第 5 章

循 环 结 构

5.1 引　　言

5.1.1 问题导入

在第 4 章，我们学习了如何实现一个生成减法算式的程序。该程序能够随机生成一道 100 以内（含 0 和 100）的减法题目，并通过选择结构判断用户的答案，从而辅助用户进行减法练习。然而，若需要程序在一次运行时生成多个减法算式以供用户连续练习，仅仅通过重复编写生成算式的代码显然不够高效且难以维护。这种做法不仅导致程序冗长，而且在调整生成算式数量时极为不便。

解决此类问题的关键在于实现代码的自动重复执行，这正是循环结构的用武之地。循环是程序设计中的核心概念，它允许程序在满足特定条件时，重复执行某段代码块，直至达到预定目标。Python 提供了循环结构，如何使用循环结构来实现多个减法算式的生成呢？如何灵活控制代码的重复次数，以适应不同的需求？接下来，我们将深入探讨 Python 中的循环结构，学习如何利用它简化程序逻辑、提高代码的可读性和可维护性，并高效地实现更多功能。

5.1.2 知识结构导图

本章围绕如何发现重复操作中的规律，并使用循环结构来实现这些操作展开介绍。Python 提供了两种类型的循环：逻辑控制循环 while 和计数控制循环 for。本章的知识结构导图如图 5.1 所示。

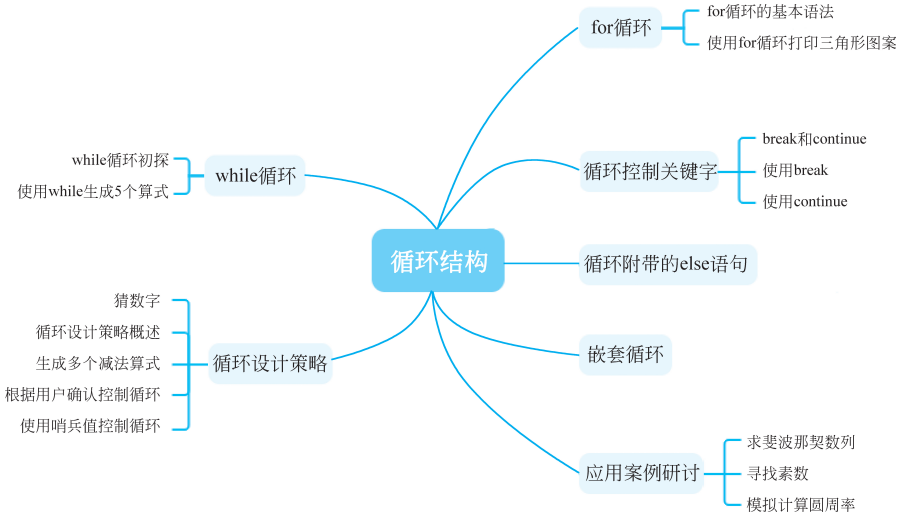

图 5.1　循环结构知识结构导图

5.2 while 循环

5.2.1 while 循环初探

while 循环是一种基本的条件控制循环，当条件表达式为 True 时重复执行语句，直到该条件变为 False。其语法如下：

```
while 条件表达式:
    语句块
```

其中，条件表达式被称为循环的继续条件，它是一个布尔表达式，其结果为 True 或 False，决定循环是否应当继续执行。条件表达式紧跟在 while 关键字之后，并通过冒号（：）分隔。缩进的语句块则被称为循环体，它在每次循环的继续条件结果为 True 时被重复执行。循环体的一次完整执行被称为一次迭代。

while 循环的执行流程如图 5.2 所示。它首先检测继续条件的结果值，若结果值为 True，则执行循环体语句，每次循环迭代完成后，Python 会重新检测继续条件的结果值，并决定是否继续执行循环体语句。只要条件表达式的结果为 True，循环体语句就会重复执行，直到继续条件的结果值为 False 时，不再执行循环体语句，循环结束，转到 while 循环语句的后继语句。

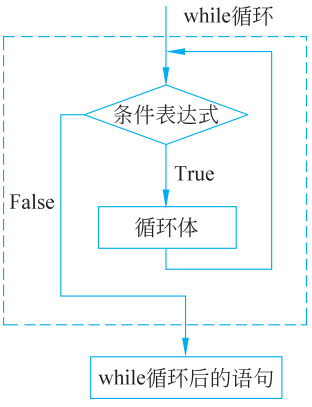

图 5.2 while 循环的执行流程

这里强调 Python 中一个至关重要的语法特性，循环体内的语句块通过明确的缩进机制来界定其作用域。Python 解释器在执行过程中，会严格依据语句块的缩进级别来识别并解析循环的开始与终止点。因此，必须确保循环体内每一条语句都遵循了正确的缩进规则，否则可能会导致逻辑错误、代码难以阅读，甚至引发运行时异常，从而严重影响程序的稳定性和性能。

阅读下列代码一和代码二，分析程序实现的功能，写出程序运行的结果。

```
#代码一
x = 0
while x < 4:
    x = x + 1
print("x is", x)

#代码二
x = 0
while x < 4:
    x = x + 1
    print("x is", x)
```

代码一和代码二的流程图如图 5.3(a)和(b)所示。代码一和代码二的循环的继续条件

都为 x ＜ 4，代码一中的循环体只有语句 x ＝ x ＋ 1 ，而代码二中的循环体有两条语句：x ＝ x ＋ 1 和 print("x is", x)。

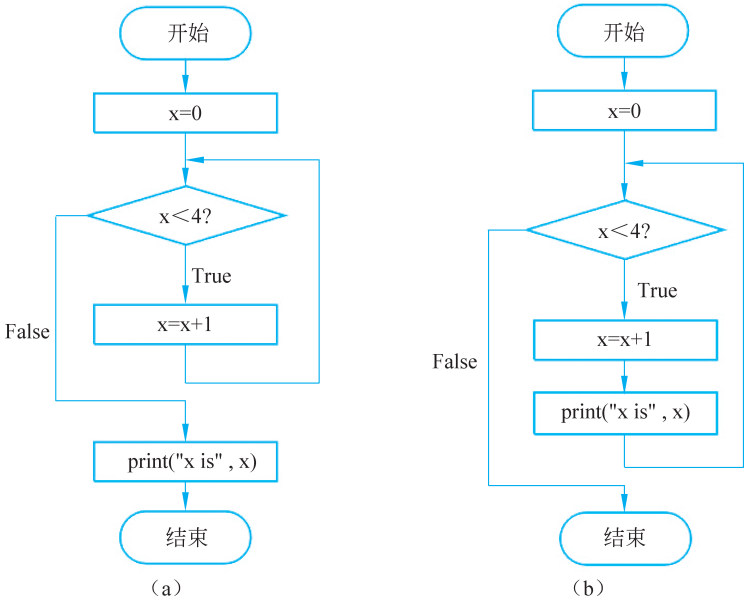

图 5.3　代码一和代码二的流程图

代码一中，当 while 循环继续条件为 True 时，执行循环体语句 x＝x＋1，共执行 4 轮。在执行第 4 轮后，x＝4，然后重新检测继续条件 x＜4 的结果为 False，循环结束，不再执行循环体语句。接着执行 while 循环后的语句 print("x is", x)，运行结果为：x is 4。

代码二中，当 while 循环继续条件为 True 时，执行循环体语句块，每一轮按顺序执行两条语句，都会执行 print("x is", x)进行输出，共执行 4 轮，因此输出 4 次。在执行第 4 轮后，x＝4，然后重新检测继续条件 x＜4 的结果为 False，循环结束，不再执行循环体语句。接着执行 while 循环后的语句，无语句可执行，程序执行结束。表 5.1 展示了代码二的循环执行过程。

表 5.1　代码二的循环执行过程

循环轮数	进入本轮循环时 x 的值	继续条件	执行本轮循环后 x 的值	输出
1	0	True	1	x is 1
2	1	True	2	x is 2
3	2	True	3	x is 3
4	3	True	4	x is 4
5	4	False	不执行循环体语句	程序结束，无输出

5.2.2　使用 while 生成 5 个算式

【微实例 5-1】 实现自动生成 5 个减法算式，每个算式由[1,100]范围的随机整数组成，并确保被减数不小于减数，提示用户逐一回答每个算式，并在提交答案后给出用户回答是否

正确的反馈。

【问题分析】

假如 count 表示生成的算式个数，count 初始值为 0，则当布尔表达式 count＜5 为 True 时，就重复执行生成一个减法算式的语句块。使用 while 循环实现生成 5 个减法算式的代码(5-1.py)如下。

```
import random
count = 0                                          #表示生成的算式个数,初始值为 0
NUMBER_OF_QUESTIONS = 5                            #常量,表示生成算式的个数
#使用 while 循环生成多个减法算式
while count < NUMBER_OF_QUESTIONS:
    #生成两个 100 以内的随机整数
    num1 = random.randint(1, 100)
    num2 = random.randint(1, 100)
    #如果 num1<num2,互换二者的值
    if num1 < num2:
        num1, num2 = num2, num1
    #提示用户回答 num1-num2=?并保存答案
    answer = eval(input(f"{num1} - {num2} =?"))
    #验证用户输入的答案是否正确,然后显示回答是否正确
    if num1 - num2 == answer:
        print("回答正确!")
    else:
        print(f"回答错误。\n{num1}-{num2}={num1 - num2}")
    count += 1                                     #count 值增加 1,表示生成算式个数加 1
```

运行程序，可能的一次运行结果如下，问号(?)后面的数值为用户输入的答案。

```
64 - 23=? 5
回答错误。
64-23=41
94 - 38=? 58
回答错误。
94-38=56
79 - 13=? 66
回答正确!
98 - 79=? 19
回答正确!
46 - 18=? 24
回答错误。
46-18=28
```

5.3 循环设计策略

5.3.1 猜数字

【微实例 5-2】 随机生成 0～100(包含 0 和 100)的整数，猜猜计算机里存储的数字是什

么。提示用户输入一个数字，直到该数字与随机生成的数字相匹配。对于输入的数字，如果猜中，输出“恭喜，猜中啦!”；如果未猜中，提示用户猜的数字是否太低或太高。

【问题分析】

随机生成的数字范围为[0,100]。为了尽快猜中计算机里存储的数字，可以采用二分折半的方法来猜。首先猜0～100的中间整数值为50，如果提示太高了，那么可以得出随机生成的数字范围为[0,49]。如果提示太低了，则推断出随机生成的数字范围为[51,100]。按类似的规则在缩小的范围里继续猜，直到猜到为止。如何编写程序实现此问题的求解呢？下面首先分析猜一次的过程。

(1) 输入：随机生成0～100(包含0和100)的整数；提示用户连续输入一个数字。

(2) 处理和输出：比较用户输入的数字和随机生成的数字，若用户输入的数字等于随机生成的数字，输出“恭喜，猜中啦!”；若用户输入的数字小于随机生成的数字，输出“猜得太高啦!”；若用户输入的数字大于随机生成的数字，输出“猜得太低啦!”。

猜一次的实现代码如下：

```
1   import random
2
3   number = random.randint(0, 100)                    #生成一个0~100的随机整数
4   #使用input()函数提示用户输入一个猜测的数字,并使用eval()函数将输入转换为整数
5   guess = eval(input("请在0~100猜数字!输入猜的整数: "))   #提示输入数字
6   if guess == number:                                #如果猜测的数字等于随机数
7       print("恭喜,猜中啦!")                          #输出恭喜信息
8   elif guess > number:                               #如果猜测的数字大于随机数
9       print("猜得太高啦!")                           #提示用户猜得太高
10  else:                                              #如果猜测的数字小于随机数
11      print("猜得太低啦!")                           #提示用户猜得太低
```

运行上面的程序，提示一次用户输入猜的数字，如果猜中，输出“恭喜，猜中啦!”，如果用户未猜中，即guess!=number时，还想让用户继续猜的话，就需要重复执行第5～11行的代码。把这段代码用while循环包裹起来，并把guess!=number作为循环的条件表达式，其值为True时，表示还需要猜数字，其值为False时，表示猜中，循环结束。需要注意，把第3行放到while循环后，guess需要赋一个初始值且满足第一次执行循环条件判断为True(即guess初始值不为0～100)，保证while循环至少执行一次，也就是猜一次就猜中的情况。完整的实现代码(5-2.py)如下。

```
import random

number = random.randint(0, 100)                    #生成一个0~100的随机整数
guess = -1                                         #初始化guess为-1,确保进入循环
#使用while循环让用户猜测数字,直到猜中为止
while guess != number:
    guess = eval(input("请在0~100猜数字!输入猜的整数: "))#提示输入数字
    if guess == number:                            #如果猜测的数字等于随机数
        print("恭喜,猜中啦!")                      #输出恭喜信息
    elif guess > number:                           #如果猜测的数字大于随机数
```

```
        print("你猜得太高了")                    #提示用户猜得太高
    else:                                       #如果猜测的数字小于随机数
        print("你猜得太低了")                    #提示用户猜得太低
```

运行程序，可能的一次输出结果如下：

```
请在 0~100 猜数字!输入猜的整数: 50
你猜得太低了
请在 0~100 猜数字!输入猜的整数: 75
你猜得太低了
请在 0~100 猜数字!输入猜的整数: 90
你猜得太高了
请在 0~100 猜数字!输入猜的整数: 85
你猜得太低了
请在 0~100 猜数字!输入猜的整数: 86
恭喜,猜中啦!
```

5.3.2 循环设计策略概述

根据微实例 5-2 的分析实现过程，可以将循环设计策略总结为以下三步。

1. 确认需要重复执行的操作

明确任务中哪些操作是需要重复进行的，即循环的语句。这些操作构成循环的“主体”或“循环体”。

2. 把循环语句包裹在一个循环里

选用合适的循环结构（如 while 循环或 for 循环）来包裹这些操作。选择哪种循环取决于具体需求。如果不知道循环需要执行多少次，或者循环的执行次数取决于某些条件，选 while 循环更合适；如果知道循环需要执行的确切次数，那么 for 循环可能是更好的选择。

以 while 循环为例，包裹在 while 循环里的语句格式如下：

```
while True:
    循环语句
```

3. 编写循环中的继续条件并添加控制循环结束的逻辑

（1）设置初始条件：在 while 循环之前，设置任何必要的变量或条件，这些变量或条件将在循环体内被修改，以决定循环何时结束。

（2）循环体中的逻辑：在循环体内，编写需要重复执行的代码，并可能包括修改用于控制循环条件的变量。

（3）跳出循环的条件：在循环体内，可以使用 break 语句来明确地跳出循环，或者使用条件表达式来控制 while 循环的条件，使其最终变为 False，从而自然结束循环。

以 while 循环为例设计策略的语法格式顺序如下：

```
设置初始条件语句                                #第三步
while 条件表达式:                               #第二步
```

循环语句	#第一步
其他用于控制循环的语句	#第三步

5.3.3　生成多个减法算式

【微实例 5-3】　在微实例 5-1 的问题描述中，实现生成多个减法算式，生成算式个数由用户来决定，用户回答完所有的问题后，输出正确答案的个数，同时显示完成所有算式所用的时间。

【问题分析】

按照 5.3.2 节循环设计策略概述进行分析，该程序设计分为以下三步。

第一步：确定需要被循环的语句。

要生成多个减法算式，首先生成一个减法算式，然后重复生成一个减法算式的语句若干次，重复次数由用户输入的次数决定。生成一个减法算式的过程如下：首先获取两个随机数，判断第一个数是否小于第二个数，若是，则交换两个数，并提示用户做减法。然后比较用户输入的结果是否与正确计算结果一致，给此题打分。

第二步：将完成一个减法算式和打分的语句包裹在 while 循环里。

第三步：编写 while 循环中的继续条件，实现 5 个减法算式的生成和打分。

在编写程序时，需要考虑初始值的设定、循环控制变量的添加以及循环继续条件的设定。程序开始时，应将正确回答数 correctCount 初始化为 0，并在每次循环中根据答题情况更新此值：答对一题则加一，答错则保持不变。答完所有题目后，correctCount 即为总答对题数。同时，添加循环控制变量 count 并初始化为 0，每次循环后自增 1。当 count 小于设定的题目数量时，程序继续；达到或超过题目数量时，循环结束。这样即可控制循环次数，实现程序控制。同时添加记录用户开始答题的时间变量 startTime 和答题结束时的时间变量 endTime，用于计算完成所有算式所用的时间来评估用户的答题速度。完整的实现代码(5-3-1.py)如下。

```
import random
import time
numberOfQuestions = int(input("请输入生成算式的个数:"))    #输入生成算式的个数
count = 0                                #算式个数计时器,表示生成的算式个数,初始值为 0
correctCount = 0                         #正确计数器,表示答对的个数,初始值为 0
startTime = time.time()                  #开始做题的时间

while count < numberOfQuestions:
    #1.生成两个随机数
    num1 = random.randint(1, 100)
    num2 = random.randint(1, 100)
    #2. 确保 num1 大于 num2
    if num1 < num2:
        num1, num2 = num2, num1
    #3. 提示用户回答 num1-num2=?并保存答案
    answer = eval(input( f"{num1}-{num2}=?"))
    #4. 验证用户输入的答案是否正确,然后显示回答是否正确,答对时正确计数器加 1
```

```
        if num1 - num2 == answer:
            print("回答正确!")
            correctCount += 1
        else:
            print(f"回答错误。\n{num1}-{num2}={num1 - num2}")

        #5. 算式个数计时器加 1
        count += 1
    endTime = time.time()                          #答题结束的时间
    testTime = int(endTime - startTime)            #获得答题用时
    print(f"{numberOfQuestions}题答对了{correctCount}个题目。用时{testTime}秒。")
```

运行程序，输入 2 后回车，可能的一次输出结果如下。

```
请输入生成算式的个数:2
23-16=? 7
回答正确!
35-16=? 19
回答正确!
2 题答对了 2 个题目。用时 12 秒。
```

5.3.4 根据用户确认控制循环

如果想让用户来决定是否还要继续循环，如何实现？可以创建一个记录用户意愿的变量，通过用户输入值来控制循环的继续或结束。当用户输入 Y 时，继续循环，输入 N 时结束循环，实现代码框架如下。

```
continueLoop = 'Y'                          #初始化循环控制变量为 'Y',表示可以开始循环
while continueLoop == 'Y':                  #当 continueLoop 等于 'Y' 时,继续循环
    ...                                     #这里编写循环体内部需要执行的代码
    #获取用户输入,根据用户的选择决定是否继续循环
    continueLoop = input("输入'Y'继续循环,输入'N'退出循环: ")
```

例如，在微实例 5-3 中，由用户来决定是否继续答题，当用户输入 Y 时表示想答题，输入 N 时表示不想答题，实现代码(5-3-2.py)如下。

```
import random
import time
continueLoop = input("请输入是否想答题,输入 Y 表示想答题,输入 N 表示不想答题:")
count = 0                                   #算式个数计时器,表示生成的算式个数,初始值为 0
correctCount = 0                            #正确计数器,表示答对的个数,初始值为 0
startTime = time.time()                     #开始做题的时间

while continueLoop =='Y':
    #1.生成两个随机数
    num1 = random.randint(1, 100)
    num2 = random.randint(1, 100)
```

```
        #2. 确保 num1 大于 num2
        if num1 < num2:
            num1, num2 = num2, num1
        #3. 提示用户回答 num1-num2=?并保存答案
        answer = eval(input( f"{num1}-{num2}=?"))
        #4. 验证用户输入的答案是否正确,然后显示回答是否正确,答对时正确计数器加 1
        if num1 - num2 == answer:
            print("回答正确!")
            correctCount += 1
        else:
            print(f"回答错误。\n{num1}-{num2}={num1 - num2}")

        #5. 算式个数计时器加 1
        count += 1
        #输入是否答题
        continueLoop = input("是否想继续答题,输入 Y 继续答题,输入 N 不想答题:")

    endTime = time.time()                          #答题结束的时间
    testTime = int(endTime - startTime)            #获得答题用时
    print(f"{count}题答对了{correctCount}个题目。用时{testTime}秒。")
```

运行程序,可能的一次输出结果如下。

```
是否想继续答题,输入 Y 继续答题,输入 N 不想答题:Y
99-83=? 16
回答正确!
是否想继续答题,输入 Y 继续答题,输入 N 不想答题:Y
67-54=? 13
回答正确!
是否想继续答题,输入 Y 继续答题,输入 N 不想答题:N
2 题答对了 2 个题目。用时 26 秒。
```

5.3.5　使用哨兵值控制循环

若循环执行的次数不是预先确定的,可指定一个特殊的输入值来表示循环结束,这个特殊值称为哨兵值。使用哨兵值来控制循环的方式称作步哨式控制,就像哨兵在岗位上根据特定信号来控制通行一样。

【微实例 5-4】 实现一个猜数字游戏,计算机生成一个随机数,玩家通过输入数字来猜测,可以随时输入 Bye 选择退出游戏。

【问题分析】

使用一个名为 user_input 的变量来存储输入值。使用一个名为 answer 的变量来存储产生的随机值。只要用户输入的值不为 Bye(称 Bye 为哨兵值),就让用户重复猜数字的游戏。使用 while 循环的实现代码(5-4.py)如下。

```
1   import random
2   answer = random.randint(1, 100)              #生成一个 1~100 的随机数作为答案
```

```
#提示用户游戏规则和如何退出游戏
print("欢迎来到猜数字游戏!我已经生成了一个 1~100 的随机数。")
print("你可以输入一个数字来猜测答案。输入 'Bye' 来退出游戏。")
user_input = input("请输入你的猜测: ")          #获取用户的首次输入
#开始游戏循环,直到用户输入 'Bye' 退出
while user_input.lower() != 'bye':
    guess = int(user_input)                     #尝试将用户输入转换为整数进行猜测
    if guess == answer:                         #如果猜测的数字与答案相同
        print("恭喜,你猜对了!再猜一轮!")        #恭喜猜对,并告知开始新一轮游戏
        answer = random.randint(1, 100)         #生成一个新的随机数作为答案
    elif guess < answer:                        #如果猜测的数字小于答案
        print("猜的数字太小了,请再试一次。")    #提示用户数字太小
    else:                                       #如果猜测的数字大于答案
        print("猜的数字太大了,请再试一次。")    #提示用户数字太大
    user_input = input("请输入你的猜测,退出输入'Bye': ")   #再次获取用户输入
```

程序第 2 行生成一个 1～100 的随机数作为答案，然后玩家通过输入数字来猜测答案。玩家可以随时选择退出游戏，输入 Bye 即可退出。程序会根据玩家的猜测提供反馈，在玩家猜对答案时又重新生成一个随机数字（第 12 行），开始下一轮的猜数字游戏。程序运行一次的结果如下：

```
欢迎来到猜数字游戏!我已经生成了一个 1~100 的随机数。
你可以输入一个数字来猜测答案。输入 'Bye' 来退出游戏。
请输入你的猜测: 50
猜的数字太大了,请再试一次。
请输入你的猜测,退出输入'Bye': 25
猜的数字太大了,请再试一次。
请输入你的猜测,退出输入'Bye': 12
猜的数字太大了,请再试一次。
请输入你的猜测,退出输入'Bye': 6
猜的数字太大了,请再试一次。
请输入你的猜测,退出输入'Bye': 3
猜的数字太大了,请再试一次。
请输入你的猜测,退出输入'Bye': 2
恭喜,你猜对了!再猜一轮!
```

5.4 for 循环

5.4.1 for 循环的基本语法

通常情况下，当循环的执行次数是已知的时，可以使用一个控制变量来统计执行次数。这类循环被称为计数控制循环，而 for 循环就是一种典型的计数控制循环。其基本格式如下：

```
for 临时变量 in 可迭代对象:
    循环体
```

在 for 语句中，in 后面的可迭代对象可以为一个列表、一个字符串等。当循环执行时，