 循环嵌套的四种形式

循环嵌套的逻辑流程和代码与分支结构的嵌套类似，都可以多重嵌套，类型相同或不同的循环结构可以互相嵌套，但要注意各重循环不能逻辑交叉，所以在实际编程过程中一定要格外注意各重循环的冒号与缩进，保证逻辑清晰。

**【例 3-8】** 鸡兔同笼问题：鸡兔共有 35 只，脚 94 只，求鸡有多少只？兔子有多少只？

```
chicken = 0
while chicken <= 35:
    rabbit = 0
    while rabbit <= 35:
        if (2 * chicken + 4 * rabbit == 94) and (chicken + rabbit == 35):
            print("鸡有 %d 只,兔有 %d 只" %(chicken, rabbit))
        rabbit += 1
    chicken += 1
```

结果为：

```
鸡有 23 只,兔有 12 只
```

本例使用 while 循环嵌套完成求解，程序并不难理解，首先将鸡的数量设置为初值 0，执行 while 循环，条件是鸡的数量小于或等于 35 只，然后在循环体里设置兔子初值为 0，执行嵌套的 while 循环，条件是兔子的数量也小于或等于 35 只，判断鸡和兔子的脚之和如果等于 94，同时二者数量和为 35，就找到了答案，将 chicken 和 rabbit 两个变量的值输出即可，不满足条件则在内循环推进 rabbit 变量加 1，再次执行内循环，直到 rabbit 不满足小于或等于 35 的条件退出内循环，进入外层循环，执行后面的 chicken 变量加 1，如此往复直到找到最终结果。要注意的是，要严格执行语句缩进，否则容易出现死循环或错误。如果将上面的例子改用 for 循环嵌套来解决，代码如下，相对简洁，但其实二者的原理是一样的。

```
for chicken in range(1,35):
    for rabbit in range(1,35):
        if (2 * chicken + 4 * rabbit == 94) and (chicken + rabbit == 35):
            print("鸡有 %d 只,兔有 %d 只" %(chicken, rabbit))
```

### 3.2.4 循环控制语句

循环结构有两个辅助循环控制的语句，分别是 break 语句和 continue 语句。break 语句是用来跳出最内层循环(for 循环或者 while 循环)，执行该循环后续代码；continue 语句用来结束当前当次循环，即跳过循环体中尚未执行的语句，但不会跳出当前的循环。二者的区别在于 continue 语句只是结束本次循环，而不是终止整个循环的执行，而 break 语句则会结束整个当前的循环。

**【例 3-9】** 用户输入一个 1～100 的整数，根据这个整数，计算出 1 到这个整数间的偶数之和。

```
n = eval(input('请输入一个 1～100 的整数'))
sum = 0
for i in range(1,n+1):
    if (i % 2! = 0):
        continue
    sum += i
print(sum)
```

当输入 10 的时候，输出结果为 30，表示 1～10 中的偶数和为 30。这里的 continue 语句的作用是当循环变量 $i$ 为奇数的时候跳过本次循环，推进循环变量 $i$ 向前，当 $i$ 为偶数的时

候对 sum 变量做值的修改累加，最后得到 1～10 中的偶数和 30。

如果将例 3-9 中的 continue 语句换成 break 语句，那么上述程序无论用户输入 1～100 中的哪个整数，都会在刚一执行循环时就跳出，然后输出 sum 变量的值为 0，读者可自行尝试，领会两个语句的不同之处。

## 3.3 异常处理

### 3.3.1 异常的常见形式

异常通常是指代码运行的时候，因输入数据不合法或者某条件临时不满足发生的错误。异常常见的形式包括将 0 作为除数、变量不存在或拼写错误、打开一个不存在的文件、数据表操作 SQL 命令错误或访问的字段不存在、要求输入整数但经 input()函数输入的内容无法使用 int()函数转换为整数、要访问的属性不存在、文件传输过程中网络连接突然断开等。这些情况都会引发代码异常，代码一旦引发异常就会使程序出现问题，如果得不到正确的处理就会使整个程序中止运行。下面的代码在 IDLE 交互模式下演示了常见的异常表现形式。

```
>>> 5/0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
>>> with open(r'c:\a.txt',encoding = 'utf-8') as fp:
    content = fp.read
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    with open(r'c:\a.txt',encoding = 'utf-8') as fp:
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\a.txt'
>>> import pymysql
>>> conn = pymysql.connect(host = "localhost",user = "root",password = "wang",db = "test")
>>> cursor = conn.cursor()
>>> cursor.execute("select * from student limit 1")
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    cursor.execute("select * from student limit 1")
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
cursors.py", line 148, in execute
    result = self._query(query)
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
cursors.py", line 310, in _query
    conn.query(q)
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
connections.py", line 548, in query
    self._affected_rows = self._read_query_result(unbuffered = unbuffered)
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
connections.py", line 775, in _read_query_result
```

```
    result.read()
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
connections.py", line 1156, in read
    first_packet = self.connection._read_packet()
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
connections.py", line 725, in _read_packet
    packet.raise_for_error()
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
protocol.py", line 221, in raise_for_error
    err.raise_mysql_exception(self._data)
  File "C:\Users\dell\AppData\Local\Programs\Python\Python 37\lib\site-packages\pymysql\
err.py", line 143, in raise_mysql_exception
    raise errorclass(errno, errval)
pymysql.err.ProgrammingError: (1146, "Table 'test.student' doesn't exist")
>>> a = int(input('请输入一个正整数'))
请输入一个正整数 35,000
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    a = int(input('请输入一个正整数'))
ValueError: invalid literal for int() with base 10: '35,000'
>>> b = [2,4,6,8,10]
>>> b.rindex(3)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    b.rindex(3)
AttributeError: 'list' object has no attribute 'rindex'
```

上面的代码中加粗的内容就是提示语句发生的异常，当出现异常时不要惊慌失措，仔细阅读错误提示，通过提示发现问题的原因，之后尝试修改直至解决异常问题。细心的读者可能已经发现，异常提示往往在最后一行给出了错误的类型，如 ZeroDivisionError、NameError、FileNotFoundError、pymysql. err. ProgrammingError、ValueError、AttributeError 等，倒数第二行给出了导致错误的那一行代码。

### 3.3.2 异常处理结构语法

一个设计质量高的 Python 程序代码应该是充分考虑可能发生的错误并进行预防处理，可以给出相应提示信息或者忽略异常继续执行，代码足够健壮，这就要用到异常处理了。异常处理结构的一般思路是首先尝试运行代码，如果不出现异常就正常执行，如果发生异常就根据异常类型的不同采取应对的处理方案。异常处理结构语法格式如下。

```
try:
    #可能引发异常的代码块
except 异常类型 1 as 变量 1:
    #处理异常类型 1 的代码块
except 异常类型 2 as 变量 2:
    #处理异常类型 2 的代码块
…
[else:
    #如果 try 块中的代码没有引发异常,就执行这里的代码块
]
[finally:
    #不管 try 块中的代码是否引发异常,也不管异常是否被处理
    #总是最后执行这里的代码块
]
```

上面的语法格式里，else 和 finally 是可选项，根据程序需要决定是否书写在代码中，except 语句的数量也要根据具体业务逻辑确定。

**【例 3-10】** 要求用户输入一个正整数，求 1 到这个正整数的累加和，对用户输入进行有效性检查判断，如果输入的是正整数就继续完成功能，否则给出相应提示。

```
sum = 0
try:
    n = int(input('请输入一个正整数'))
    assert n > 0
except:
    print('您输入的数据有误,必须输入正整数')
else:
    for i in range(1,n + 1):
        sum + = i
print('1 到 % d 之间的数的累加和为 % d' % (n,sum))
```

结果为：

```
请输入一个正整数 10
1 到 10 之间的数的累加和为 55
```

再次运行，输入一个负整数则会给出错误提示。

```
请输入一个正整数 - 3
您输入的数据有误,必须输入正整数
```

## 3.4 综合例题

（1）编写程序判断“水仙花数”，若是水仙花数，则输出“Yes”，若不是则输出“No”。所谓水仙花数，是指 1 个 3 位的十进制数，其各位数字的立方和等于该数本身。例如，153 是水仙花数，因为 $153=1^3+5^3+3^3$。

```
def daffodil (n):
    if n > 99 and n < 1000:
        a = n//100
        b = n//10 % 10
        c = n % 10
        if n == a**3 + b**3 + c**3:
            return 'Yes'
        else:
            return 'No'
    else:
        print('please enter a three-digit number')
>>> daffodil (153)
'Yes'
>>> daffodil (995)
'No'
>>> daffodil (15)
please enter a three-digit number
```

上例通过定义一个带有参数 $n$ 的函数 daffodil()来判断 $n$ 是否为水仙花数，使用到了

多分支嵌套。关于函数的内容会在第 6 章中详细介绍。

(2) 编写程序,计算百钱买百鸡问题。假设公鸡 5 元一只,母鸡 3 元一只,小鸡 1 元三只,现在有 100 元钱,想买 100 只鸡,问有多少种买法?

```
for x in range(0,20):                          #20 表示公鸡最大购买数
    for y in range(0,33):                      #33 表示母鸡最大购买数
        z = 100 - x - y                        #z 表示除了公鸡和母鸡外的小鸡数量
        if z % 3 == 0 and 5 * x + 3 * y + z/3 == 100:
            print('公鸡: {}只,母鸡: {}只,小鸡: {}只.'.format(x,y,z))
```

结果为:

```
公鸡: 0 只,母鸡: 25 只,小鸡: 75 只.
公鸡: 4 只,母鸡: 18 只,小鸡: 78 只.
公鸡: 8 只,母鸡: 11 只,小鸡: 81 只.
公鸡: 12 只,母鸡: 4 只,小鸡: 84 只.
```

上面的例子是 for 循环嵌套,注意用到了 format 格式化输出。

(3) 输入一个正整数,统计该数的各位数字中零的个数,并求各位数字中的最大者。例如,输入 10032,则输出为:零的个数为 2,最大值为 3。

```
n = int(input('请输入一个正整数 n = '))
count = 0
imax = 0
while n:
    t = n % 10
    if t == 0:
        count + = 1
    if t > imax:
        imax = t
    n = n//10                                  #是求整商,需要注意
print("该数各位数字中零的个数为 %d,最大值为 %d" %(count,imax))
```

结果为:

```
请输入一个正整数 n = 10032
该数各位数字中零的个数为 2,最大值为 3
```

(4) 猜数字游戏:随机生成一个 1～100 的数字 randnum,用户猜测一个数字 guess,如果 guess > randnum,打印“太大了”,如果 guess < randnum,打印“太小了”,如果 guess 等于 randnum,打印“恭喜”,并且退出。用户总共有 5 次猜数字的机会,超过 5 次就打印“失败”,并且退出。

```
import randnum
for i in range(1,6):
    randnum = random.randint(1,100)     #randint()函数生成随机数字
    guess = int(input('共 5 次机会,请第 %i 次输入你所猜之数(1～100): ' % i))
    if guess > randnum:
        print('太大了')
    elif guess < randnum:
        print('太小了')
    else:
```

```
            print('恭喜')
            break
    print('失败')
```

## 习题

### 一、选择题

1. 下列属于非法 Python 语句的是(　　)。

A. a=b=c=1　　B. a=(b=c+1)　　C. a,b=b,a　　D. a+=b

2. 下列哪一个表达式的值是 True?(　　)

A. ('c','b')>('1','2')　　B. '123'>'abc'

C. 3>2>2　　D. False>True

3. 下面代码的输出结果是(　　)。

```
a = 1
for i in range(4,0, - 1):
     b = (a + 1) * 2
     a = b
print(b)
```

A. 23　　B. 46　　C. 92　　D. 106

### 二、操作题

输入一个由数字和字母组成的字符串,统计数字个数和字母个数。