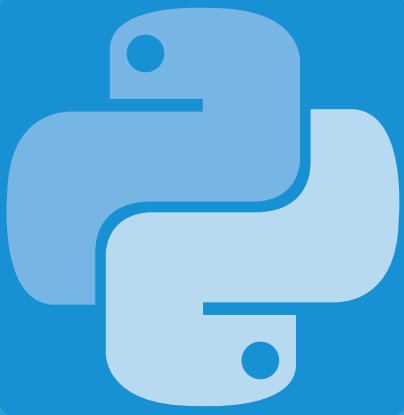

第3章 程序控制结构

程序控制结构是决定代码如何执行的关键，Python主要包括顺序结构、选择结构和循环结构三种。本章重点介绍选择结构、循环结构，并通过大量的实例演示这些流程控制语句的使用方法。

3.1 基本程序结构

Python基本程序结构主要包括顺序结构、条件结构（分支结构）和循环结构三种。

1. 顺序结构

顺序结构是最基本的结构，代码按照从上到下的顺序逐行执行，无分支或循环，是结构化程序设计中最简单的基础结构。示例如下：

```
a =5                    # 第1步：赋值
b = 12                  # 第2步：赋值
c = a + b               # 第3步：计算
print("结果：", c)      # 第4步：输出
```

2. 条件结构

条件结构又叫分支结构或选择结构，根据条件判断结果，选择不同的代码分支执行。Python主要使用if、elif和else语句来实现。

3. 循环结构

循环结构重复执行某段代码，直到满足终止条件。Python有for循环和while循环两种主要的循环方式。

3.2 条件结构

在Python语言中，条件语句通过判断条件决定程序的执行路径。简单来说，就是对语句中的条件进行判断，并根据不同的条件执行不同的语句。条件结构主要包括以下形式。

- **单分支（if）：**仅当条件为真时执行代码块。
- **二分支（if-else）：**条件为真时执行一个分支，否则执行另一个分支。
- **多分支（if-elif-else）：**依次判断多个条件，执行第一个满足条件的分支。

3.2.1 单分支if语句

单分支条件结构由if语句单独组成，其基本语法格式如下：

```
if 条件：
    代码块
```

其中，if标志条件判断的开始，条件是结果为布尔值（True/False）或可转换为布尔值的表达式。如果表达式的值为真，则执行代码块，否则跳过代码块，继续执行后面的语句。具体流程如图3-1所示。

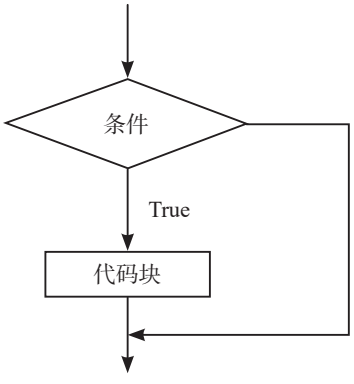

图 3-1

例3-1 单分支条件结构应用示例

功能描述：使用简单的if语句判断用户输入的年龄是否成年，如果成年，则输出“您已成年！”。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 判断用户年龄是否成年
age = int(input("请输入年龄："))  # 获取用户输入并转为整数

if age >= 18:  # 条件表达式后加冒号`:`
    print("您已成年！")  # 缩进4个空格或1个Tab
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-2所示。

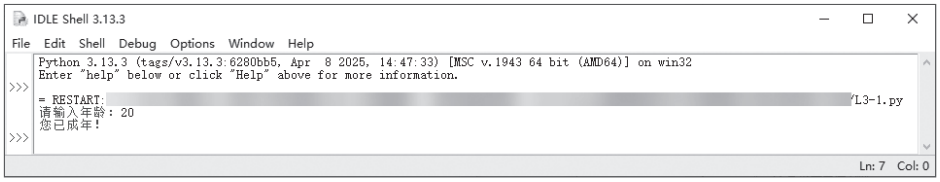

图 3-2

需要注意，if语句后面的“:”不能省略。它表示条件表达式的结束，后续是条件成立的代码块。代码块不需要用大括号括起来，但是必须向右缩进相同的长度。

3.2.2 二分支if-else语句

二分支条件结构通过if-else语句实现，可以根据条件真伪选择执行两个不同的代码分支。其基本语法格式如下：

```
if 条件：
    代码块1
else:
    代码块2
```

如果条件的值为真，则执行代码块1，如果条件的值为假，则执行代码块2，具体流程如图3-3所示。

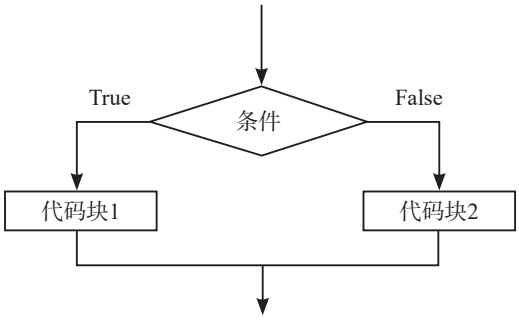

图 3-3

例3-2 双分支条件结构应用示例1

功能描述：使用if-else语句判断用户输入的年龄是否成年，如果成年，则输出“您已成年！”，如果未成年则输出“您未成年！”。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 获取用户输入的年龄并转为整数
age = int(input("请输入您的年龄："))
```

```
# 双分支条件判断
if age >= 18:
    print("您已成年！")
else:
    print("您未成年！")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-4所示。

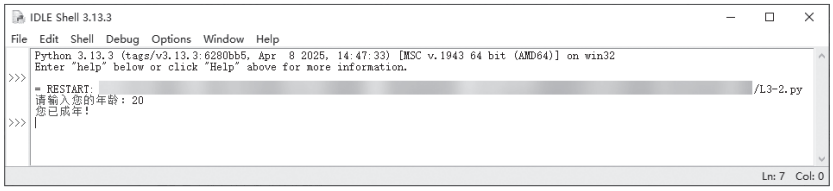

图 3-4

步骤 03 重新执行程序，输入16，执行结果如图3-5所示。

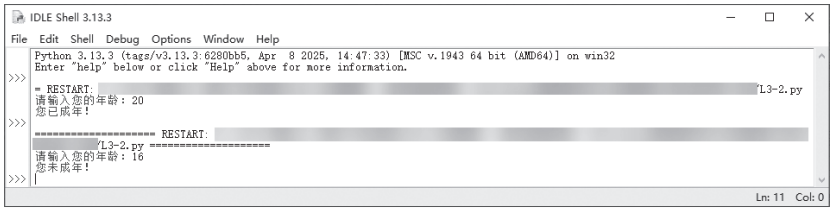

图 3-5

分析程序的两次执行结果，可以发现程序已经可以根据用户输入的年龄分别执行不同的代码块，输出不同的执行结果。

注意：

① if和else要左对齐。

② else后面需要加冒号。

③ else不能单独使用，必须和if一起使用。

3.2.3 多分支if-elif-else语句

多分支条件结构通过if-elif-else语句实现，能够根据多个互斥条件选择不同的执行路径。其基本语法格式如下：

```
if 条件1:
    代码块1
elif 条件2:
    代码块2
elif 条件3:
    代码块3
…
else:
    代码块n
```

如果条件1的值为真，则执行代码块1，然后结束整个分支语句；如果为假，则跳过代码块1，进入elif的判断，如果条件2为真，则执行代码块2，然后结束整个分支语句；以此类推，如果前面所有的条件的值都为假，则执行else后面的代码块n，具体流程如图3-6所示。

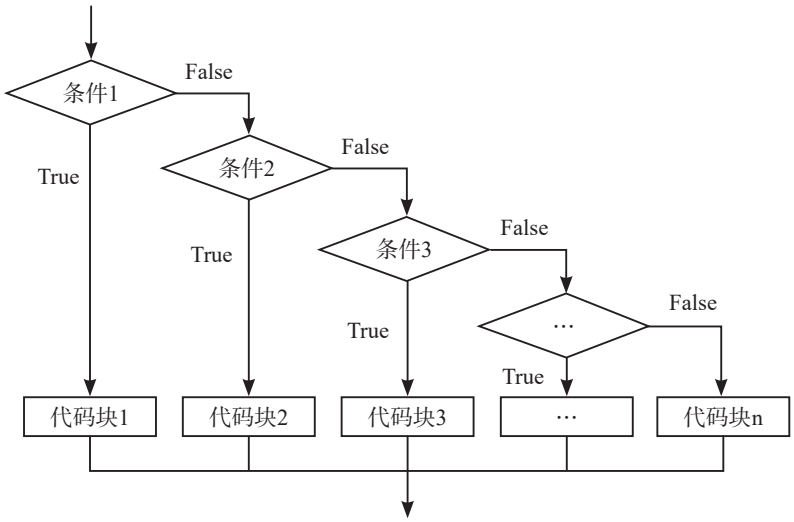

图 3-6

例3-3 多分支条件结构应用示例

功能描述：使用多分支结构判断数字是正数、负数还是0。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 获取用户输入并转为浮点数（兼容整数和小数）
num = float(input("请输入一个数字："))

if num > 0:
    print("正数")
elif num < 0:
    print("负数")
else:
    print("零")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-7所示。

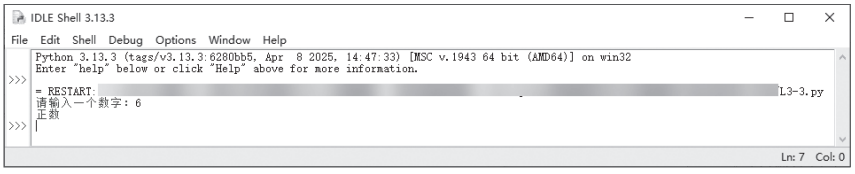

图 3-7

步骤 03 重复执行程序，并输入不同的内容，将得到不同的结果，如图3-8所示。

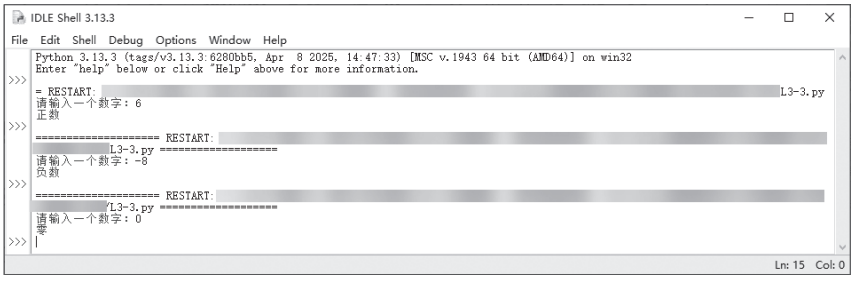

图 3-8

3.2.4 嵌套条件结构

在Python 3中，可以将上述三种条件结构嵌套使用，常用的语法格式如下：

```
if 条件1:
    if 条件2:
        代码块1
    else:
        代码块2
```

若不满足条件1，则直接跳过执行后面的代码；当满足条件1时，再检查是否满足条件2，若满足则执行代码块1，不满足则执行代码块2。

也可以在if-else语句中嵌套if-else语句，语法格式如下：

```
if 条件1:
    if 条件2:
        代码块1
    else:
        代码块2
else:
    if 条件3:
        代码块3
    else:
        代码块4
```

当满足条件1时，检查是否满足条件2，若满足则执行代码块1，不满足则执行代码块2；当不满足条件1时，则检查是否满足条件3，若满足则执行代码块3，不满足则执行代码块4。

例3-4 条件结构嵌套应用示例

功能描述：判断数字是否同时为5的倍数且为正数。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
num = int(input("请输入一个整数："))

# 外层条件：判断是否为正数
if num > 0:
    # 内层条件：判断是否为5的倍数
    if num % 5 == 0:
        print(f"{num} 是正数且为5的倍数")
    else:
        print(f"{num} 是正数，但不是5的倍数")
else:
    print(f"{num} 不是正数")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-9所示。

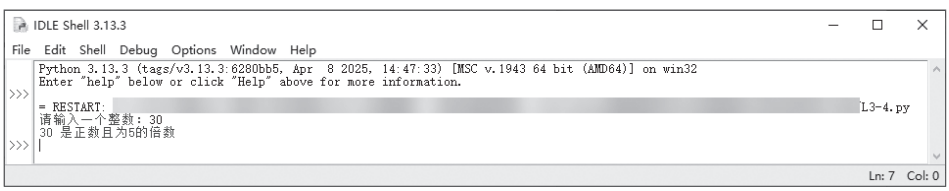

图 3-9

例3-5 条件结构嵌套应用示例

功能描述：判断数字是否为正数，若是则判断是正偶数还是正奇数。若不是正数则判断是负数还是零。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 获取用户输入并转为整数（假设输入为整数）
num = int(input("请输入一个数字："))

# 外层条件：判断是否为正数
if num > 0:
    # 内层条件1：判断正数的奇偶性
    if num % 2 == 0:
        print(f"{num} 是正偶数")
    else:
        print(f"{num} 是正奇数")
else:
    # 内层条件2：判断非正数的情况（负数或零）
    if num == 0:
        print(f"{num} 是零")
    else:
        print(f"{num} 是负数")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-10所示。

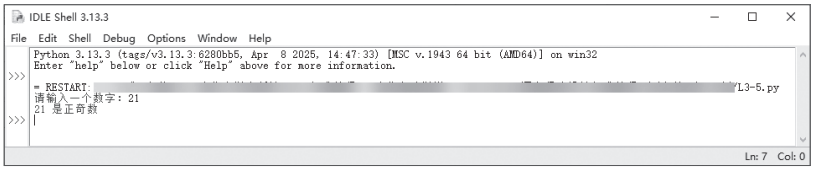

图 3-10

在Python中使用if语句嵌套时，应注意以下4点。

- 保持代码的缩进规范，一般使用4个空格或一个制表符（Tab）。
- 避免过深的嵌套，过深的嵌套会使代码难以阅读和维护。
- 确保每个if语句都有对应的else语句（如果需要的话），以避免逻辑漏洞。
- 使用elif（else if的缩写）检查多个条件，可以减少不必要的嵌套。

例3-6 成绩等级判定

功能描述：通过多分支结构判断输入的成绩等级。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
score = int(input("请输入成绩（0-100）："))

if score < 0 or score > 100:
    print("输入无效！成绩必须在 0-100 之间")
elif score >= 90:
    print("等级：A")
elif score >= 80:
    print("等级：B")
elif score >= 70:
    print("等级：C")
elif score >= 60:
```

```
    print("等级：D")
else:
    print("等级：F（不及格）")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-11所示。

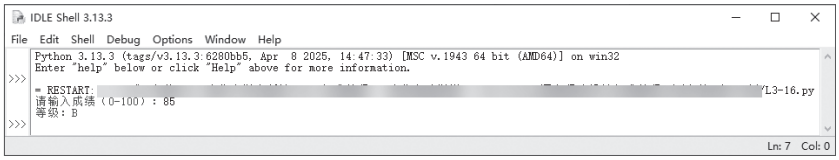

图 3-11

3.3 循环结构

循环结构可以重复执行一段代码，直至满足特定条件。Python语言中常用while和for语句作为循环执行程序。本节对循环结构进行介绍。

3.3.1 while语句

while语句可以在条件表达式为真时重复执行代码块，是实现不确定性循环的主要方式。其语法格式如下：

```
while 条件:
    代码块
```

当表达式的返回值为真时，执行代码块（循环体），然后重新判断表达式的返回值，直到表达式的值为假时结束循环。具体流程如图3-12所示。

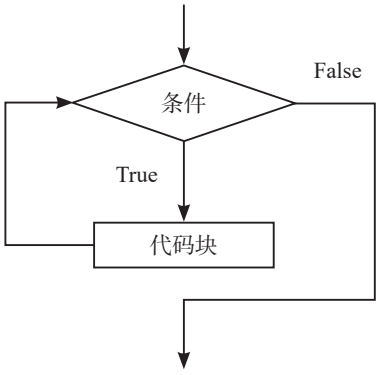

图 3-12

例3-7 while语句应用示例

功能描述：使用while语句计算数字1～10的整数之和。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入如下代码：

```
total = 0                # 初始化总和为0
i = 1                    # 从1开始计数
while i <= 10:           # 当i小于或等于10时循环
    total += i           # 累加当前i的值
    i += 1               # 更新i的值（避免死循环）
print("1到10的和为:", total)  # 输出结果：55
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-13所示。

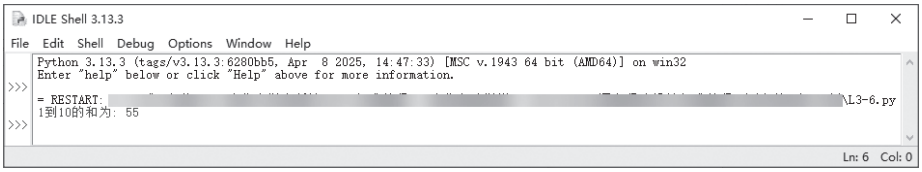

图 3-13

需要注意在编写while循环语句时，一定要保证程序能够正常结束，否则将导致“死循环”，程序一直执行无法退出。例如，下面的代码就是一个死循环。

```
while 1 < 2:
    print("正确")
```

上面代码中，无论循环执行多少次，1永远小于2，循环会一直执行下去，不停输出结果“正确”。

3.3.2 for语句

for语句可以遍历任何可迭代对象（如字符串、列表、元组、字典、集合等）中的元素，并对每个元素执行相应的代码块，通常应用于已知循环次数的情况。其语法格式如下：

```
for 循环变量 in 序列:
    代码块
```

其中，循环变量是一个临时变量，在每次循环迭代时，会依次被赋值为序列中的每个元素，序列是一个可迭代对象，如for语句会按照序列中元素的顺序，依次将每个元素赋值给循环变量，代码块由缩进的语句组成，可以保护各种Python语句。for语句具体流程如图3-14所示。

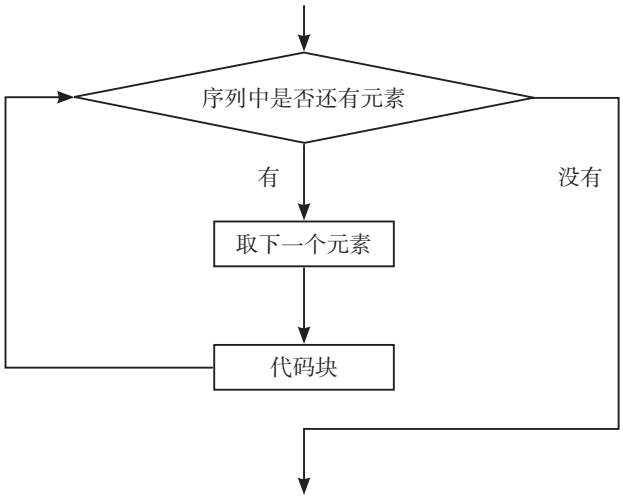

图 3-14

例3-8 for语句应用示例

功能描述：使用for语句计算数字1～10的整数之和。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
total = 0  # 初始化总和为0
for num in range(1, 11):  # range(1,11) 生成1～10的整数
    total += num          # 累加每个数字
print("1到10的和为:", total)  # 输出结果：55
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-15所示。

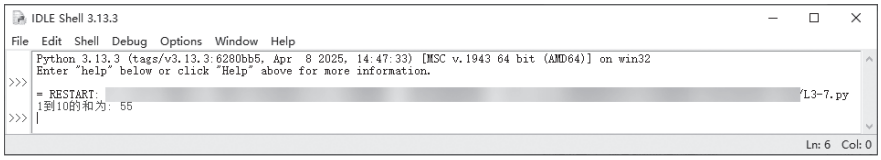

图 3-15

3.3.3 循环嵌套

Python语言允许在一个循环体里嵌入另外一个完整的循环语句，而在内部的这个循环语句中还可以嵌入其他循环语句，这就是所谓的循环嵌套。循环嵌套比较复杂，for和while循环语句中都可以嵌套，它们之间还可以互相嵌套。

1. for循环嵌套

for循环中可以嵌套for循环，它的语法格式如下：

```
for 循环变量1  in 序列1:
    for 循环变量2  in 序列2:
        代码块 2
    代码块1
```

在for循环嵌套中，每执行一次外层的for循环，都要将内层的for循环全部执行一遍。需要注意的是，内层循环是用缩进来表示的。

例3-9 **for循环嵌套示例**

功能描述：使用嵌套for循环输出九九乘法表。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
for i in range(1, 10):           # 外层循环控制行（乘数1）
    for j in range(1, i+1):      # 内层循环控制列（乘数2）
        print(f"{j}×{i}={i*j}", end="\t")  # 用制表符分隔
    print()                      # 换行
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-16所示。

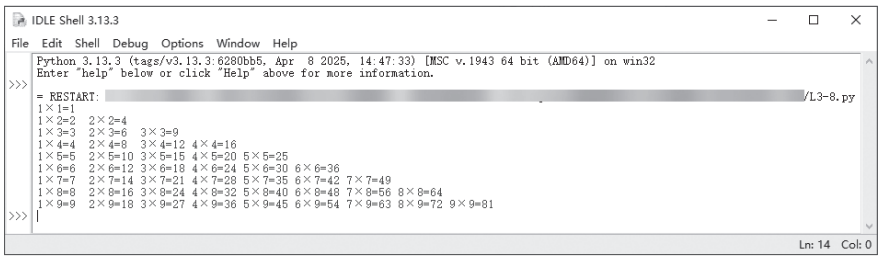

图 3-16

至此，完成for循环嵌套的应用示例。

2. while循环嵌套

while循环中还可以嵌套while循环，语法格式如下：

```
while 条件1:
    while 条件2:
        代码块 2
    代码块1
```

在while循环嵌套中，每执行一次外层的循环，都要将内层的循环全部执行一遍。需要注意内层循环是用缩进来表示的。

例3-10 while循环嵌套示例

功能描述：使用嵌套while循环输出九九乘法表。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
i = 1  # 初始化外层循环变量（控制行数）
while i <= 9:
    j = 1  # 初始化内层循环变量（控制列数）
    while j <= i:  # 确保列数不超过当前行数
        print(f"{j}×{i}={i*j}", end='\t')  # 输出当前乘法项，使用制表符分隔
        j += 1  # 内层循环变量自增
    print()  # 换行
    i += 1  # 外层循环变量自增
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-17所示。

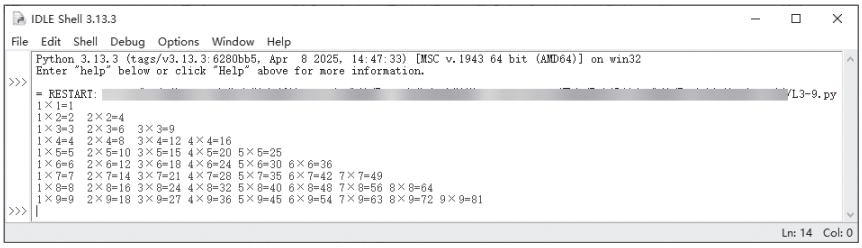

图 3-17

3. 在while循环中嵌套for循环

while循环中还可以嵌套for循环，语法格式如下：

```
while 表达式:
    for 循环变量 in 序列:
        代码块2
    代码块1
```

在while循环中嵌套for循环，意味着每执行一次外层while循环，都要将内层的for循环全部执行一遍。

例3-11 while循环中嵌套for循环示例

功能描述：通过在while循环中嵌套for循环的方式输出九九乘法表。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
i = 1  # 外层循环变量，控制行数（1到9）
while i <= 9:
    # 内层用 for 循环遍历列（1到i）
    for j in range(1, i + 1):
        print(f"{j}×{i}={i*j}", end="\t")  # 输出乘法式，横向对齐
    print()  # 换行
    i += 1   # 更新外层循环变量
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-18所示。

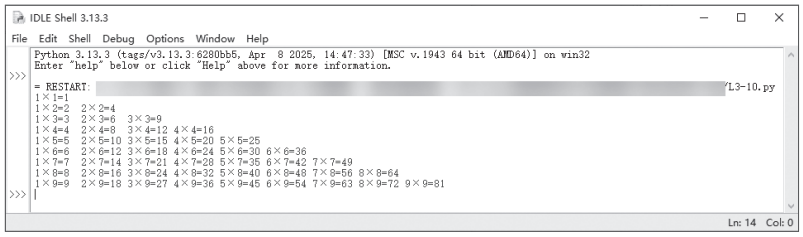

图 3-18

4. 在for循环中嵌套while循环

for循环中也可以嵌套while循环，语法格式如下：

```
for 循环变量 in 序列:
    while 表达式:
        代码块2
    代码块1
```

在for循环中嵌套while循环，意味着每执行一次外层for循环，都要将内层的while循环全部执行一遍。

例3-12 **for循环中嵌套while循环示例**

功能描述：通过在for循环中嵌套while循环的方式输出九九乘法表。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
for i in range(1, 10):          # 外层循环控制行（1到9）
    j = 1                       # 内层循环变量初始化
    while j <= i:               # 内层循环控制列（1到i）
        print(f"{j}×{i}={i*j}", end="\t")
        j += 1                  # 更新内层循环变量
    print()                     # 换行
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-19所示。

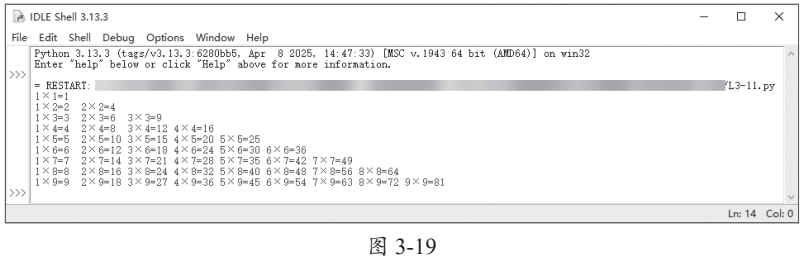

图 3-19

至此，完成for循环中嵌套while循环的应用示例。

例3-13 统计1～20的质数数量

功能描述：通过循环结构统计1～20的质数数量并打印质数。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 定义质数判断函数
def is_prime(n):
    if n <= 1:
        return False  # 1及以下的数不是质数
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# 初始化计数器和质数列表
count = 0
prime_list = []

# 循环遍历1-20之间的数
for num in range(1, 21):
    if is_prime(num):
        count += 1
        prime_list.append(num)

# 输出结果
print(f"1-20之间的质数数量：{count}")
print("质数列表：", prime_list)
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-20所示。

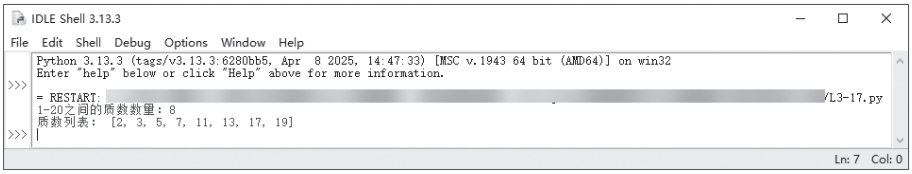

图 3-20

3.4 程序的循环控制

除了达成条件结束循环外，开发人员还可以设定条件跳出循环。常用跳出循环的语句包括break语句和continue语句。

3.4.1 break语句

break语句可以立即终止当前所在的循环，直接跳出循环体，继续执行循环之后的代码。

1. 跳出while循环

使用break跳出while循环的常见形式如下：

```
while 条件1:
```

```
    代码块
    if 条件2:
        break
```

例3-14 使用break跳出while循环示例

功能描述：利用break跳出while循环。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
i = 1
result = 0
while i<10:
    result = result + i
    print("i 的值是: " , i)
    i += 1
    if result > 10:    # 当result > 10时，结束循环
        break
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-21所示。

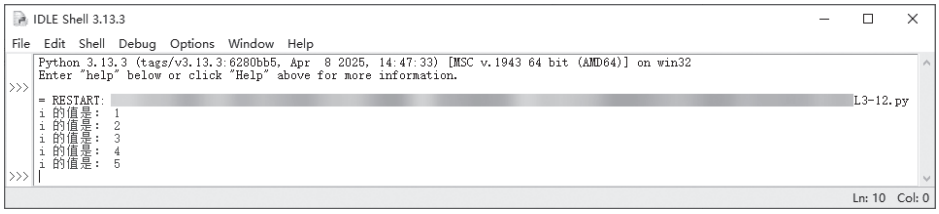

图 3-21

从执行结果可以发现，循环执行5次后就退出循环了。

2. 跳出for循环

使用break跳出for循环的常见形式如下：

```
for 循环变量 in 序列:
    代码块
    if 条件:
        break
```

例3-15 使用break跳出for循环示例

功能描述：利用break跳出for循环。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
for num in [1, 2, 3, 4, 5]:
    print(f"当前数字: {num}")
    if num == 3:
        print("触发 break，终止循环")
        break

print("循环结束")
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-22所示。

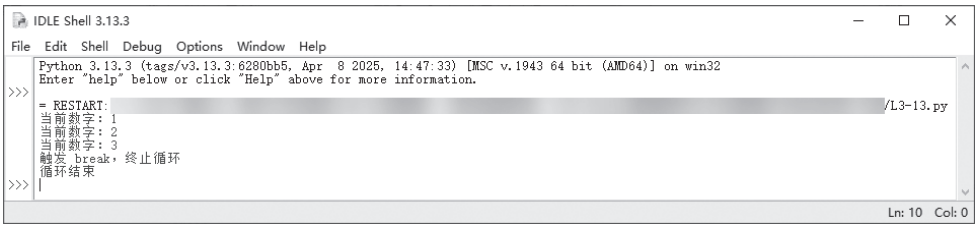

图 3-22

从执行结果可以发现，循环遍历到数字3时就退出循环了。

3.4.2 continue语句

break语句强制终止循环，跳出当前循环体。continue语句则跳过当前迭代的剩余代码，直接进入循环的下一次迭代（不终止循环）。也就是说，continue语句只结束本次循环，并不终止整个循环的执行。

1. 在while循环中使用continue语句

在while循环中使用continue语句的常见形式如下：

```
while 条件1:
    代码块
    if 条件2:
        continue
```

例3-16 在while循环中使用continue语句示例

功能描述：在while循环中使用continue语句跳过奇数的输出。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
i = 0
while i < 10:
    i += 1
    if i % 2 != 0:
        continue
    print(i)
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-23所示。

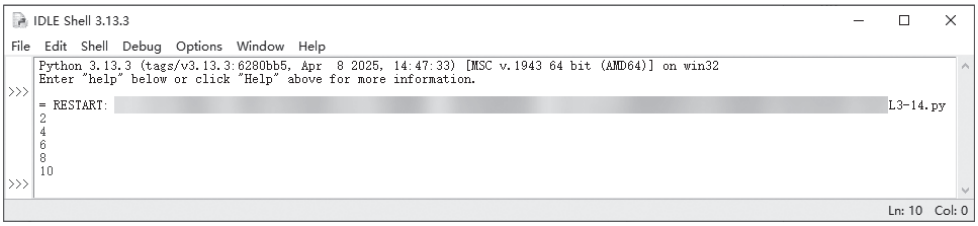

图 3-23

从执行结果可以发现，当循环执行遇到小于10的奇数时，程序跳出本次循环，没有执行打印信息的操作。

2. 在for循环中使用continue语句

在for循环中使用continue语句的常见形式如下：

```
for 循环变量 in 序列:
    代码块
    if 条件:
        continue
```

例3-17 在for循环中使用continue语句示例

功能描述：在for循环中使用continue语句跳过奇数的输出。

步骤 01 打开IDLE，执行File | New File命令打开IDLE的编辑器，输入代码如下：

```
# 遍历 1 到 10 的整数
for i in range(1, 11):
    if i % 2 != 0:
        continue
    print(i)
```

步骤 02 按Ctrl+S组合键保存文件。执行Run | Run Module命令，切换到IDLE Shell窗口，执行当前程序，如图3-24所示。

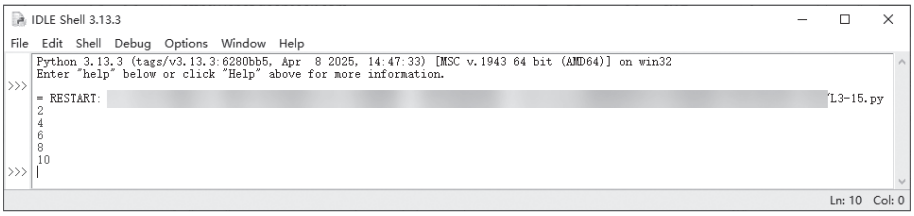

图 3-24

从执行结果可以发现，当循环执行遇到小于10的奇数时，程序跳出本次循环，没有执行打印信息的操作。

3.5 拓展练习

练习1 通过键盘输入一个整数，并判断该整数能否被3整除。

练习2 通过键盘输入成绩，并判断是否及格（成绩范围为0～100，否则输出“输入错误”提示信息）。

练习3 猜数字游戏，通过3次机会猜1～10的任意数字。

扫码查看参考答案