第3章 流程控制

任何一门编程语言编写的程序都是由顺序结构、分支结构和循环结构这三种基本流程控制结构组成。Python 代码默认按书写顺序从前往后依次执行，这样的语句执行结构被称为顺序结构。在顺序结构中，语句代码顺序执行，不作任何条件判断。但有些情况下，却需要有选择地执行某些语句，这就需要使用条件结构；而有时则需要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，这就要用到循环结构语句。

有了顺序、选择和循环这三种基本结构，就可以在这基础上构建任意复杂的程序代码。本章将介绍 Python 语言的条件结构、循环结构和循环控制语句。

3.1 条件结构


视频讲解

Python 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 子句，满足条件时将执行指定操作，不满足条件则什么也不做。单向 if 语句执行流程如图 3.1 所示。

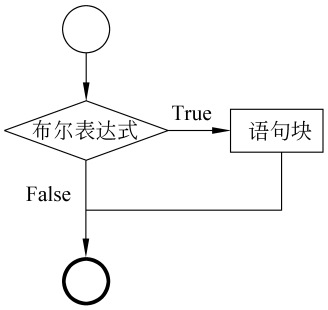

图 3.1 单向 if 语句执行流程图

单向 if 语句的语法结构如下。

```
if 布尔表达式:
    语句块
```

语法释义：当执行到 if 语句时，首先判断布尔表达式是否满足，满足则执行语句块，否则执行 if 语句块后的内容。

【示例 3.1】 单向 if 语句示例。

```
age = int(input("请输入您的年龄:"))          #获取输入的年龄,并转换为整数
if age < 0 or age >= 120:
    print("年龄不真实,将采用默认值。")
    age = 20
print("您的年龄为:", age)
```

程序运行结果:

```
请输入您的年龄: 132
年龄不真实,将采用默认值。
您的年龄为: 20
```

if 语句块可以包含多条语句,也可以只有一条语句。当 if 语句块由多条语句组成时,要有统一的缩进形式,否则将可能会出现逻辑错误,即语法检查没错,结果却非预期。

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。

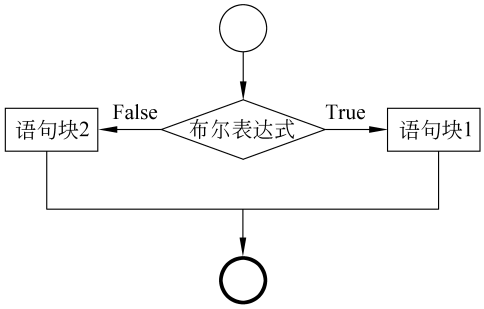

图 3.2 if-else 语句的执行流程图

if-else 语句的语法结构如下。

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

语法释义:当程序执行到 if 语句时,会判断布尔表达式结果是否为真,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。

```
num = int(input("请输入一个整数: "))
if num % 2 == 0:                    #对 num 做模 2 运算
    print("这是一个偶数!")          #布尔表达式结果为真,则执行该代码
else:
    print("这是一个奇数!")          #结果为假,则执行该行代码
```

程序运行结果：

```
请输入一个整数:23
这是一个奇数!
```

注意：①else 语句不能独立存在，需要和 if 语句配合使用；②else 语句块的缩进必须与它所对应的 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操作执行，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语句块。if-elif-else 语句的备选操作较多，但是有且只有一组操作会被执行。程序执行流程如图 3.3 所示。

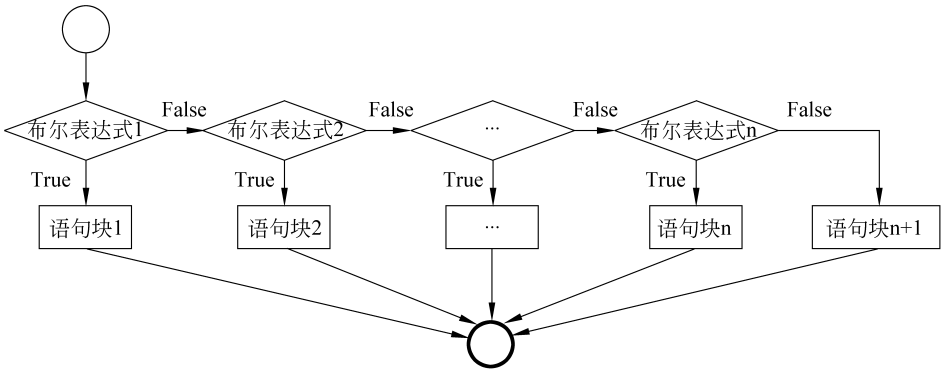

图 3.3　if-elif-else 语句流程执行图

多分支 if-elif-else 语句语法结构如下。

```
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
```

语法释义：当执行到 if 语句时，从布尔表达式 1 开始依次判断表达式结果是否为真，真则执行当前表达式下的语句块，表达式 1～n 结果均为假时则执行语句块 n＋1。

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

```
1    score = float(input("请输入您的分数:"))
2    if score >= 90:
3        grade = "优秀"
4    elif score >= 80:
```

```
    grade = "良好"
elif score >= 70:
    grade = "中等"
elif score >= 60:
    grade = "及格"
else:
    grade = "不及格"
print("您的成绩等级为:",grade)
```

程序运行结果:

```
请输入您的分数: 65
您的成绩等级为: 及格
```

以上的布尔表达式判断其实存在逻辑包含关系,如 score>=60,逻辑上就包含了 score>=90。因此,在编写多分支 if-elif-else 语句的布尔表达式时,如果存在逻辑包含关系,应将范围小的布尔表达式放在前面,范围大的布尔表达式放在后面,否则将产生逻辑错误。读者可以调换该程序的布尔表达式先后顺序来加深理解这一思想。

需要说明的是,if-elif-else 语句的功能完全可以使用多层嵌套的 if-else 语句等价实现,但代码会更加烦琐,因此并不建议这样做。

【示例 3.4】 使用多层嵌套的 if-else 语句判断学生成绩等级。

```
score = float(input("请输入您的分数: "))
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("您的成绩等级为: ", grade)
```

程序运行结果:

```
请输入您的分数: 65
您的成绩等级为: 及格
```

通常来讲,不建议 if-else 语句的嵌套超过 3 层,否则会影响程序的可读性。

另外,Python 并不要求 if-elif-else 结构后面必须有 else 代码块,有时省略 else 语句反而逻辑会更清晰,代码更加安全。这是因为 else 是一条相对不太安全的语句,只要不满足 if 或 elif 中的条件测试,其代码就会执行,这就为恶意的代码植入留下机会。

【示例 3.5】　使用多层嵌套的 if-elif 语句判断学生成绩等级。

```
score = float(input("请输入您的分数:"))
grade = "尚不存在"                    #先将 grade 定义为"尚不存在"
if score >= 90 and score <= 100#为了使语句更加安全,该表达式也做了修改
    grade = "优秀"
elif score >= 80 and score < 90:
    grade = "良好"
elif score >= 70 and score < 80:
    grade = "中等"
elif score >= 60 and score < 70:
    grade = "及格"
elif score >= 0 and score < 60: #改为 elif,使所有的用户输入,都要接受判断
    grade = "不及格"
print("您的成绩等级为:",grade)
```

程序运行结果：

```
请输入您的分数: -20
您的成绩等级为: 尚不存在
```

通过将 if-elif-else 结构转化为纯粹的 if-elif 结构，使得每个代码块都需要进行判断才能执行，从而使程序变得更加安全可控。

3.1.4　简化版的 if 语句


视频讲解

条件判断语句的使用频率很高，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。其语法结构如下。

```
表达式 1  if  布尔表达式  else  表达式 2
```

语法释义：如果布尔表达式结果为 True，那么整个语句的返回结果就是表达式 1 的计算结果；否则，将返回表达式 2 的计算结果。

【示例 3.6】　编写程序，输出两个数中的最小数。

```
num1 = eval(input("请输入 num1:"))
num2 = eval(input("请输入 num2:"))
smaller = num1 if num1 < num2 else num2          #返回两个数中的最小数
print("较小的数为:", smaller)
```

程序运行结果：

```
请输入 num1:12
请输入 num2:24
较小的数为: 12
```

思考与练习

3.1　判断题：表达式 x>y>=z 是合法的。

3.2　判断题：Python 通过缩进来判断代码块是否处于分支结构中。

3.3　编写代码，使用简化版的 if 语句，获得 3 个数中的最小值。

3.4　请分析下面的程序。如果输入 score 为 90，输出 grade 为多少？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

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，尽量避免程序进入死循环。

3.2.1　while 语句

while 语句在条件满足的情况下，重复执行 while 循环体，直到循环持续条件不满足为止。其语法结构如下。

```
while 循环持续条件:
    循环体
```

语法释义：程序先判断循环持续条件，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，条件满足则再执行循环体，依次往复，直到循环继续条件不满足，才跳出循环。

为了避免程序进入死循环，在设计 while 循环时，通常都会在循环体中对循环持续条件进行修改。

【示例 3.7】　求 1～100 之间所有整数之和。

```
index = 1
total = 0                          #设置初始 total 值为 0
while index <= 100:                #设置循环持续条件
    total += index                 #total 累加
    index += 1                     #改变循环条件的值
print("1 到 100 总和为:", total)
```

程序运行结果：

```
1 到 100 总和为: 5050
```

在编写 while 语句时，以下几点需要注意。

（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持续条件，即控制循环持续执行的布尔表达式。每次循环都要计算该布尔表达式的值，如果计算结果为真，则执行循环体；否则，Python 解释器将终止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，它根据循环持续条件的真假来控制程序的执行。

3.2.2　for 循环

for 循环是一种遍历式循环，它依次对某个序列中的全体元素进行遍历，遍历完所有元素后便终止循环。for 循环常用于循环次数确定的场景。

for 循环的语法结构如下。

```
for 控制变量 in 可遍历序列：
    循环体
```

语法释义：在 for 循环语句中，控制变量是一个临时变量，可遍历序列可以是一个列表、元组、字符串、字典等序列或可迭代对象，其中保存了多个元素，for 循环语句将序列中的元素依次取出，赋值给控制变量后，程序执行循环体，再从可遍历序列中取下一个元素。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，程序退出循环。

【示例 3.8】　求 1～100 之间所有整数之和。

```
1    total = 0                                  #初始 total 的值为 0
2    for index in range(1, 101):                #index 从 1~100 依次取值
3        total += index                         #循环累加 total 的值
4    print("index 最后值为:", index)
5    print("1 到 100 总和为:", total)
```

程序运行结果：

```
index 最后值为: 100
1 到 100 总和为: 5050
```

通常来讲，能用 for 循环实现的程序，也可以用 while 循环来实现，但一般 for 循环的效率更高。

【示例 3.9】　使用 while 循环实现示例 3.8。

```
1    total = 0
2    index = 1
3    while index <= 100:     #进行 while 循环
4        total += index
5        index += 1
6    print("index 最后值为:", index)
7    print("1 到 100 总和为:", total)
```

程序运行结果：

```
index 最后值为: 101
1 到 100 总和为: 5050
```

通过对比可以发现,while 循环执行时,要等控制变量的值变化以后,再判断循环持续条件,不满足则退出循环。而 for 循环则是先确定好 range()函数的取值范围,取数完毕后再跳出循环。因此 while 循环的结果是打印 101,而 for 循环打印的是 100。


视频讲解

3.2.3 range()函数

在 Python 程序中,for 循环和 while 循环经常结合 range()函数一起使用。

range()函数用于生成整数数字序列。其语法格式如下。

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

函数说明如下。

(1) start:计数从 start 开始,默认为 0。

(2) stop:计数到 stop 前 1 位整数结束,不包括 stop,该参数必填。如 range(a,b)函数将返回连续整数 a、a+1…b-2 和 b-1 的序列。

(3) step:步长,表示每次递增或递减的数量,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()对象,可通过循环遍历其元素或通过下标访问其元素。

【示例 3.10】 range()函数的运用。

```
1    a = range(1,11)          #生成 1 个 range()对象
2    print(a)
```

程序运行结果:

```
range(1, 11)
```

可通过索引来获得 range()对象中的内容,也可以使用 list()函数将其转换为列表,更加方便地提取其中的元素。

【示例 3.11】 获得 range()对象中的元素。

```
1    a = range(1,11)          #等价于 a = range(1, 11, 1)
2    print(a[0])              #打印 a 的索引 0 元素
3    b = list(a)              #使用 list()函数将 range 对象 a 转换为列表
4    print(b)
5    print(b[1])
```

程序运行结果:

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

range()函数的步长设置为负数时，表示每次递减，这时 start 参数要大于 stop 参数，否则无法取数。另外，步长也可以设置为其他整数。

【示例 3.12】　步长 step 设置示例。

```
a = range(10, 1, -1)#步长为-1,从 10 取数,到 2 结束,不包含 1
print(list(a))
b = range(1, 10, 2)     #步长为 2
print(list(b))
c = range(6)     #未设置 start 值,默认取 0,步长默认取 1
print(list(c))
```

程序运行结果：

```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
[1, 3, 5, 7, 9]
[0, 1, 2, 3, 4, 5]
```

3.2.4　循环嵌套

在处理一些较为复杂的问题时，可能会用到循环的嵌套。如在 while 循环中可以再嵌入 while 循环或 for 循环，或在 for 循环中再嵌入 for 循环或 while 循环。一般建议循环嵌套层次不要超过 3 层，以保证程序的可读性。

【示例 3.13】　编写程序实现九九乘法表，效果如图 3.4 所示。

```
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

分析：九九乘法表是一行行打印，共有 9 行，第 n 行有 n 个式子，每一行的式子打印时，第 2 个乘数不变，第 1 个乘数从 1 开始不断递增，直到 n 为止。

```
for i in range(1, 10):                    #行数 i 从 1 到 9
    for j in range(1, i+1):               #每行列数 j 的最大值为行数 i
        print(str(j)+"*"+str(i)+"="+str(i*j), end=" ")
    print("")                            #每行结束时换行
```

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
```

示例中运用到了双重循环。由于每行中有多列,并存在多行,因此第一个循环是行的循环,第二个循环是列的循环。

3.2.5 在循环中修改列表*

在 for 循环体中一般会对控制变量进行操作,但一般不建议在 for 循环中对控制变量的取值列表进行修改,以免导致程序结果难以预测。

【示例 3.14】 在 for 循环中修改列表。

```
1    ls = [0, 1, 0, 1, 0, 1, 0]                #设置列表内容
2    for i in ls:                              #使用 for 循环修改控制变量的取值列表
3        if i == 0:
4            ls.remove(0)
5    print(ls)                                 #打印取值后的列表结果
```

程序运行结果:

```
[1, 1, 1]
```

【示例 3.15】 在 for 循环中修改列表。

```
1    ls = [0, 1, 0, 1, 0, 1, 0, 0, 0, 0]      #设置列表内容,列表后面多加了 3 个 0
2    for i in ls:                              #使用 for 循环修改控制变量的取值列表
3        if i == 0:
4            ls.remove(0)
5    print(ls)                                 #打印取值后的列表结果
```

程序运行结果:

```
[1, 1, 1, 0, 0]
```

这时,程序的运行结果与读者的预期可能不太一致,没有达到删除列表中 0 元素的效果。这是 Python 的 for 循环所存在的问题,但如果使用 while 循环来处理,则不存在这样的问题。

【示例 3.16】 使用 while 循环修改列表。

```
1    ls = [0, 1, 0, 1, 0, 1, 0, 0, 0, 0]      #设置列表内容
2    while 0 in ls:                            #使用 while 循环修改控制变量的取值列表
3        ls.remove(0)
4    print(ls)                                 #打印取值后的列表结果
```

程序运行结果:

```
[1, 1, 1]
```

可以看到，使用 while 循环对控制变量的取值列表进行修改，没有出现问题，且代码更加简洁。

如果必须要使用 for 循环，对控制变量的取值列表进行修改，可以考虑从后往前删除，这个请读者自行尝试。

思考与练习

3.5　判断题：range(start，stop，step)函数的参数 step 不可以为负数。

3.6　判断题：一般来讲，循环嵌套最好不要超过 3 层，否则会影响程序的可读性。

3.7　编写程序求 1～100 范围内的所有奇数之和。

3.8　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

在执行循环过程中，如果要跳过某次循环，或者强制跳出整个循环，就需要用到循环控制语句。

3.3.1　循环控制语句


视频讲解

循环控制语句主要包括 break 和 continue 语句。

break 语句用于终止当前循环层语句，即使循环持续条件为 True 或者序列还没遍历结束，也会停止执行循环语句。如果是循环嵌套，break 语句将跳出当前层次循环，并开始执行当前层次循环外的循环语句的下一行代码。

【示例 3.17】　在循环中使用 break 语句。

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

程序运行结果：

```
i = 3
total = 3
```

而 continue 语句则用于终止当次循环，忽略 continue 之后的语句，提前进入下一次循环过程。

【示例 3.18】 在循环中使用 continue 语句。

```
total = 0                    #初始 total 值为 0
for i in range(1, 10):       #将 1~9 依次取出,赋值给临时变量 i
    if i % 3 == 0:           #依次判断 i 是否能够整除 3
        continue             #结束当次循环
        total += i           #求和
print("i =", i)
print("total =", total)
```

程序运行结果:

```
i = 9
total = 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。


视频讲解

3.3.2 循环中的 else 语句*

和许多编程语言不同,Python 的循环语句还可以带有 else 子句,else 子句在序列遍历结束后(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。

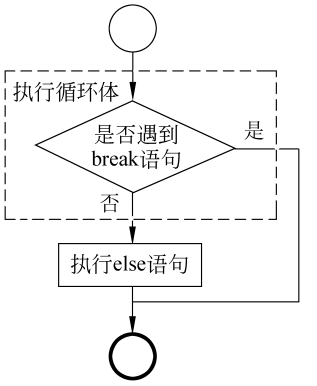

图 3.6 带 else 子句的循环语句执行流程

带有 else 子句的 while 循环语句的语法结构如下。

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

带有 else 子句的 for 语句的语法结构如下。

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

【示例 3.19】　在示例 3.17 中加上 else 子句。

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

程序运行结果：

```
total = 3
```

循环语句在执行 break 语句后，直接就跳出了循环，不再执行 else 子句，程序执行结果最后会输出 total=3。

【示例 3.20】　在示例 3.18 中加上 else 子句。

```
total = 0                    #初始 total 值为 0
for i in range(1, 10):       #将 1~9 依次取出,赋值给临时变量 i
    if i % 3 == 0:           #依次判断 i 是否能够整除 3
        continue             #中断,退出 for 循环
    total += i               #求和
    else:
    print("i =", i)
print("total =", total)
```

程序运行结果：

```
i = 9
total = 27
```

循环语句在执行 continue 语句后，会跳过当次循环，然后继续遍历序列，直至遍历结束，因此会执行 else 语句。该程序会输出两行结果，分别为 i=9 和 total=27。

对于具有 else 子句的 while 循环语句，也和前两例的 for 循环类似，只有在正常执行循环结束后才会执行 else 子句。如果遇到 break 语句时，则会直接跳出循环，不再执行 else 子句。

思考与练习

3.9　判断题：一般来讲，for 循环都可以用 while 循环实现，反之亦然。

3.10　判断题：break 关键字用来跳出当前循环层。

3.11　对于示例 3.15，使用 for 循环来实现删除列表 ls 中的 0 元素(进阶)。

3.12　编写代码，使用 while 循环对 1～100 中的偶数求和，并使用 else 子句打印最后的求和结果。

3.4 应用案例

接下来通过一个综合小例子演示多种控制结构同时使用的情况。

【示例 3.21】 根据用户输入的行数,打印出图 3.7 和图 3.8 所示的菱形图案。

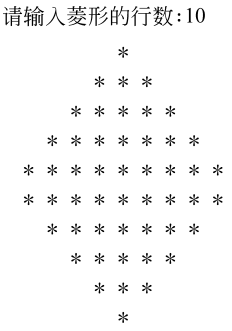

图 3.7 十行菱形效果图

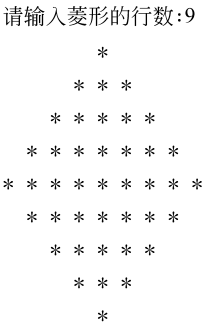

图 3.8 九行菱形效果图

程序分析:用户输入的行数为奇数时,中间最长的星号序列只有 1 行,而用户输入为偶数时,中间最长的星号序列则有 2 行。每个星号之间有 1 个空格,整个图形应上下分开打印,参考代码如下。

```
1    rows = int (input('请输入菱形的行数:')) #将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                           #row 为偶数时,行数为输入整数的一半
5    else:
6        up = half +1                        #row 为奇数时,上部分应比下部分多一行
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 Python 程序的流程控制，包括顺序结构、选择结构和循环结构。

选择结构是程序执行到某个阶段时，会根据实际情况有选择性地执行某些语句。Python 的选择结构主要有单分支 if 语句、双分支 if-else 语句和多分支 if-elif-else 语句。由于 if 语句在编程中经常使用，Python 还提供了简化版的 if 语句，它是双分支语句的简化。

循环结构包含条件式 while 循环和遍历式 for 循环。while 循环会判断是否满足循环持续条件，满足则执行循环体，不满足则跳出循环。而 for 循环则是一种遍历式的循环，是将序列中的元素依次取出，然后执行循环体，所有元素均取完后则结束循环。

循环语句通常会配合 range()函数使用。range()函数主要用于生成左闭右开的整数序列。

循环控制语句主要包括 break 语句和 continue 语句。break 语句用于跳出当前循环层；continue 语句则是跳过当次循环，继续下次循环，循环并没有结束。另外，Python 的循环语句还可以加入 else 子句，else 子句是循环正常结束后才会执行的语句。

本章最后通过一个综合小案例，综合使用了条件语句和循环语句。

课后习题

一、单选题

1. 以下表达式中，等价于 False 的是(　　)。

A. 1+2　　B. []　　C. 0+1　　D. '0'

2. 以下表达式中，等价于 True 的是(　　)。

A. 0　　B. []　　C. ()　　D. '0'

3. continue 关键词的作用是(　　)。

A. 跳出当前循环

B. 忽略后面代码，提前进入下一次循环

C. 继续一次当前循环

D. 以上都不对

4. 以下对循环体中的 else 子句，表述正确的是(　　)。

A. 不管循环是否正常结束，else 子句都会执行

B. 循环中如果使用了 continue，则 else 子句一定会执行

C. 只有循环正常结束，else 子句才会执行

D. 以上都不对

二、填空题

1. 以下 for 循环执行的结果是_____。

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

print(sum)
```

2. 以下 for 循环执行的结果是_____。

```
sum = 0
for i in range(10):
    if i // 4 == 2:
        continue
    sum += i

print(sum)
```

3. 以下 while 循环执行的执行结果为_____。

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

print(sum)
```

4. 以下 while 循环执行的执行结果为_____。

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

print(i)
```

三、编程题

1. 编写程序,判断用户输入的年份是闰年还是平年(闰年的标准为能被 4 整除但不能被 100 整除,或者能被 400 整除。其他年份都为平年)。

2. 编写程序,求一个自然数除了自身以外的最大约数。程序运行效果如图 3.9 所示。

3. 编写程序对整数进行质因数分解,并输出结果。程序运行效果如图 3.10 所示。

```
请输入一个整数：49
49 的最大约数为: 7
```

图 3.9　求一个整数的最大约数

```
请输入一个整数：90
90 = 2 * 3 * 3 * 5
```

图 3.10　对整数进行质因数分解

4. 编写代码,分别使用 for 循环和 while 循环求 1～100 内所有奇数的和。