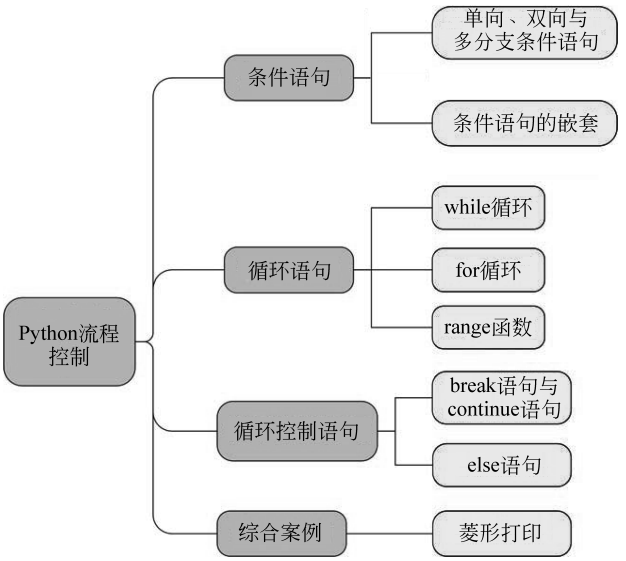
第3章

流 程 控 制

本章要点

- 条件结构
- 循环结构
- 循环控制语句
- 综合案例

本章知识结构图



本章示例

```
请输入菱形的行数10
          *
        * * *
      * * * * *
    * * * * * * *
  * * * * * * * * *
* * * * * * * * * * *
  * * * * * * * * *
    * * * * * * *
      * * * * *
        * * *
          *
```

第 2 章主要介绍了 Python 的基本语法规范，包括 Python 的基本数据类型和常用运算符等，并且通过程序演示了它们的用法，使用的演示程序都是从上往下顺序执行，其执行结果也是确定的。而现实生活中往往充满了复杂性和不确定性，例如 QQ 登录，如果用户身份验证通过，则可以进入主界面查看相关信息；如果验证不通过，则会执行其他的操作程序。也就是说，程序的执行结果是不确定的，程序会根据用户的输入情况，选择性地执行相应的程序。要想解决这类问题，读者需要学习流程控制模块。

任何一个复杂的系统都是由顺序结构、条件结构和循环结构这三种基本流程控制结构组成的，本章将集中进行讲解。Python 语言的语句默认是按照书写顺序依次执行的，这样的语句结构称为顺序结构。在顺序结构中，各语句按自上而下的顺序执行，语句执行间不作任何判断。有时则需要根据特定的情况，有选择地执行某些语句，这时就需要使用条件结构。还有一种情况，可以在给定条件下重复执行某些语句直到条件满足或者不满足，这些语句称为循环语句。

有了顺序、条件和循环这三种基本的结构，就可以在此基础上构建复杂的程序了。本章主要介绍 Python 语言的条件结构、循环结构和循环控制语句，以及其综合运用案例。

3.1　条件结构


视频讲解

在 Python 语言中主要有三类条件结构：单向 if 语句、双向 if-else 语句、多分支 if-elif-else 语句。

3.1.1　单向 if 语句

单向 if 语句只有 if 没有 else，只针对满足条件的情况做一些额外操作，如果不满足条件则什么都不做。单向 if 语句的执行流程如图 3-1 所示。

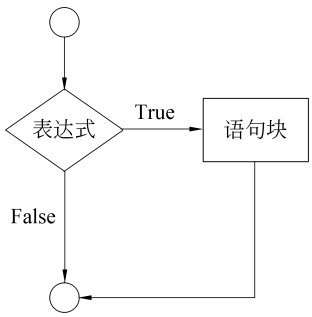


图 3-1　单向 if 语句执行流程图

单向 if 语句的语法形式如下。

```
if  布尔表达式 :
    语句块
```

当执行到 if 语句时，判断布尔表达式是否满足，若满足则执行语句块。

【示例 3.1】 使用单向 if 语句进行用户年龄判断。

代码如下：

```
1  age = int( input ("请输入你的年龄: ")) #获取用户输入的数字字符串,并转换为整数
2  if  age <0 or age >150:
3      print("输入不合法,采用默认值!")
4      age = 20
5  print(age)
```

程序运行结果：

```
请输入你的年龄:-5
输入不合法,采用默认值!
20
```

程序执行示例 3.1 的代码时，语句块只有当表达式的值为 True 时才会执行，否则，程序就会直接跳过这个语句块，执行紧跟在这个语句块之后的语句。

另外，在编写 if 语句块时，要严格执行缩进规则。这里的语句块可以包含多条语句，也可以只有一条语句。当语句块由多条语句组成时，要有统一的缩进形式，否则往往会出现逻辑错误，即语法检查没有错误，结果却非预期。

建议读者尝试调整缩进及改变 print(age)语句的位置，加深对程序的理解。

3.1.2 双向 if-else 语句

if-else 语句是一种双向结构，是对单向 if 语句的扩展，如果表达式结果为 True，则执行语句块 1，否则执行语句块 2。if-else 语句的执行流程如图 3-2 所示。

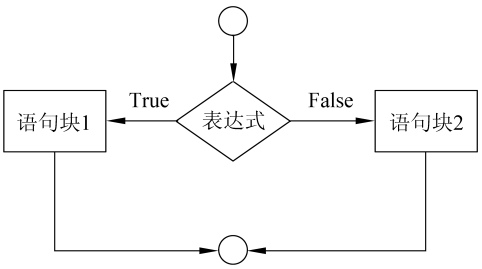


图 3-2 if-else 语句执行流程图

if-else 语句的语法形式如下。

```
if 布尔表达式:
    语句块 1
else:
    语句块 2
```

语法说明：当程序执行到 if 语句时，会判断布尔表达式是否为真，若为真，则执行语句块 1，否则执行语句块 2。

【示例 3.2】 使用双向 if-else 语句判断奇偶数。

代码如下：

```
1  num = int(input("请输入一个整数: "))
2  if  num %2 ==0:                          #对 num 做模 2 运算
3      print("这是一个偶数!")
4  else:
5      print("这是一个奇数!")
```

程序运行结果：

```
请输入一个整数:23
这是一个奇数!
```

注意：①else 语句块不能独立存在，需要和 if 语句块配合使用；②else 语句块的缩进与它所对应的 if 语句块缩进相同。

3.1.3　多分支 if-elif-else 语句

如果需要在多组操作中选择一组执行，就会用到多分支结构，即 if-elif-else 语句。该语句利用一系列布尔表达式进行检查，并在某个表达式为真的情况下执行相应的代码。if-elif-else 语句的备选操作较多，但是有且只有一组操作被执行。程序执行流程如图 3-3 所示。

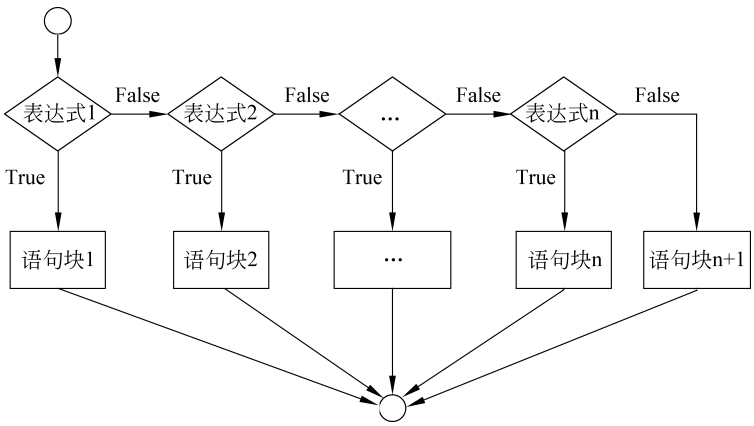


图 3-3　if-elif-else 语句执行流程图

多分支 if-elif-else 语句语法形式如下。

```
if 布尔表达式 1:
    语句块 1
elif 布尔表达式 2:
    语句块 2
⋮
elif 布尔表达式 n:
    语句块 n
else:
    语句块 n+1
```

语法说明：当执行到 if 语句时，从表达式 1 开始，依次判断表达式 1～n 是否为真，若为真，则执行表达式下的语句块，均不满足时则执行语句块 n＋1。

【示例 3.3】　使用多分支 if-elif-else 语句判断学生成绩等级。

代码如下：

```
score = float(input("请输入你的分数："))
if score >=90:
    grade = "优秀"
elif score >=80:
    grade = "良好"
elif score >=70:
    grade = "中等"
elif score >=60:
    grade = "及格"
else:
    grade = "不及格"
print(score, "对应的等级为:", grade)
```

程序运行结果：

```
请输入你的分数:65
65.0 对应的等级为：及格
```

也可以用多层次 if else 实现上述功能，但代码更烦琐。代码如下：

```
score = float(input("请输入你的分数："))
if score >=70:
      if score >=80:
          if score >=90:
              grade = "优秀"
          else:
              grade = "良好"
      else:
          grade = "中等"
else:
      if score >=60:
          grade = "及格"
      else:
          grade = "不及格"
print(score, "对应的等级为:", grade)
```

程序运行结果：

```
请输入你的分数:55
55.0 对应的等级为：不及格
```

以上两个程序虽然都可以判断 0～100 之间的分数对应等级，但对于负数和超过 100 的不合法输入却缺乏判断。因此可以将程序优化为条件语句嵌套。

代码如下：

```
score = float(input("请输入你的分数: "))
if score <0 or score >100:
  print("输入不合法,请重新输入!")
else:
  if score >=90:
      grade = "优秀"
  elif score >=80:
      grade = "良好"
  elif score >=70:
      grade = "中等"
  elif score >=60:
      grade = "及格"
  else:
      grade = "不及格"
  print(score, "对应的等级为:", grade)
```

程序运行结果：

```
请输入你的分数:70
70.0 对应的等级为: 中等
```

条件语句嵌套，即在条件语句的 if 或 else 语句块中还存在 if 判断，经过优化后程序将更加健壮。使用条件语句嵌套可以便于程序做进一步的判断。条件语句嵌套时，可以通过缩进查看条件语句的层次关系。理论上，嵌套的层次没有限制，但实际编程中，应尽可能避免三层以上的嵌套。

3.1.4　简化版 if 语句

条件判断的使用频率很高，为了简化条件判断语句的书写，Python 中提供了简化版的 if 语句，语法结构如下。

```
表达式 1  if  布尔表达式  else  表达式 2
```

语法说明：如果布尔表达式结果为 True，那么整个表达式的结果就是表达式 1；否则，表达式的结果就是表达式 2。例如，想将变量 number1 和 number2 中较大的值赋给 max，可以使用下面的条件表达式简洁地完成。

```
max = number1  if  number1 >number2  else  number2
```

【示例 3.4】 编写程序，提示用户输入两个数，打印出它们中较小的数。

代码如下：

```
num_1 = eval(input("请输入第一个数: "))
num_2 = eval(input("请输入第二个数: "))
min = num_1 if num_1 <num_2 else num_2      #返回 num_1,num_2 中的较小数

print("两个数中较小的为:", min)
```

程序运行结果：

```
请输入第一个数:12
请输入第二个数:24
两个数中较小的为: 12
```

思考与练习

3.1　判断题：在 Python 中，表达式 x ＞ y ＞＝ z 是合法的。

3.2　判断题：Python 通过缩进来判断语句块是否处于分支结构中。

3.3　分析下面的程序。如果输入的 score 为 90，输出 grade 是什么？程序是否符合逻辑，为什么？

```
if score >=60:
    grade = "及格"
elif score >=70:
    grade = "中等"
elif score >=80:
    grade = "良好"
elif score >=90:
    grade = "优秀"
```


视频讲解

3.2　循环结构

3.1 节中介绍了 Python 的条件结构，主要包括单向的 if 语句、双向的 if-else 语句、多分支 if-elif-else 语句以及条件结构的嵌套。条件语句嵌套，即在条件语句 if 或 else 中还可以包含 if 语句，需要注意区分多分支 if-elif-else 语句和条件语句嵌套，多分支 if-elif-else 语句是一种并列关系。

循环结构就是在一定条件下重复执行某些操作。Python 提供了两种类型的循环语句：while 条件式循环语句和 for 遍历式循环语句。

学习循环语句需要重点关注循环的开始和结束条件，尽量避免执行死循环。

3.2.1　while 循环

while 循环是在条件满足的情况下循环执行某段程序，重复处理某一任务。基本语法格式如下。

```
while 循环继续条件:
    循环体
```

在 while 循环中，程序先判断循环继续条件，条件满足则执行循环体，执行完循环体后，再继续判断循环继续条件，若满足条件则再执行循环体，依次往复，直到循环继续条件不满足，才跳出循环。通常来讲，需要在循环体中对循环继续条件进行修改，以避免程序进入死循环。

【示例 3.5】 求 1～100 之间所有整数之和。

代码如下：

```
1  index = 1                              #当前开始的数
2  sum = 0                                #初始 sum 的值为 0
3  while index <=100:                     #循环条件
4      sum = sum +index                   #sum 累加
5      index = index +1                   #改变循环条件的值
6  print(sum)
```

程序运行结果：

```
5050
```

在编写 while 循环语句时，要注意以下几点。

(1) 循环体可以是一个单一的语句或一组具有统一缩进的语句。

(2) 每个 while 循环都包含一个循环继续条件，即控制循环执行的布尔表达式。每次循环都要计算该布尔表达式的值，如果它的计算结果为真，则执行循环体；否则，终止整个循环并将程序控制权转移到 while 循环后的语句。

(3) while 循环是一种条件控制循环，它是根据一个条件的真假来控制程序执行的。

3.2.2　for 循环

for 循环是一种遍历型的循环，它会依次对某个序列中的全体元素进行遍历，遍历完所有元素之后便终止循环。for 循环常用于循环次数确定的场景。

for 循环的一般格式如下。

```
for 控制变量 in 可遍历序列:
    循环体
```

在 for 循环语句中，控制变量是一个临时变量，可遍历序列一般是一个列表或元组，

保存了多个元素。将序列中每个元素依次赋值给控制变量后，程序执行循环体，循环体中一般会对控制变量进行操作。但不建议在 for 循环中对序列进行修改，以免导致程序结果难以预测。

【示例 3.6】 求 1～100 之间所有整数之和。

代码如下：

```
1  sum = 0                                  #初始 sum 的值为 0
2  for index in range(1, 101):              #变量从 1 到 100 依次循环
3      sum = sum +index                     #循环累加 sum 的值
4  print(sum)
```

程序运行结果：

```
5050
```

注意：①可遍历序列中保存了多个元素，如列表、元组、字符串等。②可遍历序列被遍历处理，每次循环时，都会将控制变量设置为可遍历序列的当前元素，然后执行循环体。当可遍历序列中的元素被遍历一次后，即没有元素可供遍历时，退出循环。

在 Python 中，for 循环语句经常结合 range()函数一起使用，range()函数用于生成整数数字序列。

以下是 range()函数的函数说明。

```
range(start, stop[, step])
```

函数说明：

(1) start：计数从 start 开始，默认为 0；

(2) stop：计数到 stop 结束，但不包括 stop，该参数必填。如 range(a, b)函数返回连续整数 a，a+1，…，b−2，b−1 的序列；

(3) step：步长，表示每次变化的数，默认为 1，正数表示递增，负数表示递减；

(4) start、stop、step 只能为整数，不能为浮点数；

(5) 返回值为 range 类型，需通过循环遍历其元素或通过下标访问其元素。

【示例 3.7】 range()函数的运用。

```
1  a = range(1,100)
2  print(a)
```

程序运行结果：

```
range(1, 100)
```

当需要获取上述示例中 a 包含的值时，需使用列表转换函数 list()。

【示例 3.8】　使用 list()函数求 a 的值。

```
1   a = range(1,100)
2   print(list (a))
```

程序运行结果：

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

当 step 设置为负数时，start 设置的值要大于 stop。

【示例 3.9】　range()函数中 step 为负数的情况。

```
1   a =range(20, 1, -1)
2   print(list (a))
```

程序运行结果：

```
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
```

说明：

(1) range(a)等价于 range(0, a, 1)，即从 0 开始不包含 a，步长为 1。

(2) range(a, b)等价于 range(a, b, 1)，即从 a 开始不包含 b，步长为 1。

【示例 3.10】　求 1～100 之间所有偶数之和。

代码如下：

```
1   sum = 0
2   for i in range(0, 101, 2):                #从 0 开始 100 为止，步长为 2。
3       sum = sum +i
4   print(sum)
```

程序运行结果：

```
2550
```

在学习 while 语句和 for 语句后，不仅需要掌握它们的语法和使用规则，还要清楚它们的区别。

【示例 3.11】　使用 for 循环和 while 循环求和，了解它们之间的区别。

for 循环：

```
index = 1
sum = 0
while index <=100:
    sum = sum +index
    index = index +1
print(index)
print(sum)
```

程序运行结果：

```
101
5050
```

while 循环：

```
sum = 0
for index in range(1, 101):
    sum = sum +index
print(index)
print(sum)
```

程序运行结果：

```
100
5050
```

通过上述示例可以发现，while 循环执行时，要等控制变量的值变化以后再判断，不满足条件才退出循环。而 for 循环则是确定好了 range()函数变量的取值范围，均取完后再跳出循环。因此，while 循环次数的结果是打印 101，而 for 循环打印的是 100。

3.2.3 循环嵌套

Python 语言允许在一个循环体中嵌入另一个循环。如在 while 循环中可以再嵌入 while 循环或 for 循环；在 for 循环中也可以再嵌入 for 循环或 while 循环。一般建议循环嵌套不要超过三层，以保证程序的可读性。

【示例 3.12】 编写程序实现九九乘法表，效果如图 3-4 所示。

分析：九九乘法表是一行行打印，一共有 9 行，第 n 行有 n 个式子，每一行中式子的第二个操作数相同，第一个操作数从 1 开始不断递增，直到 n 为止。

代码如下：

```
for i in range(1, 10):                    #行数 i 为 1 到 9
    for j in range(1, i+1):               #对每行来说，列数 j 的最大值为行数 i
```

```
        print(str(j)+"*"+str(i)+"="+str(i*j), end="  ")
                                #将乘法运算式打印并通过结束符对齐,在每行结束时换行
    print("")
```

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

图 3-4　九九乘法表

程序运行结果：

```
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

示例中运用了双重循环。因为每行中有多列，并存在多行。第一个循环是行的循环，第二个循环是列的循环。

思考与练习

3.4　判断题：在 Python 中，表达式 28＞25＞＝2 是合法的，且结果为 True。

3.5　编写程序求 1～100 范围内的所有奇数之和。

3.6　编写程序打印数字金字塔。打印的行数由用户通过键盘输入，运行效果如图 3-5 所示。

```
请输入行数8
       1
      121
     12321
    1234321
   123454321
  12345654321
 1234567654321
123456787654321
```

图 3-5　数字金字塔效果

3.3　循环控制语句

视频讲解

3.2 节中介绍了 Python 语言的循环结构，主要包括 while 循环和 for 循环，以及循环嵌套，并通过九九乘法表示例演示了循环结构的嵌套使用。

本节中将继续介绍在循环结构中会经常使用到的循环控制语句。

3.3.1 循环控制语句

循环控制语句主要包括 break 语句和 continue 语句。

break 语句用于终止或中断整个循环语句，即使循环条件没有判断为 False 或者序列还没被完全遍历完，也会停止执行循环语句。如果使用嵌套循环，break 语句将跳出最深层的循环，并开始执行最深层循环语句的下一行代码。

continue 语句则是终止当次循环，忽略 continue 之后的语句，提前进入下一次循环。

【示例 3.13】 对比 break 语句和 continue 语句执行的不同结果。

break 语句代码如下：

```
1  sum = 0                                  #初始 sum 值为 0
2  for i in range(1, 10):                   #将 1 到 9 依次取出，赋值给临时变量 i
3    if i %3 ==0:                           #依次判断 i 是否能够整除 3
4      break                                #中断，退出 for 循环
5    sum = sum +i                           #求和
6  print(sum)
```

程序运行结果：

```
3
```

continue 语句代码如下：

```
1  sum = 0                                  #初始 sum 值为 0
2  for i in range(1, 10):                   #将 1 到 9 依次取出，赋值给临时变量 i
3    if i %3 ==0:                           #依次判断 i 是否能够整除 3
4      continue                             #满足条件则跳出当次循环
5    sum = sum +i                           #求和
6  print(sum)
```

程序运行结果：

```
27
```

注意：break 语句和 continue 语句均不能单独存在，需要配合循环语句使用。

3.3.2 循环中的 else 语句

和其他语言不同，Python 中的循环语句还可以带有 else 子句，else 子句在序列遍历结束(for 语句)或循环条件为假(while 语句)时执行，但循环被 break 终止时不执行。如图 3-6 所示。

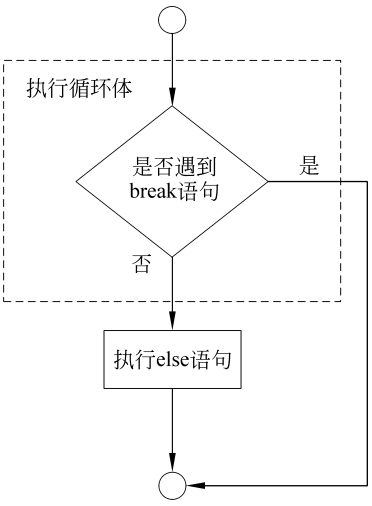


图 3-6　循环中的 else 语句执行流程图

带有 else 子句的 while 循环语句完整形式如下：

```
while 循环继续条件:
    循环体
else:
    语句块
```

带有 else 子句的 for 语句完整形式如下：

```
for 控制变量 in 可遍历序列:
    循环体
else:
    语句块
```

【示例 3.14】 在示例 3.13 中的两个示例中加上 else 语句。
break 语句代码如下：

```
1  sum = 0
2  for i in range(1, 10):
3    if i %3 ==0:
4      break
5    sum = sum +i
6  else:                                  #else 语句
7    print("i=",i)                        #如果执行 else 语句,就输出 i 的值
8  print(sum)
```

程序运行结果：

```
3
```

在 break 语句中，执行 break 语句后直接就跳出了循环，不再执行 else 语句，执行结果依然只打印 3。

continue 语句代码如下：

```
sum = 0                                  #初始和为 0
for i in range(1, 10):                   #将 1 到 9 依次取出
  if i %3 ==0:                           #依次判断是否能够整除 3
    continue                             #满足条件则跳过当次循环
  sum = sum +i                           #满足条件的值求和
else:                                    #else 语句
  print("i =",i)                         #如果执行 else 语句，就输出 i 的值
print(sum)
```

程序运行结果：

```
i = 9
27
```

执行 continue 语句时，只会跳过当次循环，会继续遍历其他元素，直至全部遍历结束，因此会执行 else 语句。所以 continue 语句会输出两行结果，分别为 i=9 和 27。

注意：else 语句在正常执行循环体结束后才会执行，遇到 break 语句时，会直接跳出循环，不再执行 else 语句。

思考与练习

3.7　判断题：所有的 for 循环语句都可以用 while 循环语句实现，所有的 while 循环语句也都可以用 for 循环语句实现。

3.8　判断题：for 循环能用来实现无限循环的编程。

3.9　判断题：死循环对于编程没有任何好处。

3.10　判断题：break 是用来跳出当前循环层的关键字。

视频讲解

3.4　综合案例

3.3 节介绍了 Python 中条件结构和循环结构的实现方式，并介绍了循环结构的控制语句。接下来通过一个综合案例演示多种结构同时使用的情况。

【示例 3.15】 编写程序实现图 3-7 和图 3-8 所示效果。程序首先提示用户输入菱形的行数，然后打印出上下对称的菱形形状。

提示：奇数行、偶数行效果会有不同，每个星号之间有一个空格。建议上下分开打印。

图 3-7　10 行菱形效果

图 3-8　9 行菱形效果

参考代码如下：

```
1   rows = int ( input('请输入菱形的行数'))  #将用户输入的数字字符串转换成整数
2   half = rows // 2                  #整除,分为上下两部分
3   if  rows %2 ==0:                  #进行奇偶判断
4       up = half                     #当为偶数时,最大行数就为输入整数的一半
5   else:
6       up = half +1                  #为奇数时,上部分显示的结果比下部分多一行
7
8   for i in range(1,up+1):           #从第一行到最大行数依次遍历
9       print('  ' * (up - i), "* " * (2 * i -1))
10
11  for i in range(half, 0, -1):      #反向遍历
12      print('  ' * (up - i), "* " * (2 * i -1))  #打印下半部分的结果
```

程序运行结果：

```
请输入菱形的行数 7
        *
      * * *
    * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

3.5　本章小结


视频讲解

本章主要讲解了 Python 程序的流程控制，从程序的三种基本结构入手，即顺序结构、条件结构和循环结构。顺序结构最为简单，应用也最为广泛，指程序的执行是按照语句书写的顺序依次执行，程序结果往往也是确定的。

条件结构也称分支结构或选择结构，是指程序执行到某个阶段时，程序会根据实际情况选择性地执行某些语句，有些语句是不执行的。条件语句部分主要介绍了：单向 if 语句，该语句先要判断是否符合某条件，符合则执行操作，不符合则什么都不执行；双向 if-else 语句，该语句执行时，先对布尔表达式进行判断，当符合条件时做什么，不符合时做什么，往往只适合于非此即彼的情况；多分支 if-elif-else 语句主要用于有多个选择时，但也只执行互斥条件中的某一个。条件语句可以嵌套，即在 if、else 语句中再嵌入 if、else 语句，当满足条件时进一步判断是否满足其他条件，需要注意其和多分支条件语句的区别。最后还简单地介绍了简化版的 if 语句，它是双分支语句的简化。

在循环结构部分，讲解了条件式的 while 循环和遍历式的 for 循环。while 循环是一直判断是否满足条件，满足则执行循环体，不满足则跳出循环。而 for 循环是一种遍历式的循环，是将元组或列表中的元素依次取出，然后执行循环体，所有元素均取完后则结束循环。通常使用 for 循环时，会配合 range()函数使用。range()函数主要用于生成整数序列，其中包含三个参数：从哪开始，到哪结束，步长为多少。range()函数默认从 0 开始，不包含最后一个数，默认步长为 1。

循环控制语句主要包括 break 语句和 continue 语句。遇到 break 语句不管是否满足循环条件，都会直接退出。continue 语句表示继续，跳过当次循环，继续下次循环，循环并没有结束，只是跳到下一个循环。和其他语言不同的是，Python 在循环语句中还可以加入 else 语句。else 语句是循环正常结束后执行的语句，而 break 语句是让循环非正常结束。

最后通过综合案例——菱形打印，学习了综合使用条件语句和循环语句。菱形打印的关键点在于打印的行数由用户输入，奇偶行数打印的形状有偏差，从而需要先进行奇偶判断，而且在打印时，不能单独每行打印，需要发现规律进而书写循环语句。

课后练习

3.1　阅读下列程序代码片段，写出代码片段的执行结果。

代码片段①

```
sum = 0
for i in range(10):
    if i %4 ==0:
        break
    sum +=i

print(sum)
```

代码片段②

```
sum = 0
for i in range(10):
```

```
    if i // 4 ==2:
        continue
    sum +=i

print(sum)
```

代码片段③

```
sum = 0
i = 0
while i <10:
    if i %4 ==0:
        continue
    sum +=i
    i +=1

print(sum)
```

代码片段④

```
i = 1
while i <5:
    i +=1
else:
    i *=2

print(i)
```

3.2　编写程序，判断某一年份是闰年还是平年。年份由用户通过键盘输入，运行效果如图 3-9 所示(闰年的标准：能被 4 整除，但是不能被 100 整除；或者能被 400 整除。其他都为平年)。

```
请输入年份：2019        请输入年份：2020        请输入年份：2100
2019 是平年!            2020 是闰年!            2100 是平年!
```

图 3-9　练习 3.2 的程序执行结果

3.3　编写程序，求一个自然数除了自身以外的最大约数。自然数由用户通过键盘输入，运行效果如图 3-10 所示。

```
请输入自然数49
49最大的约数为：7
```

图 3-10　练习 3.3 的程序执行结果

3.4　编写程序对整数进行质因数分解，并输出结果。整数由用户通过键盘输入，运行效果如图 3-11 所示。

```
请输入一个整数：90
90 = 2 * 3 * 3 * 5
请输入一个整数：120
120 = 2 * 2 * 2 * 3 * 5
```

图 3-11　练习 3.4 的程序执行结果

3.5　查阅资料，理解并熟悉 Python 编程之禅，并将其运用在后面的课程学习中。

课后习题讲解（一）

课后习题讲解（二）

第4章 常用数据结构

本章要点

- 列表
- 元组
- 字符串
- 集合
- 字典

本章知识结构图