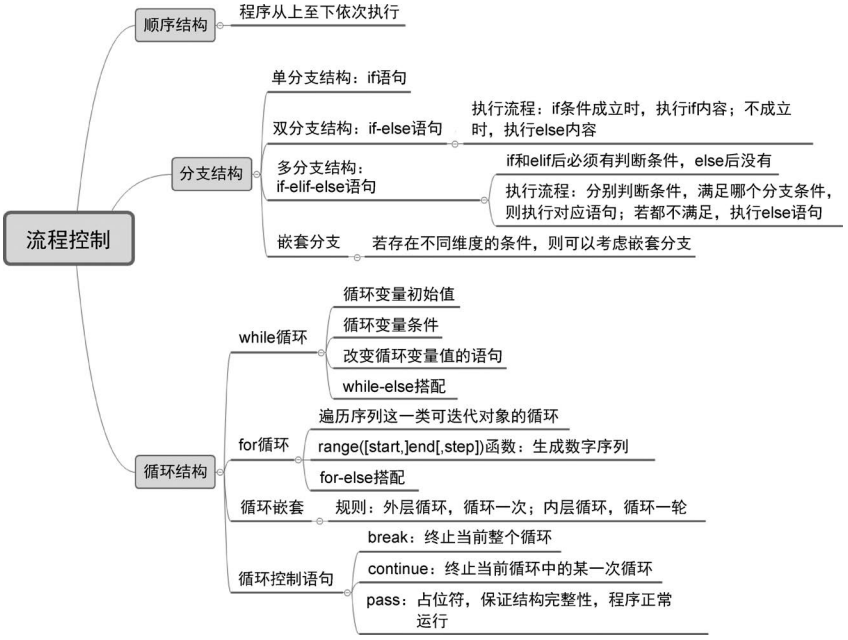
# 第 3 章

案例导读

# 流程控制语句

**学习目标**

- 理解程序的表示方法——程序流程图。
- 理解程序三大基本结构：顺序结构、分支结构和循环结构。
- 掌握 if、if-else 及 if-elif-else 语句的基本结构和语法。
- 掌握 for、while 循环的基本结构和语法。
- 掌握分支结构的嵌套、循环结构的嵌套，以及分支与循环的组合嵌套。
- 掌握 break、continue 及 pass 循环控制语句。




控制语句是程序语言的逻辑结构基础，也是程序编写的重点。掌握 Python 流程控制语句的应用，是进行程序和算法编写的必备条件。本章主要介绍 Python 的三大流程控制结构，顺序结构、分支结构和循环结构，使用这些流程控制语句可以控制程序的执行过程。其中，需要重点掌握 if、if-else、if-elif-else 三种分支结构以及 while、for 两种循环结构。

# 3.1　程序表示方法

## 3.1.1　程序流程图

程序流程图是用代表各种不同操作的框图进行组合，演示算法流程的一种表现形式，更加直观形象，便于理解。表 3-1 是一些常用的程序流程图符号。

**表 3-1　常用的程序流程图符号**

| 符　　号 | 描　　述 | 符　　号 | 描　　述 |
|---|---|---|---|
|  | 起止框 |  | 处理框 |
|  | 输入输出框 |  | 流程线 |
|  | 判断框 | ○ | 连接点 |

流程图是表示算法的较好的工具。一个流程图包括以下几部分。

(1) 表示相应操作的框。

(2) 带箭头的流程线。

(3) 框内外必要的文字说明。

**注意**：流程线不要忘记画箭头，它是用来反映流程先后逻辑的，若不画出箭头就难以判定各框的执行次序。用流程图表示算法直观形象，可以比较清楚地显示出各个框之间的逻辑关系。

## 3.1.2　基本结构流程图

Python 中主要有三种基本结构：顺序结构、分支结构和循环结构。用这三种基本结构可以实现各种复杂算法的程序逻辑。

### 1. 顺序结构

顺序结构是最简单的一种基本结构，按照代码的顺序从上到下依次执行。顺序结构的流程图如图 3-1 所示。

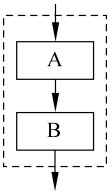



**图 3-1　顺序结构流程图**

### 2. 分支结构

Python 分支结构也称为选择结构，就是让程序“拐弯”，有选择性地执行代码。换句话说，可以跳过没用的代码，只执行有用的代码。选择结构的流程图如图 3-2 所示，如果条件 P 成立，则执行 A 操作，否则执行 B 操作，然后程序继续往后执行。

### 3. 循环结构

Python 循环结构又称为重复结构，就是不断地重复执行某一段代码。循环结构的流程图如图 3-3 所示。程序会首先判断条件 P 是否成立，如果成立则会执行 A 操作，然后再判断条件 P，直到条件 P 不成立，循环结构终止，程序继续往后执行。

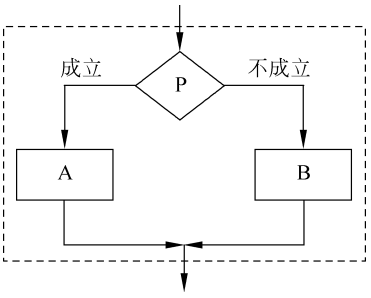



图 3-2　选择结构流程图

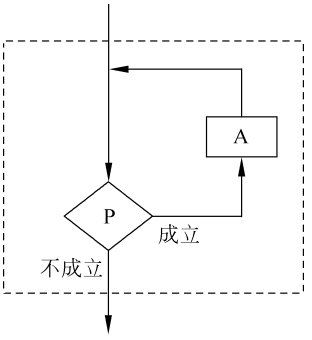



图 3-3　循环结构流程图

### 3.1.3　流程图的应用

案例：现需要判断数字序列 1～100 中，哪些数字为奇数，哪些数字为偶数，那么对应的程序思路如下。

（1）运用循环结构循环生成 1～100 的数字。

（2）再依次运用分支结构对数字奇、偶性进行判断。

（3）按照判断结果进行输出，直到循环结束。

对应的程序流程图如图 3-4 所示。

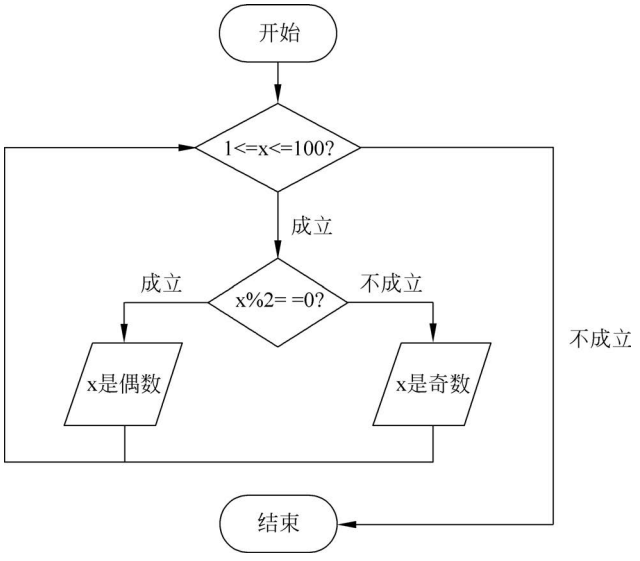



图 3-4　程序流程图

## 3.2　顺序结构

Python 顺序结构是让程序按照从头到尾的顺序依次执行每一条 Python 代码，不重复执行任何代码，也不跳过任何代码，代码如例 3-1 所示。

**【例 3-1】** 顺序结构的代码示例。

```
number1 = eval(input("请输入第一个数:"))
number2 = eval(input("请输入第二个数:"))
print(f"{number1} + {number2} = {number1 + number2}")
```

输出结果如下：

```
请输入第一个数:10
请输入第二个数:20
10 + 20 = 30
```

## 3.3　分支结构

顺序结构的代码都是按顺序执行的，也就是先执行第 1 条语句，然后是第 2 条、第 3 条……直到最后一条语句。

但是很多情况下，顺序结构的代码是远远不够用的。实际开发场景中会有很多因为满足不同条件需要执行不同操作的需求。例如，网页界面的菜单栏中，需要选择不同选项来显示不同的网页内容，或满足不同条件时进行不同的数据处理等。接下来进行分支结构的介绍。

### 3.3.1　单分支结构：if 语句

if 语句实现单分支处理，是最简单的条件判断语句。if 语句的一般形式如下：

```
if 表达式:
    语句块 1
```

如果表达式成立(真)，那么执行后面的代码块 1；如果表达式不成立(假)，那么什么也不执行，代码如例 3-2 所示。

**【例 3-2】** 单分支结构的代码示例。

```
score_cet4 = 424
if score_cet4 < 425:
    print("大学英语四级考试未通过,请继续努力。")
print("单分支程序完成。")
```

输出结果如下：

```
大学英语四级考试未通过,请继续努力。
单分支程序完成。
```

由缩进可以看出，第一个 print()输出语句属于此单分支结构，而第二个 print()输出语句则不属于该结构。所以 if 单分支条件成立与否，只决定第一个 print()函数是否输出数据；第二个 print()函数与 if 单分支结构无关。

**注意：**

(1) 每个 if 条件后要使用冒号(:)，是固定的语法规范，必须要写，表示判断条件的结束，开始分支语句的内容。

（2）使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。

### 3.3.2 双分支结构：if-else 语句

if 语句只能执行满足条件的程序代码，若不满足条件而需要执行其他程序代码，这时可以用 if-else 语句来实现。if-else 语句的基本格式如下：

```
if  表达式:
    代码块 1
else:
    代码块 2
```

如果表达式成立（真），则执行代码块 1；如果表达式不成立（假），则执行代码块 2，代码如例 3-3 所示。

**【例 3-3】** 双分支结构的代码示例。

```
score_cet4 = 500
if score_cet4 < 425:
    print("大学英语四级考试未通过,请继续努力。")
else:
    print("恭喜你成功通过大学英语四级考试。")
print("双分支程序完成。")
```

输出结果如下：

```
恭喜你成功通过大学英语四级考试。
双分支程序完成。
```

双分支结构中执行 else 子句的隐含条件相当于是 score_cet4＞＝425。注意：if 和 else 的子句都需要缩进。

### 3.3.3 多分支结构：if-elif-else 语句

还有另一种情况，如果需要判断的条件数大于两种，而 if 和 if-else 语句显然无法完成多条件判断，这时就需要用 if-elif-else 多条件判断语句。其基本格式如下：

```
if 表达式 1:
    代码块 1
elif 表达式 2:
    代码块 2
    …
elif 表达式 n:
    代码块 n
else:
    代码块 n + 1
```

对于 if-elif-else 多条件判断语句，Python 会从上到下逐个判断表达式是否成立，一旦遇到成立的表达式，就执行后面紧跟的代码块；此时，不管后面的表达式是否成立，剩下的代码都不再执行。如果所有的表达式都不成立，则执行 else 后面的代码块，代码如例 3-4 所示。

【例 3-4】　多分支结构的代码示例。

```
score_cet4 = int(input("请输入您的四级分数:"))
if score_cet4 < 425:
    print("您的四级成绩未通过,请继续努力。")
elif score_cet4 <= 500:
    print("您的四级成绩合格。")
elif score_cet4 <= 600:
    print("您的四级成绩良好。")
else:
    print("您的四级成绩优秀。")
print("多分支程序完成。")
```

输出结果如下：

```
请输入您的四级分数:600
您的四级成绩良好。
多分支程序完成。
```

### 3.3.4　嵌套分支结构：if 嵌套语句

嵌套分支是指 if、if-else 和 if-elif-else 语句还可以根据具体情况嵌入 if、if-else 或 if-elif 语句。一般应用在有多维度条件的实际问题中，其基本格式如下：

```
if 表达式 1:
    代码块 1
    if 表达式 2:
        代码块 2
    else 表达式 3:
        代码块 3
        …
elif 表达式 n:
    代码块 n
    if 表达式 n+1:
        代码块 n+1
    else:
        代码块 n+2
…
```

对于嵌套分支的执行流程，遵循从外到内依次判断的顺序，而嵌套的不管是单分支、双分支还是多分支，其执行流程和其他分支结构的规则一致，代码如例 3-5 所示。

【例 3-5】　嵌套分支结构的代码示例。

```
ticket = True
liquid_v = 150

if ticket:
    print('已有机票,请安检……')
    if liquid_v > 100:
        print('液体体积为{}ml:超出限定容量,禁止登机'.format(liquid_v))
    else:
```

```
        print('液体体积为{}ml:没有超过限定容量,允许入内'.format(liquid_v))
else:
    print('请先买票')
```

输出结果如下：

```
已有机票,请安检……
液体体积为 150ml:超出限定容量,禁止登机
```

## 3.4 循环语句

在日常生活中或是在程序所处理的问题中常遇到需要重复处理的问题，处理这些重复问题的常用方法是编写若干个重复语句。虽然这种方法可以满足需求，但是代码冗余度高、工作量大、效率低。因此，想要高效地重复执行某些操作，可以使用循环语句。

### 3.4.1 while 语句

while 语句和 if 语句类似，即在条件（表达式）为真的情况下，会执行相应的代码块。不同之处在于，只要条件为真，while 语句会一直重复执行相应的代码块。while 语句的语法格式如下：

```
while 条件表达式:
    代码块
```

这里的代码块指的是缩进格式相同的多行代码，在语句结构中，它又被称为循环体。while 语句执行的具体流程为：首先判断条件表达式的值，其值为真（True）时，则执行代码块中的语句，当执行完毕后，再重新判断条件表达式的值是否为真。若仍为真，则继续执行代码块，如此循环，直到条件表达式的值为假（False），则终止循环，代码如例 3-6 所示。

**【例 3-6】** while 语句的代码示例。

```
# 求解 10 的阶乘
result = 1
i = 1
while i <= 10:
    result *= i
    i += 1
print(result)
```

以上代码求解 10 的阶乘结果为 3628800，使用 while 语句，需要注意三点：循环控制变量 i 的初始值、对应的循环条件和改变循环控制变量的语句，即 i+=1。另外，存储最终乘积的 result 也需要赋予初始值，否则程序语句 result *=i 会报错。

Python 中的 while 语句可以跟分支结构中的 else 关键字搭配使用。当 while 语句正常结束后，会执行 else 中的语句；若是非正常终止的，即循环是通过 break 关键字直接终止的，则不会执行 else 中的语句。

例如，若要实现数字炸弹的游戏，则可以使用 while 语句和 else 的搭配来实现。其中游

戏规则如下。

(1) 随机生成指定范围内的某一个数字，如 1～1000 中的随机整数。

(2) 然后循环地猜测随机数是哪个，直到猜中为止。

代码如例 3-7 所示。

**【例 3-7】** while 语句和 else 搭配使用的代码示例。

```
from random import randint
number = randint(1, 1000)
guess_number = int(input("请随机输入一个整数:"))
while number != guess_number:
    if number < guess_number:
        print("您猜的数字大了!请再猜一次。")
    elif number > guess_number:
        print("您猜的数字小了!请再猜一次。")
    guess_number = int(input("请随机输入一个整数:"))
else:
    print("恭喜您!猜对了!")
```

输出结果如下：

```
请随机输入一个整数:500
您猜的数字小了!请再猜一次。
请随机输入一个整数:750
您猜的数字大了!请再猜一次。
请随机输入一个整数:600
您猜的数字小了!请再猜一次。
请随机输入一个整数:650
您猜的数字大了!请再猜一次。
请随机输入一个整数:625
您猜的数字大了!请再猜一次。
请随机输入一个整数:620
您猜的数字小了!请再猜一次。
请随机输入一个整数:621
恭喜您!猜对了!
```

while 语句循环的条件是 number!＝guess_number，循环正常终止的对应条件正好是 number 与 guess_number 相等，也正好执行 else 内的语句，输出结果“恭喜您！ 猜对了!”。

### 3.4.2　for 语句

3.4.1 节已详细讲解了 while 语句，本节介绍 for 语句，它常用于遍历字符串、列表、元组、字典、集合等可迭代对象，逐个获取这些对象中的各个元素。for 语句的语法格式如下：

```
for 循环变量 in 可迭代对象:
    代码块
```

循环变量用于存放从可迭代对象中读取的元素，所以一般不会在循环结构中对循环变量手动赋值；循环的对象一般为字符串、列表、元组、字典、集合等数据；代码块指的是具有相同缩进格式的多行代码(和 while 语句一样)，由于与循环结构联用，因此代码块又被称为循环体。

for 语句可以遍历字符串、列表、字典和集合等可迭代对象，代码如例 3-8 所示。

**【例 3-8】** for 语句的代码示例。

```
# 1. for 语句循环遍历字符串
str1 = "武汉是一座英雄的城市"
for character in str1:
    print(character, end = "")

# 2. for 语句循环遍历列表
circle_color = ["blue", "black", "red", "yellow" ,"green"]
for color in circle_color:
    print(color, end = " ")

# 3. for 语句循环遍历字典
stu_dict = {"院系": "计信学院", "学号": "12345678", "姓名": "小哈"}
for key in stu_dict:
    print(key, stu_dict[key])

# 4. for 语句循环遍历集合
Set = {(3, 4), True, "abc", 12.3}
for i in Set:
    print(i, end = " ")
```

输出结果如下：

```
# 1. for 语句循环遍历字符串
武汉是一座英雄的城市

# 2. for 语句循环遍历列表
blue black red yellow green

# 3. for 语句循环遍历字典
院系 计信学院
学号 12345678
姓名 小哈

# 4. for 语句循环遍历集合
abc True (3, 4) 12.3
```

### 3.4.3 嵌套循环

不仅 if 语句支持相互嵌套，while 和 for 循环结构也支持嵌套。例如，for 循环里面包含 for 循环、while 循环里面包含 while 循环，甚至 while 循环中包含 for 循环或 for 循环中包含 while 循环也都是允许的。

当两个（甚至多个）循环结构相互嵌套时，位于外层的循环结构常简称为外层循环或外循环，位于内层的循环结构常简称为内层循环或内循环，代码如例 3-9 所示。

**【例 3-9】** 使用嵌套 for 循环求 1!＋2!＋3!＋…＋n!的代码示例。

```
n = int(input("请输入计算阶乘的个数:"))
sum1 = 0
```

```
for i in range(1, n + 1):
    mul = 1
    for j in range(1, i + 1):
        mul * = j
    sum1 += mul
print("1!+ … + {}!= {}".format(n, sum1))
```

输出结果如下：

```
请输入计算阶乘的个数:5
1!+ … + 5!= 153
```

外层循环变量 i 等于 1，内层循环计算 1 的阶乘，然后加上 sum1；接着外层循环 i 变成 2，内层循环再计算 2 的阶乘，再加上 sum1，后面依次计算 3!、4! 和 5!，并且求和，存入 sum1 中，最后得出阶乘总和为 153。

用 while 循环来求解 1!＋2!＋3!＋…＋n!，代码如例 3-10 所示。

**【例 3-10】** 使用嵌套 while 循环求 1!＋2!＋3!＋…＋n!的代码示例。

```
n = int(input("请输入计算阶乘的个数:"))
sum1, i = 0, 1              # 循环变量 i 的初值
while i < n + 1:            # 外层循环的条件
    mul = j = 1             # 循环变量 j 的初值
    while j < i + 1:        # 内层循环的条件
        mul * = j
        j += 1              # 改变内层循环变量 j 的语句
    sum1 += mul
    i += 1                  # 改变外层循环变量 i 的语句
print("1!+ … + {}!= {}".format(n, sum1))
```

输出结果如下：

```
请输入计算阶乘的个数:6
1!+ … + 6!= 873
```

由例 3-9 和例 3-10 可以看出，一般情况下，当循环次数固定时，for 循环的写法比 while 循环更简单；而当循环次数不固定，且跟循环体的执行情况相关时，一般可以优先选择使用 while 循环。

Python 解释器执行循环嵌套结构代码的流程如下。

(1) 当外层循环条件成立时，执行外层循环结构中的循环体。

(2) 进入外层循环的循环体后，执行内层循环，直到内层循环结束。

(3) 内层循环结束后，再次判断外层循环条件，若仍成立，则返回第(2)步，反复执行，直到外层循环的循环条件不成立为止，整个循环结束。

(4) 当内层循环的循环条件不成立，且外层循环的循环条件也不成立时，则整个嵌套循环才算执行完毕。

在计算机领域中，认为编写的程序＝数据结构＋算法，数据结构的好坏直接关系到程序的质量，而在实际开发过程中经常会用到的各类排序算法的思想，如冒泡排序、选择排序和快速排序等，不同的排序方法处理数据的效率各有不同。现在以冒泡排序的代码为例，阐述

嵌套循环的代码特点及流程。

冒泡排序(Bubble Sort)是一种简单、直观的排序算法。它重复地访问要排序的数列，一次比较两个元素。以升序为例，如果比较的两个元素前一个大于后一个，就对它们的位置进行交换，然后重复地进行，直到没有再需要交换的元素，那么该数列已经排序完成。

冒泡排序的思想如下。

第一，比较相邻的元素。如果第一个比第二个大，就对它们进行交换。

第二，对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

第三，针对所有的元素重复以上的步骤，除了最后一个。

第四，持续对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

演示代码如例 3-11 所示。

**【例 3-11】** 嵌套循环——冒泡排序代码示例。

```
sort_num = [34, 12, 67, 13, 2, 100, 79, 23, 35, 89]
length = len(sort_num)
for i in range(length - 1):
    for j in range(length - i - 1):
        if sort_num[j] > sort_num[j + 1]:
            sort_num[j], sort_num[j + 1] = sort_num[j + 1], sort_num[j]
print(sort_num)
```

输出结果如下：

```
[2, 12, 13, 23, 34, 35, 67, 79, 89, 100]
```

由冒泡排序的思想可以推出，n 个数字只需要比较 n－1 轮，也就是例 3-11 中第一个 for 循环的 length－1；而每一轮内部的比较次数依次是 n－1 次、n－2 次、n－3 次、……、2 次、1 次，也就是第二个 for 循环中的 length－i－1。然后，Python 中交换两个变量中的数据可以直接写成 a,b＝b,a 的形式，所以交换 sort_num[j]和 sort_num[j＋1]，就可以直接写成 sort_num[j],sort_num[j＋1]＝sort_num[j＋1],sort_num[j]。

### 3.4.4 循环控制语句：break、continue 和 pass 语句

在执行 while 循环或 for 循环时，只要循环条件满足，程序将会一直执行循环体，不停地循环。但某些场景需要在循环结束前就强制结束循环，Python 提供了两种强制离开当前循环体的方法：break 语句和 continue 语句。此外，Python 还提供了 pass 语句，用来让解释器跳过此处，什么都不做。

#### 1. break 语句

break 语句可以立即终止当前循环的执行，跳出当前所在的循环结构。无论是 while 循环还是 for 循环，只要执行 break 语句，就会直接结束当前正在执行的循环体。

这就好比在操场上跑步，原计划跑 5 圈，可是当跑到第 3 圈的时候，临时有事，于是直接停止跑步并离开操场，这就相当于使用了 break 语句提前终止了循环。

break 语句的语法非常简单，只需要在相应 while 或 for 循环中直接加入即可，代码如

例 3-12 所示。

**【例 3-12】** 使用 break 语句的代码示例。

```
for i in range(1, 6):
    if i == 3:
        print("第 3 圈临时有事,停止锻炼。")
        break
    print("这是第{}圈"。format(i))
```

输出结果如下：

```
这是第 1 圈
这是第 2 圈
第 3 圈临时有事,停止锻炼。
```

例 3-7 中实现数字炸弹游戏,是使用 while 循环和 else 搭配实现的,那么使用 while 循环和 break 搭配也同样能实现,代码如例 3-13 所示。

**【例 3-13】** while 循环和 break 搭配使用的代码示例。

```
from random import randint
number = randint(1,1000)
while True:
    guess_number = int(input("请随机输入一个整数:"))
    if number < guess_number:
        print("您猜的数字大了!请再猜一次。")
    elif number > guess_number:
        print("您猜的数字小了!请再猜一次。")
    else:
        print("恭喜您!猜对了!")
        break
```

输出结果如下：

```
请随机输入一个整数:500
您猜的数字大了!请再猜一次。
请随机输入一个整数:250
您猜的数字小了!请再猜一次。
请随机输入一个整数:375
您猜的数字小了!请再猜一次。
请随机输入一个整数:435
您猜的数字大了!请再猜一次。
请随机输入一个整数:405
您猜的数字小了!请再猜一次。
请随机输入一个整数:420
您猜的数字大了!请再猜一次。
请随机输入一个整数:415
您猜的数字大了!请再猜一次。
请随机输入一个整数:410
恭喜您!猜对了!
```

将 while 循环的条件设置为 True,是 while 循环中的一种特殊写法,True 代表循环条件永远成立,也就意味着循环无法通过条件终止,但可以通过 break 来进行终止,例 3-13

中，不论玩家猜的数字是大还是小，都无法终止循环，只有当猜对之后，需要终止，所以在else分支内部加一个break即可。

**2. continue语句**

和break语句相比，continue语句的作用则没有那么强大，它只能终止执行本次循环中剩下的代码，直接从下一次循环继续执行。

仍然以在操场上跑步为例，原计划跑5圈，但跑到第3圈累了，第3圈不跑，改为走一圈，休息一会儿，并直接从第4圈开始跑。

continue语句的用法和break语句一样，只要在while或for循环中的相应位置加入即可，代码如例3-14所示。

**【例3-14】** continue语句的使用代码示例。

```
for i in range(1, 6):
    if i == 3:
        print("第3圈走一圈,休息一会儿。")
        continue
    print("第{}圈跑步运动".format(i))
```

输出结果如下：

```
第1圈跑步运动
第2圈跑步运动
第3圈走一圈,休息一会儿。
第4圈跑步运动
第5圈跑步运动
```

**3. pass语句**

在实际开发中，有时会先搭建起程序的整体逻辑结构，暂时不去实现某些细节，而是在这些地方加一些注释，方便以后再添加代码，同样可以用pass语句作为占位符，实现同样的功能，且程序代码看起来更简洁、美观，代码如例3-15所示。

**【例3-15】** 使用pass语句的代码示例。

```
for i in range(1, 6):
    if i == 3:
        pass
    print("第{}圈跑步运动".format(i))
```

输出结果如下：

```
第1圈跑步运动
第2圈跑步运动
第3圈跑步运动
第4圈跑步运动
第5圈跑步运动
```

该代码示例中，当i等于3时，满足分支条件，但是里面处理的代码，还没有确定怎么写，那么可以先用pass语句进行占位，此时相当于满足条件，但什么都不做。这样就可以先运行其他代码，保持程序结构完整性，不会因为缺少语句导致程序出错。如果确定了分支结

构中的程序语句怎么写了，直接删除 pass 语句，补充对应程序语句即可。

pass 也是 Python 的关键字之一，用来让解释器跳过此处，什么都不做。就像上面的情况，有时候程序需要占一个位置或放一条语句，但又不希望这条语句被执行，此时就可以通过 pass 语句来实现。使用 pass 语句比使用注释更加美观。

## 3.5　实训案例


视频讲解

小红和小芳商量下周如何度过，如果是周末，温度高于 30℃，就去游泳，否则就去爬山；非周末，假如天气好，就去旅游，否则，就去图书馆学习。编写程序，实现上述推荐选择。

**1. 案例的实现思路**

(1) 建议定义三个变量，分别存放周几、温度和天气。

(2) 利用 if 语句来实现。

**2. 案例参考代码**

```
print('请输入周几\n')
day = input()                    #今天周六
print('请输入温度\n')
temp = input()                   #温度为 31℃
print('请输入天气\n')
weather = input()
if day == 6 or day == 7:
    if temp > 30:
       print("建议您去游泳")
    else:
       print("建议您去爬山")
else:
    if weather == "天气好":
        print("建议您去旅游")
    else:
        print("建议您去图书馆学习")
```

运行结果如图 3-5 所示。

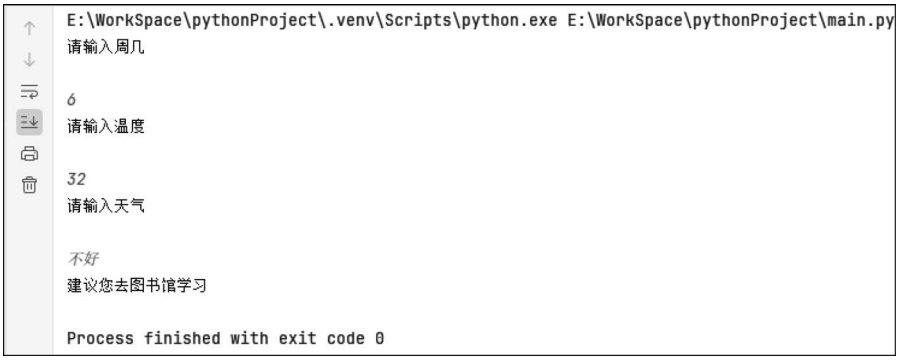



**图 3-5　运行结果**

## 本章小结

本章主要介绍 Python 的流程控制结构，包括顺序结构、分支结构、循环结构，使用流程控制语句可以控制程序的执行过程。其中，需要重点掌握 if、if-else、if-elif-else 分支结构、while、for 循环结构、选择及循环的嵌套用法。

## 习题

### 一、判断题

1. 对于带有 else 子句的循环语句，如果是因为循环条件表达式不成立而自然结束循环，则执行 else 子句中的代码。（　　）

2. while 循环的循环体至少执行一次。（　　）

3. 在循环语句中，continue 语句的作用是跳过当前循环中的剩余语句，提前进入下一次循环。（　　）

4. 在循环语句中，break 语句的作用是提前结束本层循环。（　　）

5. Python 提供了两种实现循环的语句，分别是 while 和 for。（　　）

### 二、选择题

1. 执行以下代码，其结果为（　　）。

```
n = 20
sum = 0
number = 1
while number <= n:
    sum = sum + number
    number += 1
print(sum)
```

A. 0　　B. 209　　C. 210　　D. 66

2. 阅读下面的代码：

```
sum = 0
for i in range(10):
    if(i % 5):
        continue
    sum = sum + i
print(sum)
```

上述程序的执行结果是（　　）。

A. 5　　B. 4　　C. 3　　D. 10

3. 已知 x=5，y=6，z=7，以下语句执行后 x、y、z 的值是（　　）。

```
if x < y:
    z = x
    x = y
    y = z
```

A. 6,5,6　　B. 5,6,6　　C. 6,5,5　　D. 5,6,5

4. 以下程序的执行结果是(　　)。

```
s = 0
for i in range(1,10):
    s += i
    if i == 5:
        print(s)
        break
```

A. 66　　B. 15　　C. 45　　D. 0

## 三、编程题

1. 编写程序,从两个字符串中输出较长的字符串。比较两个字符串的长度,输出长度较长的字符串。如果两个字符串的长度相同,则输出第 1 个字符串。

2. 编写程序,输出九九乘法表。

3. 编写程序,利用 if 条件运算符的嵌套来完成此题:学习成绩>=90 分输出 A,60~89 分输出 B,60 分以下输出 C。