# 实验 3　Python控制语句与程序调试

## 学习目标

- 了解条件控制语句。
- 了解循环控制语句。
- 了解循环控制语句中的辅助语句。
- 了解程序调试步骤。

本实验将要涉及的 Python 控制语句与程序调试框架如图 3.1 所示。

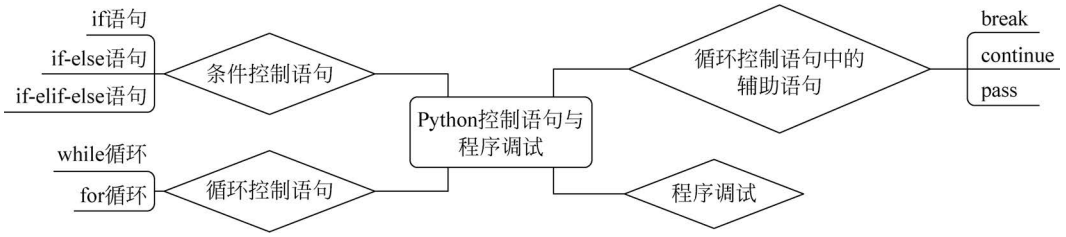


图 3.1　实验 3 知识框架图

## 3.1　控制语句


Python 中的控制语句是用来控制程序流程的关键结构，具体内容如表 3.1 所示。

**表 3.1　Python 控制语句语法与举例**

| 名　称 | 语　法 | 举　例 |
|---|---|---|
| 条件控制语句：<br>if 语句 | if 语句：用于基于单个条件执行相应代码块<br>if condition：<br>　　statement(s) | age = 21<br>if age >= 18：<br>　　print("成年人")<br># 如果 age 变量的值大于或等于 18，<br># 则会输出"成年人" |
| 条件控制语句：<br>if-else 语句 | if-else 语句：除了满足条件执行语句外，还可以在条件不满足时执行另一段代码<br>if condition：<br>　　statement(s)<br>else：<br>　　alternative_statement(s) | score = 85<br>if score >= 90：<br>　　print("优秀")<br>else：<br>　　print("一般") |
| 条件控制语句：<br>if-elif-else | if-elif-else 结构：用于检查多个条件，并执行第一个条件为真的相关代码块<br>if condition1：<br>　　statement(s)<br>elif condition2：<br>　　other_statement(s)<br>else：<br>　　default_statement(s) | score = 85<br>if score >= 90：<br>　　print("优秀")<br>elif score >= 60：<br>　　print("及格")<br>else：<br>　　print("不及格") |

续表

| 名　　称 | 语　　法 | 举　　例 |
| --- | --- | --- |
| 循环控制语句：while 循环 | while 循环：只要给定的条件为真，就会重复执行一个代码块<br>while condition：<br>　　statement(s) | #打印从 1 到 10 的整数，使用 while 循环和条件判断<br>counter = 1<br>while counter <= 10：<br>　　print(counter)<br>　　counter += 1 |
| 循环控制语句：for 循环 | for 循环主要用于遍历序列(如列表、元组、字符串)或其他可迭代对象，常与成员函数 in 联合使用<br>for variable in iterable：<br>　　statement(s) | #遍历列表并打印每个元素<br>numbers = [1, 2, 3, 4, 5]<br>for number in numbers：<br>　　print(number) |
| 循环控制语句中的辅助语句：break | break 出现在程序中，程序将立即跳出当前循环，不再执行循环体中剩余的语句 | for i in range(10)：<br>　　if i == 5：<br>　　　　break<br>　　　　print(i)<br>#输出结果：0 1 2 3 4 |
| 循环控制语句中的辅助语句：continue | 当 continue 语句被执行时，程序将跳过当前循环体中 continue 之后的所有语句，直接进入下一轮循环的迭代 | for i in range(10)：<br>　　if i % 2 == 0：#如果 i 是偶数<br>　　　　continue<br>　　　　print(i)<br>#输出结果：1 3 5 7 9 |
| 循环控制语句中的辅助语句：pass | 程序什么都不做 | for i in range(10)：<br>　　if i % 2 == 0：<br>　　　　pass　#对于偶数，目前不执行任何操作 |

**【例 3.1】** 从键盘输入成绩，并给出评分(小于 60 分为不及格；60～90 分为合格；90 分以上为优秀)。

```
score = input("请输入分数:")
score = eval(score) #转换为数值型
if score>=90:
   print("你输入的分值为{},评语为:优秀".format(score))
elif(score>=60):
   print("你输入的分值为{},评语为:合格".format(score))
else:
   print("你输入的分值为{},评语为:不及格".format(score))
```

**【例 3.2】** 分别使用 while 循环和 for 循环，求 100 以内所有整数的和。

```
#while 循环实现
sum = 0                          #需要一个变量求和
num = 0                          #需要一个变量作为加数
while(num<=100):
  sum += num
  num += 1                       #一定要改变 num 的值,否则会进入死循环
print("100 以内所有整数的和为:{}".format(sum))
#for 循环实现
sum=0     #重新初始化 sum 变量,否则 for 循环结果为 10100
for x in range(101):#range 生成公差为 1 的等差数列,但不包括 101
```

```
    sum += x
print("100 以内所有整数的和为:{}".format(sum))
```

程序执行结果如下。

```
100 以内所有整数的和为: 5050
100 以内所有整数的和为: 5050
```

## 3.2　程序调试

程序出错一般分为语法错误和逻辑错误，如果是语法错误，编译器会直接提醒，相对比较简单。如果是逻辑错误，编译器不会报警，但是程序执行结果和程序员想要的结果有差异，这个时候就需要用到程序调试。由于 Jupyter Notebook 需要依赖第三方软件才能实现可视化程序调试，因此本节介绍如何使用 Spyder 调试程序，如果装了 Anaconda，Spyder 也会自动安装。

使用 Spyder 调试程序的步骤如图 3.2 所示。

**（1）设置断点，打开 Python 脚本文件**。在想要暂停执行的代码行左侧单击，设置一个断点。在该行的左侧会出现一个红点，表示已成功设置断点。

**（2）启动调试器**。在 Spyder 的顶部菜单中，选择 Run→Debug file，或者直接单击工具栏上的绿色虫子图标（通常带有“Debug”字样），将启动调试会话，并在到达第一个断点时暂停执行。

**（3）控制执行流程**。一旦程序暂停在断点处，就可以使用调试工具栏中的按钮来控制执行流程。这些按钮通常包括 Step into（进入函数内部）、Step over（执行当前行但不进入函数）、Step out（跳出当前函数）以及 Continue（继续执行直到下一个断点）。

**（4）查看变量值**。在变量查看器（Variable Explorer）窗口中，可以看到当前作用域内所有变量的值。也可以在调试控制台（Debug Console）中直接输入变量名来查看其值。

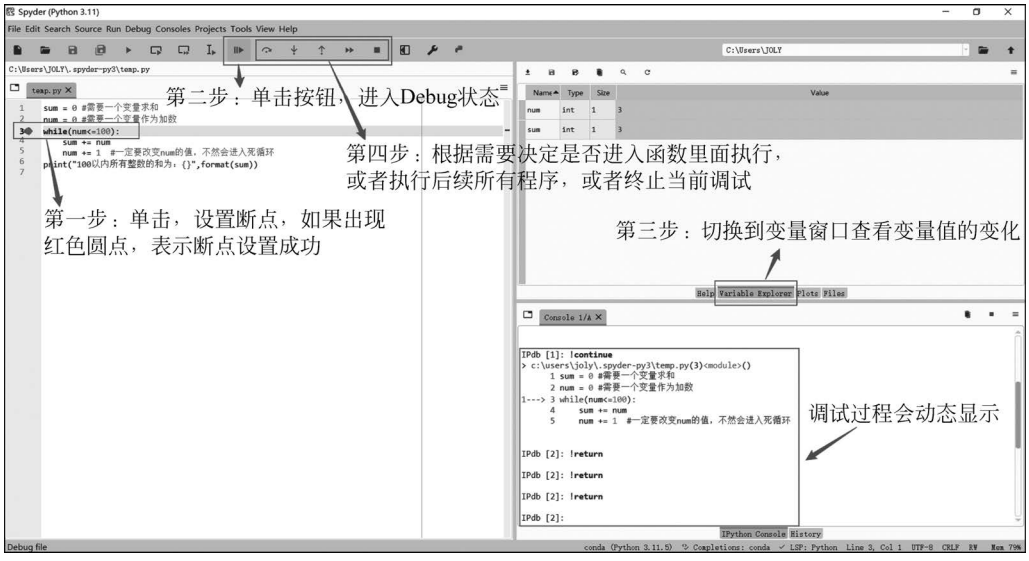


图 3.2　Spyder 调试程序的步骤

# 实验作业 3

1. 计算 100 以内(包括 100)所有偶数的和。
2. 用 Spyder 单步执行,查看每一步的变化。