任务三

Python程序的控制结构——超市购物

3.1 任务说明

3.1.1 任务描述

本任务实现超市购物活动，在一家超市有牙刷、毛巾、水杯、苹果和香蕉等商品，商品价格如表 3-1 所示。

表 3-1 某超市商品价格

编　　号	商 品 名 称	价　　格
1	牙刷	8.8 元
2	毛巾	10.0 元
3	水杯	18.8 元
4	苹果	12.5 元
5	香蕉	15.5 元

用户输入商品序列号进行商品购买，用户输入购买数量后计算出所需要花费的金额，一次购买结束后，需要用户输入“Y/y”或“N/n”，“Y/y”代表继续购买，“N/n”代表购物结束。

3.1.2 任务目标

知识目标

1. 理解程序的控制结构
2. 掌握单分支、双分支和多分支结构的用法
3. 掌握循环语句的用法
4. 掌握 break、continue、pass 和 else 语句的作用

能力目标

1. 学会分析“超市购物”任务实现的逻辑思路
2. 能够使用分支语句编写程序
3. 能够使用循环语句编写程序
4. 能够完成“超市购物”程序的代码编写
5. 会调试简单的程序错误

素养目标

1. 培养学生对复杂问题的逻辑判断能力,使学生具备发现问题解决问题、理解数据处理逻辑的基本素养
2. 通过动手编写分支和循环程序,解决实际问题,培养学生解决复杂问题的能力

3.2 任务相关知识

3.2.1 程序的三种控制结构

程序是一个语句序列,执行程序就是按特定的次序执行程序中的语句。程序中执行点的变迁称为控制流程,当执行到程序中的某一条语句时,也可以说控制转到了该语句。由于复杂问题的解法可能涉及复杂的执行次序,因此编程语言必须提供表达复杂控制流程的手段,它被称为编程语言的控制结构,或程序控制结构。任何简单或复杂的算法都可以由顺序结构、选择(分支)结构、循环结构这三种基本结构组合而成,所以这三种结构就是程序设计的基本结构。

1) 流程图

流程图又称程序框图,它用统一规定的标准符号描述程序运行具体步骤。程序框图的设计是在处理流程图的基础上,通过对输入输出数据和处理过程的详细分析,将计算机的主要运行步骤和内容标识出来。流程图是进行程序设计最基本的依据,因此它的质量直接关系到程序设计的质量。流程图直观、清晰,更有利于人们设计与理解算法。所以流程图是算法的一种图形化表示方式。

流程图使用一组预定义的符号来说明如何执行特定任务,基本要素为表示相应操作的框,带箭头的流程线,框内外必要的文字说明。

流程图采用的符号如图 3-1 所示。

2) 顺序结构

顺序结构程序就是指按语句出现的先后顺序执行的程序结构,是结构化程序中最简单的结构。编程语言并不提供专门的控制流语句来表达顺序控制结构,而是用程序语句的自然排列顺序来表达。计算机按此顺序逐条执行语句,当一条语句执行完毕,控制自动转到下一条语句。现实世界中这种顺序处理的情况是非常普遍的,例如,我们接受学校教育一般都是先上小学,然后上中学,最后上大学;我们烧菜一般都是先热油锅,然后将蔬菜入锅翻炒,

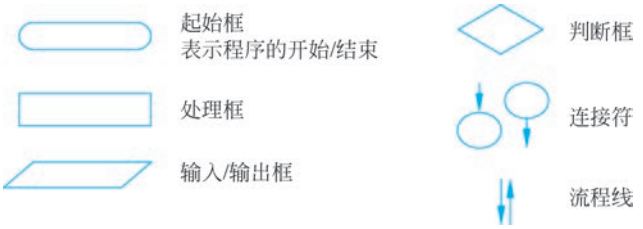

图 3-1　流程图符号

再加盐等佐料，最后装盘。

3）选择结构

选择结构又称为分支结构。当程序执行到控制分支的语句时，首先判断条件，根据条件表达式的值选择相应的语句执行，同时放弃另一部分语句的执行。分支结构包括单分支、双分支和多分支三种形式。

4）循环结构

采用循环结构可以实现有规律的重复计算处理。当程序执行到循环控制语句时，根据循环判定条件对一组语句重复执行多次。

3.2.2　条件表达式

表达式是运算符和操作数所构成的序列，条件表达式是由比较运算符或者逻辑运算符和操作符构成的序列。接下来详细介绍条件表达式中使用的比较运算符和逻辑运算符。

1）比较运算符

比较运算符用于比较两个数，其返回的结果只能是 True 或 False。表 3-2 列举了 Python 中的比较运算符。

表 3-2　Python 中的比较运算符

运算符	描　　述	实　　例
==	检查两个操作数的值是否相等，如果是，则条件成立	如 a=3，b=3，则(a==b)为 True
!=	检查两个操作数的值是否不相等，如果是，则条件成立	如 a=1，b=3，则(a!=b)为 True
>	检查左操作数的值是否大于右操作数的值，如果是，则条件成立	如 a=7，b=3，则(a>b)为 True
<	检查左操作数的值是否小于右操作数的值，如果是，则条件成立	如 a=7，b=3，则(a<b)为 False
>=	检查左操作数的值是否大于或等于右操作数的值，如果是，则条件成立	如 a=3，b=3，则(a>=b)为 True
<=	检查左操作数的值是否小于等于右操作数的值，如果是，则条件成立	如 a=3，b=3，则(a<=b)为 True

比较运算符使用示例如下。

```
num1 = 10
num2 = 10
num3 = 20
print('num1 == num2 的比较结果为:',num1 == num2)    # 输出表达式 num1 == num2 的值
print('num1!= num2 的比较结果为:',num1!= num2)      # 输出表达式 num1!= num2 的值
print('num1 > num3 的比较结果为:',num1 > num3)      # 输出表达式 num1 > num3 的值
print('num1 < num3 的比较结果为:',num1 < num3)      # 输出表达式 num1 < num3 的值
print('num3 >= 20 的比较结果为:',num3 >= 20)        # 输出表达式 num3 >= 20 的值
print('num3 <= 20 的比较结果为:',num3 <= 20)        # 输出表达式 num3 <= 20 的值
```

运行结果如下。

```
num1 == num2 的比较结果为: True
num1!= num2 的比较结果为: False
num1 > num3 的比较结果为: False
num1 < num3 的比较结果为: True
num3 >= 20 的比较结果为: True
num3 <= 20 的比较结果为: True
```

字符串之间的比较示例如下。

```
# 字符串之间的比较
s1 = 'abc'
s2 = 'abc'
s3 = 'abcd'
s4 = ' abc'      # abc 前有一空格
s5 = 'hello'
s6 = 'abc '      # abc 后有一空格
print('s1 == s2 的比较结果为:',s1 == s2)
print('s1 == s4 的比较结果为:',s1 == s4)
print('s1 > s4 的比较结果为:',s1 > s4)
print('s1 > s6 的比较结果为:',s1 > s6)
print('s2 > s3 的比较结果为:',s2 > s3)
```

运行结果如下。

```
s1 == s2 的比较结果为: True
s1 == s4 的比较结果为: False
s1 > s4 的比较结果为: True
s1 > s6 的比较结果为: False
s2 > s3 的比较结果为: False
```

说明: 字符串之间按照字符的 ASCII 码进行比较,字符 a 的 ASCII 码比空格的 ASCII 码大,所以 s1>s4 的比较结果为 True。

2) 逻辑运算符

逻辑运算符用来表示日常交流中的“并且”“或者”“取反”等思想。Python 支持逻辑运算符,表 3-3 列举了 Python 中的逻辑运算符。

表 3-3　Python 中的逻辑运算符

运算符	描　　述
and	and 运算符可以对符号两侧的值进行与运算，只有在符号两侧的值都为 True 时，才会返回 True；只要有一个为 False，就返回 False
or	or 运算符可以对符号两侧的值进行或运算，只有在符号两侧的值都为 False 时，才会返回 False；只要有一个为 True，就返回 True
not	not 运算符可以对符号右侧的值进行非运算。对于布尔值，非运算会对其进行取反操作，即 True 变为 False，False 变为 True；对于非布尔值，非运算会先将其转换为布尔值，然后再取反

逻辑运算符使用示例如下。

```
# 与运算
result1 = 1 == 1 and 2 > 1    # 将逻辑表达式"1 == 1 and 2 > 1"的值赋值给 result1
result2 = 1 == 1 and 2 < 1
print('result1 的结果为：%s, result2 的结果为：%s '%(result1, result2))
# 或运算
result3 = 1 == 1 or 2 < 1     # 将逻辑表达式"1 == 1 and 2 < 1"的值赋值给 result3
result4 = 2 == 1 or 2 < 1
print('result3 的结果为：%s, result4 的结果为：%s '%(result3, result4))
# 非运算
result5 = not 1 == 1          # 将逻辑表达式"not 1 == 1"的值赋值给 result3
result6 = not 2 < 1
print('result5 的结果为：%s, result6 的结果为：%s '%(result5, result6))
```

运行结果如下。

```
result1 的结果为:True, result2 的结果为:False
result3 的结果为:True, result4 的结果为:False
result5 的结果为:False, result6 的结果为:True
```

说明：逻辑运算符对符号两侧的表达式进行不同的逻辑运算，例如，表达式 1==1 and 2>1，先计算 1==1 表达式的结果为 True，2>1 表达式的结果为 True，然后二者进行与运算后结果为 True。

在数学中构成三角形的条件是任意两边之和大于第三边，假设这三条边用 x，y，z 表示，在程序中这个条件的表示如下。

```
x + y > z and z + x > y and y + z > x
```

3.2.3　程序的选择结构

选择结构就是指满足某些条件才允许执行对应语句，不满足条件时直接跳过。例如，用户登录时，只有用户名和密码全部正确，才能登录成功，跳转到应用界面。Python 中提供了多种判断语句实现选择功能。接下来一一讲解这些判断语句。

1）单分支 if 语句

单分支结构是最简单的一种选择结构，其语法格式如下所示。

```
if 条件表达式:
    语句块
```

语法说明:

(1) 条件表达式后面的":"是不可缺少的,它表示一个语句块的开始,后面的几种形式的选择结构和循环结构中的":"也都是不可缺少的。

(2) 在 Python 语言中代码的缩进非常重要。缩进是体现代码逻辑关系的重要方式,所以在编写语句块的时候,务必注意代码缩进。同一个代码块必须保证相同的缩进量。

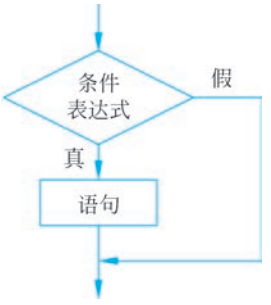

图 3-2 if 语句的执行流程

上述语法中,先计算条件表达式的值,当条件表达式的值为 True 的时候,语句块将被执行;如果条件表达式不成立,语句块就不会被执行,程序会继续执行后面的语句(如果有),执行流程如图 3-2 所示。在这里,语句块有可能被执行,也有可能不被执行,是否执行取决于条件表达式的判断结果。

例 3-1:用户输入任意两个整数 num1 和 num2,通过比较大小保证输出的 num1 是较大者。如果输入的两个数中 num1>num2,直接输出即可,如果 num1<num2,则交换后输出。实现代码如下。

```
num1 = int(input("请输入第一个数:"))
num2 = int(input("请输入第二个数:"))
print("交换前输入的值:num1 = %d" %num1,",num2 = %d" %num2)
if num1 < num2:
      num1,num2 = num2,num1    # 实现交换
print("交换后输出的值:num1 = %d" % num1,",num2 = %d" %num2)
```

运行结果如下。

```
请输入第一个数:23
请输入第二个数:56
交换前输入的值:num1 = 23 ,num2 = 56
交换后输出的值:num1 = 56 ,num2 = 23
```

从结果可以看出输入的第一个数比第二个数小,满足条件,因此执行两个数的交换;如果输入的第一个数比第二个数大,不满足条件,则两数交换的语句不会执行。运行结果如下。

```
请输入第一个数:89
请输入第二个数:26
交换前输入的值:num1 = 89 ,num2 = 26
交换后输出的值:num1 = 89 ,num2 = 26
```


if…else 语句

2) 双分支 if…else 语句

有时候不仅要考虑条件满足的情况,同时也要处理条件不满足的情况,这时就需要双分支结构。Python 使用关键字 if-else 实现双分支条件控制,语法格式如下所示。

```
if 判断条件:
    语句块 1
else:
    语句块 2
```

判断条件成立时执行语句块 1，当判断条件不成立时执行 else 语句下的语句块 2，执行流程如图 3-3 所示。

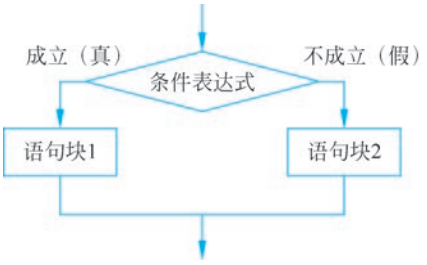

图 3-3　if-else 语句的执行流程

例 3-2：超市促销，满 100 减 20，小敏去超市购物，编程实现小敏应付金额。实现代码如下。

```
# 超市促销
money = int(input("请输入购物金额:"))
# 计算应付款
if money >= 100:
    pay = money - 20
else:
    pay = money
# 输出付款金额
print("您需要支付:%d元" % pay)
```

如果输入金额为 130 元，则运行结果如下。

```
请输入购物金额:130
您需要支付:110元
```

如果输入金额为 79 元，则运行结果如下。

```
请输入购物金额:79
您需要支付:79元
```

从结果可以看出，如果条件满足才会执行减 20 元的语句块，否则应付款就等于购买商品的金额。

例 3-3：输入任意一个年份，判断是否为闰年。

分析：什么是闰年？年份能被 4 整除且不能被 100 整除的为闰年，或者能被 400 整除的是闰年。

定义年份变量为 year，如果条件 year % 4 == 0 和 year % 100!=0 同时满足为闰年，或者满足条件 year % 400==0 的是闰年，需用逻辑运算符表示条件。当条件表达式需要多个条件同时判断时，使用 or（或）表示两个条件中只要有一个成立即为真；使用 and

(与)表示两个条件同时成立判断条件才为真,可连续使用 and 和 or 联立多个条件表达式。实现代码如下。

```
# 判断闰年
year = int(input("请输入一个年份:"))    # 将键盘输入转为整型
if (year % 4) == 0 and (year % 100) != 0 or (year % 400) == 0:
     print("{0}年是闰年".format(year))
else:
     print("{0}年不是闰年".format(year))
```

运行程序,从键盘输入 2022,运行结果如下。

```
请输入一个年份:2022
2022 年不是闰年
```

再次运行程序,从键盘输入 2020,运行结果如下。

```
请输入一个年份:2020
2020 年是闰年
```


多分支 if…elif 语句

3) 多分支 if…elif 语句

根据一个条件的结果控制一段代码块的执行用单分支 if 语句,若条件失败时执行另一代码块用 else 语句。如果需要检查多个条件,并在不同条件下执行不同代码块,就要使用多分支 elif 子句,它是具有条件判断功能的 else 子句,相当于 else if。多分支结构的语法形式如下。

```
if 条件表达式 1:
     执行语句块 1
elif 条件表达式 2:
     执行语句块 2
elif 条件表达式 3:
     执行语句块 3
……
else:
    执行语句块 n
```

语法说明:

语法中 if 必须和 elif 配合使用,elif 可以有 0 个、1 个或者多个,else 最多只能有一个,也可以没有。

执行过程如下:

(1) 当满足条件表达式 1 时,执行语句块 1,然后整个 if 结束;

(2) 如果条件表达式 1 不满足,那么判断是否满足条件表达式 2,满足则执行语句块 2,然后整个 if 结束;

(3) 如果条件表达式 1 和 2 都不满足,那么判断是否满足条件表达式 3,满足则执行语句块 3,然后整个 if 结束;

(4) 以此类推,当所有条件都不满足时,执行 else 之后的语句块;

if…elif 语句的执行流程如图 3-4 所示。

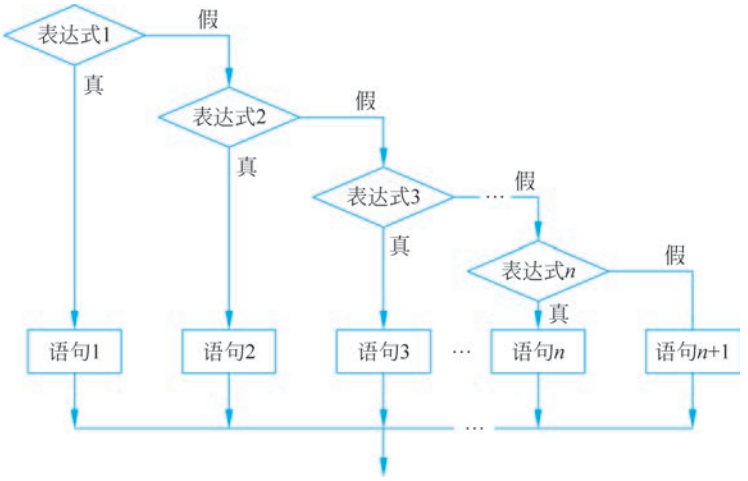

图 3-4　if…elif 语句的执行流程

例 3-4：某超市为了促销，采用购物打折的办法。1000 元及以上，按九五折优惠；2000 元及以上，按九折优惠；3000 元及以上，按八五折优惠；5000 元及以上，按八折优惠。编写程序，输入购物款数，计算并输出优惠价。实现代码如下。

```
# 超市购物打折
a = input("输入购物款数:")
a = int(a) # 将键盘输入转为整型
if a >= 5000:
    print("优惠价为八折:",a * 0.8,"元")
elif a >= 3000:
    print("优惠价为八五折:",a * 0.85,"元")
elif a >= 2000:
    print("优惠价为九折:",a * 0.9,"元")
elif a >= 1000:
    print("优惠价为九五折:",a * 0.95,"元")
else:
    print("没有优惠哦! 价格为:",a,"元")
```

运行程序，从键盘输入 3800，运行结果如下。

```
输入购物款数:3800
优惠价为八五折: 3230.0 元
```

再次运行程序，输入 230，运行结果如下。

```
输入购物款数:230
没有优惠哦! 价格为: 230 元
```

例 3-5：根据用户的身高和体重，计算用户的 BMI 指数，并给出相应的健康建议。BMI 指数，即身体质量指数，是用体重(kg)除以身高(m)平方的得数，BMI 是目前国际上常用的衡量人体胖瘦程度以及是否健康的标准。标准的数值如下所示。

```
过轻:低于 18.5
正常:18.5～23.9
过重:24～27.9
肥胖:28～32
过于肥胖:32 以上
```

实现代码如下。

```
# 计算 BMI 指数
height = float(input("请输入您的身高(m):"))      # 将键盘输入转换为浮点数
weight = float(input("请输入您的体重(kg):"))
BMI = weight/height/height # 计算 BMI 指数
print("您的 BMI 指数是: {:.1f}".format(BMI))      # 格式化输出,保留一位小数
if BMI < 18.5:
     print("您的体形偏瘦,要多吃多运动哦!")
elif 18.5 <= BMI < 24:
      print("您的体型正常,继续保持呦!")
elif 24 <= BMI < 28:
      print("您的体形偏胖,有发福迹象!")
elif 28 <= BMI < 32:
      print("不要悲伤,您是个迷人的胖子!")
else:
      print("什么也别说了,……")
```

运行程序,输入身高 1.8m,体重 82 kg,运行结果如下。

```
请输入您的身高(m):1.80
请输入您的体重(kg):82
您的 BMI 指数是: 25.3
您的体形偏胖,有发福迹象!
```

思考:如果将第 2 个条件“elif 18.5 <= BMI<24:”变为“elif BMI<24:”是否可行?为什么?修改代码中条件表达式,效果是一样的,代码如下。

```
# 计算 BMI 指数
height = float(input("请输入您的身高(m):"))
weight = float(input("请输入您的体重(kg):"))
BMI = weight/height/height
print("您的 BMI 指数是: {:.1f}".format(BMI))
if BMI < 18.5:
     print("您的体形偏瘦,要多吃多运动哦!")
elif BMI < 24:
      print("您的体型正常,继续保持呦!")
elif BMI < 28:
      print("您的体形偏胖,有发福迹象!")
elif BMI < 32:
      print("不要悲伤,您是个迷人的胖子!")
else:
      print("什么也别说了,……")
```

条件表达式需要多个条件同时判断时,还可以用下列方法。

```
if BMI<18.5:
     print("您的体形偏瘦,要多吃多运动哦!")
elif 18.5 <= BMI and BMI<24:
      print("您的体型正常,继续保持呦!")
elif 24 <= BMI and BMI< 28:
      print("您的体形偏胖,有发福迹象!")
elif 28 <= BMI and BMI<32:
      print("不要悲伤,您是个迷人的胖子!")
else:
      print("什么也别说了,……")
```

4）if 语句嵌套


if 语句嵌套

在 if 选择结构中,语句块本身也可以是一段 if 语句,就形成了 if 语句的嵌套结构

例 3-6：使用键盘输入一个三位数的正整数,输出其中的最大的一位数。例如,输入 386,输出 8；输入 290,输出 9。

可以将此问题分解成两步,第一步,需要从用户输入的三位数中分离出百位数、十位数和个位数；第二步,从百位数、十位数和个位数中找最大的一个数字。实现代码如下。

```
# 输出一个三位正整数中最大的一位数字

num = int(input("请输入一个三位正整数:"))
a = num//100          #取 num 的百位数字
b = num//10 % 10      #取 num 的十位数字
c = num % 10          #取 num 的个位数字

if a>b:
     if a>c:
        max_num = a
     else:
        max_num = c
else:
     if b>c:
        max_num = b
     else:max_num = c
print("%d 中的最大数字是:%d" % (num,max_num))
```

运行结果如下。

```
请输入一个三位正整数:968
968 中的最大数字是:9
```

此程序采用了 if 结构的嵌套,外层 if 和 else 分支中的语句块都是由一组内层 if 结构组成的。

当然,从三位整数中找最大的数字,当我们学到函数时也可以用 Python 语言的内置函数 max()来解决,对应语句为 max_num=max(a,b,c)。本程序这样写是为了讲解 if 嵌套语句。求一个三位数中的最大数字,本身并不是一个复杂的问题,但解决这个问题的种种尝试和实现方法却体现了程序设计的一些重要思想：绝大多数的计算问题,都有多种解决方法。这就意味着遇到问题时,不要急于编写出我们脑海中的第一个想法,我们的目的是要找

到一个正确的算法,之后力求清晰、高效,让代码变得赏心悦目,让阅读和维护代码变得简单、轻松。

5) pass 语句

pass 是空语句。当暂时没有确定如何实现功能,或者为以后软件升级预留功能,一般用 pass 来"占位"。可以用在类、函数、选择结构、循环结构中,示例代码如下。

```
if a<b:
      pass          # 什么操作也不做
else:
      z = a
```

比如输入一个表示季度的数字,如果输入不是 1～4 则打印输入错误,否则什么都不做,代码如下。

```
# 输入一个季度,如果输入不是 1～4 则打印输入错误,否则什么都不做
n = int(input("输入一个季度(1～4):"))
if 1<=n<=4:
      pass
else:
     print("您的输入有错!")
```

3.2.4 程序的循环结构

在实际生活中,有不少问题是规律性地重复,例如,从周一到周日,周日过后又是周一,这就是一个循环;一年有四季春夏秋冬,循环往复。在程序中解决此类问题就需要重复执行某些语句。一组被重复执行的语句称为循环体,能否继续重复,取决于循环的终止条件。循环结构是在一定条件下反复执行某段程序的流程结构,被反复执行的程序称为循环体。循环语句是由循环体及循环的终止条件两部分组成的。Python 的循环结构包括 for 循环和 while 循环,接下来详细讲解这两种循环。

1) range 函数

在讲解 Python 循环语句之前,先介绍 range 函数。迭代一个范围内的数字是十分常见的操作,Python 提供了一个内置的函数 range 函数,它可以返回包含一个范围内的数值的数组,经常用在 for 语句中,生成遍历序列。range 函数有以下几种不同的调用方法。

(1) range(n)。

range(n)得到的迭代序列为:0,1,2,3,…,n－1。例如,range(100)表示序 0,1,2,3,…,99。

(2) range(m,n)。

range(m,n)得到的迭代序列为:m,m＋1,m＋2,…,n－1。例如,range(11,16)表示序列 11,12,13,14,15。

(3) range(m,n,d)。

range(m,n,d)得到的迭代序列为:m,m＋d,m＋2d,…,按步长值 d 递增,如果 d 为负则递减,直至那个最接近但不包括 n 的等差值。因此,range(11,16,2)表示序列:11,13,15;range(15,4,－3)表示序列:15,12,9,6。这里的 d 可以是正整数,也可以是负数,正整

数表示增量，而负数表示减量，也有可能出现空序列的情况。

如果 range()产生的序列为空，那么用这样的迭代器控制 for 循环的时候，其循环体将一次也不执行，循环立即结束。

2）for 语句


for 语句

for 循环是一种遍历控制循环，通过遍历的当前情况来控制循环。for 循环可以遍历任何序列的项目。语法格式如下。

```
for 循环变量 in 遍历序列:
    语句块 1
[else:
    语句块 2]
```

语法说明：

（1）for、in 和 else 都是关键字。关键字 for 开始的行是循环的控制结构，它控制 for 中语句块 1 的执行次数，for 中的语句块 1 称为循环体。要注意的是，for 中的语句块需要缩进，以表示其是 for 中包含的内容，缩进量通常为 4 字符。

（2）遍历序列也称迭代器，迭代器是 Python 语言中的重要机制之一，一个迭代器是一个值序列，或值集合。循环过程中，循环变量依次从迭代器中取值，并对取得的每个值执行 for 循环体的代码。迭代器中的值的个数就是 for 循环的次数。当循环变量取完迭代器中的所有值后，循环将结束。如果遍历序列为空，则循环体一次也不执行。

（3）中括号中的内容表示可选项，else 必须和 for 配对使用，不能单独使用。

执行过程：

先判断遍历序列中是否有未遍历的元素，若有，将该序列中第一个未遍历的元素的值赋值给循环变量，然后执行语句块 1，再判断遍历序列中有无未遍历的元素，若有，取出该值赋值给循环变量，继续执行语句块 1，直至取完序列中的所有值，循环结束。循环结束后如果有 else 子句则执行语句块 2，如果没有 else 则执行循环结构之后的语句。

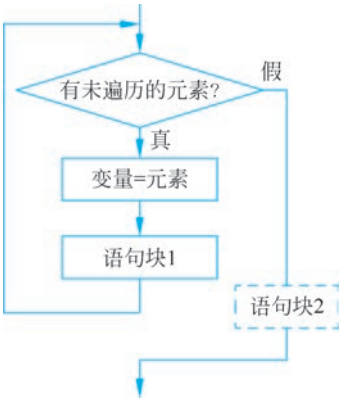

图 3-5　for 语句的执行流程

for 语句的执行流程图如图 3-5 所示。

例 3-7：盈盈为了考验令狐冲夺冠的决心，要他说一百遍“我能行!”。要求每十次回答占一行，一共输出 10 行。

分析：说一百遍“我能行!”相当于在程序中输出 100 次字符串“我能行!”，所以要用到循环结构。在循环输出时每 10 次输出占一行，那么 print()函数中 end 参数就必须重新赋值，默认是换行符，在输出之前要判断次数是否是 10 的倍数（次数除以 10 的余数为 0），如成立，则使用 print()换行。实现代码如下。

```
print("盈盈问:你能夺冠吗?")
print("令狐冲回答:")
for answer in range(100):
    if answer % 10 == 0:
```

```
        print()                    # 每输出 10 个换行
    print("我能行!",end = ' ')     # 不换行
```

运行结果如下。

```
盈盈问:你能夺冠吗?
令狐冲回答:

我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行! 我能行!
```

例 3-8: 计算 1～100 所有偶数的和。

分析: 使用 range 函数构造一个 1 到 100 的偶数序列,设置步长为 2 即可,每次从这个序列中取出一个值和之前的和进行相加,这个过程称为累加。

```
# 求 1～100 之和
sum = 0                          #定义一个变量存放结果,初值为 0
for i in range(2,101,2):
    sum = sum + i                # 累加
print("1 到 100 之和为:",sum)    # 循环结束后输出最终结果
```

运行结果如下。

```
1 到 100 之和为: 2550
```

试一试将 print 语句和 sum＝sum＋i 语句缩进一致,结果是什么?

例 3-9: 从键盘输入一行英文句子,统计句子中大写字符、小写字符和数字各有多少个?

分析: 字符串是可以迭代的,因此字符串也可以作为循环中的遍历序列,遍历字符串中的每一字符,使用字符串函数 isupper()、islower()和 isdigit()来判断这字符是大写字符、小写字符还是数字,并且定义三个变量 count_upper、count_lower、count_digit 存放计数结果。实现代码如下。

```
# 统计英文句子中大写字符、小写字符和数字各有多少个

str = input("请输入一句英文:") #从键盘输入一行字符
# 定义三个变量分别存放大写字符、小写字符和数字的个数
count_upper = 0
count_lower = 0
count_digit = 0
# 遍历字符串中的每一字符进行判断
for s in str:
    if s.isupper(): count_upper = count_upper + 1
```

```
        if s.islower(): count_lower = count_lower + 1
        if s.isdigit(): count_digit = count_digit + 1
    # 输出结果
    print("大写字符: %d 个" % count_upper)
    print("小写字符: %d 个" % count_lower)
    print("数字字符: %d 个" % count_digit)
```

运行结果如下。

```
请输入一句英文:This boy is 12 years old.
大写字符:1 个
小写字符:16 个
数字字符:2 个
```

3）while 语句


while 语句

在 for 语句中，关注的是遍历序列的个数和元素的值，然而有的时候循环的初值和终值并不明确，但有清晰的循环条件，这时采用 while 语句会比较方便。while 语句中用一个表示逻辑条件的表达式来控制循环，当条件成立的时候反复执行循环体，直到条件不成立的时候循环结束。语法格式如下。

```
while 条件表达式:
      语句块
```

同样，条件表达式后面的冒号“:”不可省略，语句块要注意缩进。执行 while 语句的时候，先求条件表达式的值，如果值为 True 就执行循环体语句一次，然后重复上述操作；当条件表达式的值为 False 的时候，while 语句执行结束。注意 while 语句中也可以有 else 子句，用法和 for 语句相同。执行流程如图 3-6 所示。

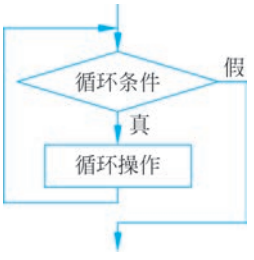

图 3-6　while 语句的执行流程

例 3-10：利用 while 语句求 1 至 10 中所有偶数的乘积。

分析：在 1～10 中，首先对第一个数 i＝1 判断奇偶性，如果是偶数则累乘，然后将计数器 i 加 1，如果 i <＝10，继续判断第 2 个数、第 3 个数……对所有的偶数循环累乘，直到 i 为 10 时结束循环。注意存放结果的变量初始值为 1，否则累乘的时候结果永远为 0。实现代码如下。

```
# 求 1 至 10 中所有偶数的积
sum = 1 # 初始值为 1
i = 1
while i <= 10:
    if i % 2 == 0:
        sum = sum * i
    i = i + 1
print("1 至 10 中所有偶数积为:", sum)
```

运行结果如下。

```
1 至 10 中所有偶数积为: 3840
```

与前面的 for 语句相比,使用 while 语句的时候,必须使用一个变量控制循环,程序中的“i=i+1”就是对变量 i 进行增量操作。如果去掉“i=i+1”这条命令,变量 i 的值一直等于 1,循环条件“i<=10”一直成立,这个循环就一直无法结束,变成了“死循环”。for 与 while 相比较而言,如果循环比较规范、循环中的控制比较简单、事先可以确定循环次数,那么用 for 语句写的程序往往会更简单清晰。

例 3-11:录入学生的 Python 课成绩,计算其平均成绩。

分析:从键盘输入一名学生的成绩,使用条件循环,询问是否要继续循环,如果回答是大写 Y 或者小写 y,则继续,否则退出循环。每循环一次,要统计输入的学生人数,计算他们的总成绩,循环结束后,输出有几人参加考试、平均成绩是多少。实现代码如下。

```
# 录入学生的 Python 课成绩,计算其平均成绩
print("开始录入成绩:")
answer = 'y'                                    # 循环控制变量,初值为 y
total = 0                                       # 存放总成绩
i = 0                                           # 存放学生人数
while answer == 'y' or answer == 'Y':           # 当 answer 的值是 y/Y 时,执行循环
    i = i+1                                     # 学生人数加 1
    print('输入第 %d 个同学的成绩:' % i)
    score = float(input("成绩 = "))             # 将键盘输入的成绩转换为浮点数
    total = total + score # 所有成绩累加
    answer = input("继续输入吗?(y/Y)")          # 输入 y 或者 Y,循环继续,否则循环结束
print("一共有 %d 个学生参加了 Python 考试,平均分为 %0.2f" %(i,total/i))
```

运行结果如下。

```
开始录入成绩:
输入第 1 个同学的成绩:
成绩 = 89
继续输入吗?(y/Y)y
输入第 2 个同学的成绩:
成绩 = 78
继续输入吗?(y/Y)y
输入第 3 个同学的成绩:
成绩 = 68
继续输入吗?(y/Y)n
一共有 3 个学生参加了 Python 考试,平均分为 78.33
```

思考:无论从键盘输入大写 Y 还是小写 y,都通过字符串函数转换成大写统一判断,程序如何修改?

提示:使用 upper()函数。

例 3-12:小明在 2020 年存入 5000 元,假设每年按复利增长 3%,请问按此增长速度,到哪一年存款将达到 1 万元?

分析:按复利计算的话,每年的存款金额为本金加本金乘以利率,如第一年的存款是本金加本金乘以 3%,这个结果作为第二年的本金,以此类推,每年的本金都是前一年的本金加本金乘以利率的和,可以用循环实现这种类推。实现代码如下。

```
money = 5000                    # 使用 money 表示本金,第一年的本金是 5000 元
year = 0
while money <= 10000:           # 循环条件
      # 本金加本金乘以利率再赋值给 money,作为下一次循环的本金
      money = money * (1 + 0.03)
      year = year + 1
print(year,"年后存款达到一万元")
```

运行结果如下。

```
24 年后存款达到一万元
```

修改问题为十年后小明的存款将达到多少？实现代码如下。

```
# 存款 5000 元,利息 3%,十年后存款为多少
money = 5000      # 存入 5000 元
year = 0
while year < 10:
      money = money * (1 + 0.03)
      year = year + 1
      print("第 %d 年存款为: %0.2f 元" % (year,money))
```

运行结果如下。

```
第 1 年存款为:5150.00 元
第 2 年存款为:5304.50 元
第 3 年存款为:5463.64 元
第 4 年存款为:5627.54 元
第 5 年存款为:5796.37 元
第 6 年存款为:5970.26 元
第 7 年存款为:6149.37 元
第 8 年存款为:6333.85 元
第 9 年存款为:6523.87 元
第 10 年存款为:6719.58 元
```

4）循环嵌套

循环嵌套指的是循环体里面还包含循环。包含另一个循环结构的循环称为外循环，被包含的循环称为内循环。循环嵌套在执行过程中，外循环每执行一次，内循环就要完整地执行一遍。break 和 continue 语句只对本层循环有效。

while 和 for 循环可以互相嵌套，自由组合。外循环体中可以包含一个或多个循环结构，但必须完整包含，不能出现交叉现象。内循环作为外循环的循环体，内循环结构要整体缩进。

```
  *
 **
***
****
*****
****
***
 **
  *
```

图 3-7　菱形图

例 3-13：使用循环嵌套输出如图 3-7 所示的图形。

分析：从图形看出，一共是 9 行，对于每一行来讲，由若干个 * 号组成，由外循环控制行数，内循环控制每行的输出。每一行为了构成一个三角形，在输出 * 号之前还需输出一定的空格，每一行空格的个数和 * 号的个数都是有规律的。菱形的上半部分空格的个数是 4 减行数，* 号的个数和行的数字一致。菱形的下半部分空格的个数是行数减 4，* 号的个数 9

减行数。实现代码如下。

```
# 输出一个菱形图形
i = 0                              # i控制外循环,初值为0
while i<9:                         # 总循环次数9
    if i<5:                        # 0到4,这5行是上半部分菱形
        # 上空格部分
        j = 0                      # j控制内循环
        while j<4 - i:
            print(" ",end="") # 输出一个空格,之后不换行
            j +=1
        #上部分*号
        j = 0
        while j<i+1:
            print("*", end=" ")
            j +=1
    else:
        # 下空格部分
        j = 0
        while j<i -4:
            print(" ",end="")
            j +=1
        # 下部分
        j = 0
        while j<9 - i:
            print("*",end=" ")
            j +=1
    print() # 内循环完成后一行的输出结束,所以要换行
    i += 1 # 外循环控制变量加1,继续下一次循环
```

可以将程序中输出空格的地方用别的符号代替,体会空格的个数与行的关系,空格的个数从第一行到第九行依次是4、3、2、1、0、1、2、3、4,如图3-8所示。

```
@@@@*
@@@* *
@@* * *
@* * * *
* * * * *
@* * * *
@@* * *
@@@* *
@@@@*
```

图3-8 空格用@符号填充的菱形图

例3-14:分别用for循环和while循环输出九九乘法表。

分析:外循环控制行,内循环控制列,重点是寻找每一行与每一列之间的关系。九九乘法表中行与列的关系为列数=行数,内循环要循环几次,取决于是第几行。实现代码如下。

```
#用for循环写九九乘法表
for i in range(1,10):
    for j in range(1,i+1):
        print("%d*%d=%d"%(i,j,i*j),end='\t')    #不换行,每一项用一个制表位分隔
    print()                                     # 一行输出完后换行

#用while循环写九九乘法表
i = 1                                           # i控制外循环
while i<10:
    j = 1                                       # j控制内循环
```

```
    while j <= i:
        print("%d * %d = %d" %(i,j,i * j),end = '\t')
        j += 1
    print()                                          # 一行输出完后换行
    i += 1
```

运行结果如下。

```
1 * 1 = 1
2 * 1 = 2  2 * 2 = 4
3 * 1 = 3  3 * 2 = 6  3 * 3 = 9
4 * 1 = 4  4 * 2 = 8  4 * 3 = 12  4 * 4 = 16
5 * 1 = 5  5 * 2 = 10  5 * 3 = 15  5 * 4 = 20  5 * 5 = 25
6 * 1 = 6  6 * 2 = 12  6 * 3 = 18  6 * 4 = 24  6 * 5 = 30  6 * 6 = 36
7 * 1 = 7  7 * 2 = 14  7 * 3 = 21  7 * 4 = 28  7 * 5 = 35  7 * 6 = 42  7 * 7 = 49
8 * 1 = 8  8 * 2 = 16  8 * 3 = 24  8 * 4 = 32  8 * 5 = 40  8 * 6 = 48  8 * 7 = 56  8 * 8 = 64
9 * 1 = 9  9 * 2 = 18  9 * 3 = 27  9 * 4 = 36  9 * 5 = 45  9 * 6 = 54  9 * 7 = 63  9 * 8 = 72  9 * 9 = 81
```

5）循环控制语句


循环控制语句

在循环结构中，可以使用控制语句来改变程序的流程。控制语句主要有 break 和 continue 语句。break 语句用于结束整个循环，continue 语句的作用是结束本次循环，紧接着执行下一次的循环。

（1）break 语句。

若要在循环中提前跳出循环，继续执行循环后的代码，则要使用 Python 中的 break 语句。break 语句的作用是结束当前循环然后跳转到循环后的下一条语句继续执行。

例 3-15：判断任意一个数 n 是否是素数。

分析：素数的定义是除了 1 和它本身外，不能被任何一个数整除。所以从 2 开始到 n－1，寻找 n 的约数。如果在循环中不使用 else 子句，实现代码如下。

```
# 判断任意一个数是否为素数
# 不使用 else
found = True
i = int(input("输入一个整数:"))
for j in range(2,i):
    if i % j == 0:
        found = False
        break
if found:
    print('%d 是一个素数' % i)
else:
    print('%d 不是一个素数' % i)
```

如果在循环遍历的过程中，发现有一个整数 i 是 n 的约数，即 i % j == 0，那就不必再循环遍历下去，因为此时已经可以判定 n 不是素数，程序中使用 break 语句退出了循环。我们借助了一个标志量 found 来判断循环结束是不是由 break 语句引起的，如果对循环的 else 子句善加利用，代码可以简洁得多。

```
# 判断任意一个数是否为素数
# 使用 else
i = int(input("输入一个整数:"))
for j in range(2,i):
    if i % j == 0:
        print('%d 不是一个素数'%i)
        break
else:
    print('%d 是一个素数'%i)
```

运行结果如下。

```
输入一个整数:53
53 是一个素数
```

当循环“自然”终结(循环条件为假)时,else 从句会被执行一次,而当循环是由 break 语句中断时,else 从句就不被执行。else 子句使程序员的生产力和代码的可读性都得到了提高。

判断一个循环是正常结束还是遇到 break 语句退出循环,也可以通过判断循环变量的值来确定,修改上述代码如下。

```
# 判断一个正整数 n(n>=2)是否为素数
n=int(input("输入一个正整数 n(n>=2):"))
for i in range(2,n):          #n 除以 2,3,4,…,n-1
    if n%i==0:                #只要一个整除说明不是素数
        break                 #结束循环
if i == n-1:
    print(n,"是素数")
else:
    print(n,"不是素数")
```

对于输入的正整数 n 来说,判断它是否为素数,就是在 2～n－1 的范围中寻找 n 的约数。如果在循环遍历的过程中,发现有一个整数 i 是 n 的约数,即 i 把 n 整除了,那就不必再循环遍历下去,因为此时已经可以判定 n 不是素数,程序中使用 break 语句退出了循环。注意当遇到 break 语句退出循环的时候,遍历还未结束,此时的 i 仍然在 2～n－1 之间。如果 n 是素数,循环情况又会怎样呢? 当 n 是素数的时候。循环体中的 if 条件永远不会成立,break 语句永远执行不到,只有当 i 的取值超出 range()的迭代范围时,循环才会退出,因此正常退出循环时 i 的值一定等于 n－1。for 语句后的 if/else 结构正是根据 i 的取值来判断循环的执行情况,从而得到 n 的判定结果。

例 3-16:模拟登录系统账号密码检测功能,并限制账号或密码输错的次数至多为 3 次。

登录系统一般具有账号密码检测功能,即检测用户输入的账号密码是否正确。若用户输入的账号或密码不正确,提示“用户名或密码错误”和“您还有 * 次机会”;若用户输入的账号和密码正确,提示“登录成功”;若输入的账号密码错误次数超过 3 次,提示“输入错误次数过多,请稍后再试”。实现代码如下。

```
#模拟登录系统账号密码检测功能
count = 0                                          # 用于记录用户错误次数
while count < 3:
      user = input("请输入您的账号:")
      pwd = input("请输入您的密码:")
      if user == 'admin' and pwd == '123':         # 进行账号密码比对
         print('登录成功')
         break
      else:
         print("用户名或密码错误")
         count += 1                                # 初始变量值自增 1
         if count == 3:                            # 如果错误次数达到 3 次,则提示并退出
             print("输入错误次数过多,请稍后再试")
         else:
             print(f"您还有{3 - count}次机会")     # 显示剩余次数
```

运行该程序,当输入的用户名和密码正确时,运行结果如下。

```
请输入您的账号:admin
请输入您的密码:123
登录成功
```

再次运行程序,当输入的用户名或者密码三次都没有输对时,运行结果如下。

```
请输入您的账号:admin
请输入您的密码:123456
用户名或密码错误
您还有 2 次机会
请输入您的账号:zhangsan
请输入您的密码:123
用户名或密码错误
您还有 1 次机会
请输入您的账号:asd
请输入您的密码:asd
用户名或密码错误
输入错误次数过多,请稍后再试
```

例 3-17：使用绝对循环与判断退出的方法修改例 3-12：小明在 2020 年存入 5000 元,假设每年按复利增长 3%,请问按此增长速度,到哪一年存款将达到 1 万元?

分析：在例题 3-12 中根据条件判断是否满足循环,满足条件执行循环,直到条件不成立循环结束。在 while 循环中也经常将循环条件设为真,这样循环条件就永远满足,这种循环叫绝对循环,也叫死循环,如果在循环体中没有判断退出的语句,该循环由于条件是真,所以会一直循环下去,因此叫死循环。为了能够退出循环,在循环体中一定要有判断退出的语句。实现代码如下。

```
money = 5000                # 本金存入 5000 元
year = 0
while True:                 # 绝对循环
      year = year + 1
      money = money * (1 + 0.03)
```

```
        if money>=10000:                    # 存款额超过一万就退出循环
            break
    print(year,"年后存款达到一万元")
```

说明：在 Python 中如果值为非 0,或者对象不为空,就都是真,那么 while True 子句中的 True 也可以换成任何非 0 的值,一般用 while 1。

(2) continue 语句。

前面我们学习了 for 语句和 while 语句,知道 while 语句是在某一条件成立时循环执行一段代码块,而 for 语句是迭代一个集合的元素并执行一段代码块。然而有时可能需要提前结束一次迭代,进行新的一轮迭代。在循环体中,如果遇到某种情况希望提前结束本次循环,并继续进行下次循环时,可以使用 continue 语句。continue 语句与 break 语句的不同之处在于 break 将结束本次循环并跳出循环,而 continue 仅仅是提前结束当前这次循环,继续进行下一次循环。

例 3-18：输出 1 到 10 中所有不是 3 的倍数的所有数字。

说明：使用循环遍历 1 到 10 之间的每一个数,判断这个数能否被 3 整除,能整除就退出本次循环,继续判断下一个数,否则输出这个数。实现代码如下。

```
# 输出 1 到 10 中所有不是 3 的倍数的所有数字
for i in range(1,10+1):
    if i % 3 == 0:
        continue
    print(i,end=',')   # 输出的每一项用逗号分隔
```

运行结果如下。

```
1,2,4,5,7,8,10,
```

例 3-19：“逢七拍腿”游戏。

规则是：参与游戏者排成一圈,从某人开始依次从 1 开始按顺序数数,数到含有 7 或 7 的倍数的人要拍腿表示越过,比如,数到 7、14、17 这类数字的人都不能数出该数字,要拍一下腿,然后下一人继续数后面的数字。请编程模拟这个游戏过程,计算从 1 数到 100,一共要拍腿多少次。

分析：逢七拍腿游戏中,遇到含有 7 或 7 的倍数的数字时要拍腿,现在要计算从 1 数到 100 有多少人次拍腿,那我们可以在 for 循环中判断某个数是否符合拍腿要求,若不符合则跳过累加拍腿人次,继续循环下一个数。判断一个数 number 如果不是 7 的倍数则条件表达式为 number % 7!=0,判断该数是否含有数字 7 可以将该数转换为字符串,利用字符串函数 endswith("7")判断是否以 7 结尾。实现代码如下。

```
total = 0                               # 记录拍腿次数的变量
for number in range(1,101):             # 创建一个从 1 到 101(不包括)的循环
    # 判断非 7 的倍数或非 7 为尾数时,跳过 total 的累加,继续循环判断下一个数
    if number % 7 != 0 and not str(number).endswith("7"):
        continue                        # 继续下一次循环
    total += 1
```

```
        print(number,end = ' ')                    # 输出需要拍腿的那个数字,以空格分隔每一个数字
print() # 输出完满足拍腿的数字后换行
print("从 1 数到 100 共拍腿",total,"次") # 显示拍腿次数
```

运行结果如下。

```
7 14 17 21 27 28 35 37 42 47 49 56 57 63 67 70 77 84 87 91 97 98
从 1 数到 100 共拍腿 22 次
```

例 3-20：猜数字游戏。

分析：要猜的这个数是在 1～100 的随机整数，可以通过随机数的方法产生一个 1～100 的随机数字，每个数字允许猜几次，即循环几次。接收用户从键盘输入的数字，表示用户猜的那个数字，如果猜对了，游戏结束，也就是即使没有循环完 n 次也强行退出循环。如果猜大了，提示猜大了，如小了，提示猜小了，返回循环重新再猜。如果 n 次都没猜对循环也结束，做循环以后的语句或者 else 子句。

Python 有一个随机数模块 random，通过 import random 语句将这个模块导入，该模块中包含的函数在程序中就可以直接使用了。有关模块的内容将在后面的章节中介绍，在这里简单理解为模块就是为了实现某一功能已经有人事先做好代码存放在一起，可以由别的程序调用无需重新再写代码，import 语句相当于有一个导航指向我们需要使用的函数的位置。在 random 模块中有一个 randint 函数，调用该函数可以生成随机整数。实现代码如下。

```
#猜数字游戏
import random                                  # 导入随机数模块
# num 变量存放系统生成的随机数
num = random.randint(1,101)                    # 调用随机模块中的 randint 函数产生 1～100 的整数
# print(num),                                  # 如果想知道这个随机数是多少,可以输出看看
for i in range(5):                             # 允许猜五次
    guess = int(input('请输入你猜的数'))# guess 变量存放用户猜的数
    if guess > num:                            # 比较用户猜的数和系统生成的随机数
        print('大了')
        continue                               # 没猜对退出本次循环,返回再猜(做下一次循环)
    elif guess == num:
        print('猜对了')
        break                                  # 猜对后直接跳出循环 else 子句也不会做
    else:
        print('小了')
        continue
else:                                          # 循环结束后执行
    print('错误次数过多')
```

在规定的次数内猜对了，运行结果如下。

```
请输入你猜的数 56
大了
请输入你猜的数 36
猜对了
```

在规定的次数内没有猜对，会执行 else 子句，运行结果如下。

```
请输入你猜的数 56
大了
请输入你猜的数 46
大了
请输入你猜的数 38
大了
请输入你猜的数 26
大了
请输入你猜的数 21
大了
错误次数过多
```

上述案例也可以使用 while 循环实现,需要考虑循环结束的条件。条件就是猜的次数没有超过允许的上限,则循环可以一直做。

```
import random                    # 引入 random 库,使用随机函数
num = random.randint(1, 101)  # 随机产生的数字
count = 0
while count < 4:
    count += 1
    guess = int(input('请猜一个数成)) # int 类型
    if guess > num:
        print('大了')
        continue
    elif guess == num:
        print('对了')
        break
    else:
        print('小了')
        continue
else:
    print('错误次数过多')
```

3.3 任务设计思路

小明要去超市购物,超市有很多商品,每种商品的价格已经在标签上打好了,从程序的角度来讲,这些商品的价格就是确定的常量,假设小明要买五种商品,这五种商品的名称和价格都是确定的,所以使用 print 函数将商品名称和价格显示出来供小明选择,相当于在超市看到的商品和价格标签。

小明选择商品的过程,在程序中就是使用 input 函数,输入商品名称或者编号以及要买的数量,由于价格已经作为常量定义了,有了数量,购买这类商品的价格就很容易计算出来。

如果小明要继续买下一种商品,程序中和购买上一种商品的做法一样,使用循环即可实现。

3.4 任务实施

步骤一:小明去超市购物,首先看到的是陈列的商品以及商品标签上打好的价格,小明要买五种商品,从程序的思维来讲,就是将这五种商品的名称和价格在控制台输出,首

先定义 5 个商品的价格，再输出 5 种商品的对应的序号供选择（挑选商品）。实现代码如下。

```
# 超市购物
# 1.商品展示
# 定义变量存放商品的价格
toothbrush = 8.8            # 牙刷价格
towel = 10.0                # 毛巾价格
cup = 18.8                  # 水杯价格
apple = 12.5                # 苹果价格
banana = 15.5               # 香蕉价格
print("--- MeetAll 超市 ---")
print()
print(" 1.牙刷：%0.2f 元" % toothbrush)
print(" 2.毛巾：%0.2f 元" % towel)
print(" 3.水杯：%0.2f 元" % cup)
print(" 4.苹果：%0.2f 元" % apple)
print(" 5.香蕉：%0.2f 元" % banana)
print("------------------")
```

运行结果如下。

```
--- MeetAll 超市 ---
  1.牙刷:8.80 元
  2.毛巾:10.00 元
  3.水杯:18.80 元
  4.苹果:12.50 元
  5.香蕉:15.50 元
------------------------
```

步骤二：接下来小明开始购买商品，要确定第一种要买的商品是什么？数量是多少？程序中需要使用到 input 函数，让用户填写购买商品的序列号以及购买的数量，然后计算这种商品一共消费多少金额。实现代码如下。

```
# 2.开始购物，只购一种商品
goods_No = input('请输入要采购的商品编号：')    # 输入的编号不经转换就按照字符串使用
count = int(input('请输入要购买商品数量：'))
if goods_No == '1':
    money = count * toothbrush                  # 计算购买牙刷所需的金额
elif goods_No == '2':
    money = count * towel                       # 计算购买毛巾所需的金额
elif goods_No == '3':
    money = count * cup                         # 计算购买水杯所需的金额
elif goods_No == '4':
    money = count * apple                       # 计算购买苹果所需的金额
elif goods_No == '5':
    money = count * banana                      # 计算购买香蕉所需的金额
print('这次购物花费 %0.2f 元' % money)
```

运行结果如下。

```
--- MeetAll 超市 ---
  1.牙刷:8.80 元
  2.毛巾:10.00 元
  3.水杯:18.80 元
  4.苹果:12.50 元
  5.香蕉:15.50 元
------------------------
请输入要采购的商品编号:2
请输入要购买商品数量:3
这次购物花费 30.00 元
```

步骤三：小明是否还要购买其他商品，如果要买，流程和第二步一样，还是使用 input 函数输入另一种要买的商品的名称和数量；如此反复，将需要购买的商品全部买好；在程序中使用循环实现这个过程，需要将步骤二的代码作为循环体。在这一步，要设置循环条件，实现代码如下。

```
# 3.无限次购物,直到不再购物
money = 0
answer = 'y'                                    # 定义循环控制变量 answer
while answer == 'y'or answer == 'Y':            # 当 answer 值为 y(大小写均可)时循环
    goods_No = input('请输入要采购的商品编号:') # 输入的编号按照字符串使用
    count = int(input('请输入要购买商品数量:')) # 数量要转换成整数才能计算
    if goods_No == '1':
        money = money + count * toothbrush      # 计算购买牙刷后累计所花费的金额
    elif goods_No == '2':
        money = money + count * towel           # 计算购买毛巾后累计所花费的金额
    elif goods_No == '3':
        money = money + count * cup             # 计算购买水杯后累计所花费的金额
    elif goods_No == '4':
        money = money + count * apple           # 计算购买苹果后累计所花费的金额
    elif goods_No == '5':
        money = money + count * banana          # 计算购买香蕉后累计所花费的金额
    answer = input('继续购物吗?')               # 改变循环控制变量的值
print('这次购物花费 %0.2f 元' % money)
```

运行结果如下。

```
-  --- MeetAll 超市 ---
  1.牙刷:8.80 元
  2.毛巾:10.00 元
  3.水杯:18.80 元
  4.苹果:12.50 元
  5.香蕉:15.50 元
------------------------
请输入要采购的商品编号:1
请输入要购买商品数量:2
继续购物吗?y
请输入要采购的商品编号:5
请输入要购买商品数量:3
继续购物吗?y
请输入要采购的商品编号:3
```

```
请输入要购买商品数量:1
继续购物吗?n
这次购物花费 82.90 元
```

程序说明：购买商品后计算金额得到的是累计花费多少钱，所以是在上一次花费的基础上加上本次消费。本任务中小明一共可以买五种商品，代码中用了五个判断，如果要买 50 种商品是否要写 50 个条件判断呢？如果这样就违背了编程的初衷，程序就是要将这种繁琐的重复自动化，这个问题我们在学习后续的知识后就可以很好地解决。本任务以这种小样本的数据讲解任务的实现原理。在后续的学习中逐步优化程序。

3.5　任务小结

本任务的完成主要使用了 Python 控制语句，包括判断语句、循环语句以及其他子句。其中，判断语句主要是 if 语句，循环语句主要是 for 语句和 while 语句。程序流程控制中条件表达式都是必不可少的。

条件表达式为零，表示条件为假；

条件表达式为非零时，表示条件为真；

条件表达式可以是任何数值类型表达式；

条件表达式也可以是字符串。

Python 的语句体用缩进形式来表示，缩进不正确，会导致逻辑错误

continue 的意思是，跳出本次循环，继续从头开始循环；break 的意思是停止整个循环，continue 和 break 语句之后的代码都是不执行的。

在 Python 开发中，这些语句的使用频率非常高，希望读者可以多加以理解，熟练掌握它们的使用。

3.6　技能训练

一、单选题

1. 有两个条件 p 和 q，只要有一个条件为真，结果一定为真的值是(　　)。

A. not p　　B. p and q

C. p or q　　D. not p and not q

2. 以下关于分支结构的描述中，错误的是(　　)。

A. 单分支结构是用 if 关键字判断满足一个条件，就执行相应的处理代码

B. 二分支结构是用 if-else 根据条件的真假，执行两种处理代码

C. 多分支结构是用 if-elif-else 处理多种可能的情况

D. python 在分支语句里使用例如 x<=y<=z 的表达式是非法的

3. 表达式 't' if 'd' else 'f' 的执行结果是(　　)。

A. True　　B. False　　C. 't'　　D. 'f'

4. 下列语句执行后的结果是(　　)。

```
if -1:
    print("成功!")
else:
    print("失败!")
```

A. 成功　　B. 失败　　C. 没有输出　　D. 运行错误

5. 有下面的程序段。

```
if k<=10 and k>0:
    if k>5:
        if k>8:
            x=0
        else:
            x=1
    else:
        if k>2:
            x=3
        else:
            x=4
```

其中 k 取(　　)哪组值时,x=3。

A. 3,4,5　　B. 3,4　　C. 5,6,7　　D. 4,5

6. 以下关于循环结构的描述,错误的是(　　)。

A. 遍历循环的循环次数由遍历序列中的元素个数来体现

B. 非确定次数循环的次数是根据条件判断来决定的

C. 非确定次数的循环用 while 语句来实现,确定次数的循环用 for 语句来实现

D. 遍历循环对循环的次数是不确定的

7. 以下代码输出结果为(　　)。

```
for i in [1, 0]:
print(i+1)
```

A. 2　1　　B. [2, 1]　　C. 2　　D. 0

8. 这段代码将输出(　　)。

```
for i in range(10):
    if not i % 2 == 0:
        print(i + 1)
```

A. 输出 2 到 10 的偶数　　B. 输出 1 到 9 的奇数

C. 输出 0 到 8 的偶数　　D. 输出 2 到 8 的偶数

9. 有如下代码,运行后会输出 counter 的值是(　　)。

```
counter = 0
while counter < 100:
    counter += 1
print(counter)
```

A. 0　　B. 99　　C. 100　　D. 101

10. 可以结束一个循环的关键字是(　　)。

A. exit　　B. if　　C. break　　D. continue