

结构化程序设计主要由顺序结构、选择结构、循环结构三种逻辑结构组成。1966 年, 计算机科学家 C. Bohm 和 G. Jacopini 在数学上证明, 任何程序都可以采用顺序、选择、循环三种基本结构实现。结构化程序设计具有以下特点: 一是程序内的每一部分都有机会被执行; 二是程序内不能存在“死循环”(无法终止的循环); 三是程序尽量保持一个入口和一个出口, 程序有多个出口时, 只能有一个出口被执行; 四是少用 goto 类语句。

3.1 顺序结构

顺序结构是一种有序结构, 它依次执行各语句模块。如图 3-1 所示, 顺序结构程序在执行完语句 1 指定的操作后, 接着执行语句 2, 直到所有语句执行完成。常见的顺序语句有导入语句、赋值语句、输入输出语句和其他程序语句等。下面主要介绍前三种顺序语句。

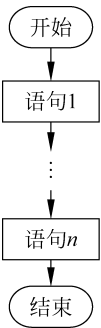


图 3-1 顺序结构

3.1.1 导入语句

1. 模块导入

Python 中库、包、模块等概念本质上就是操作系统中的目录和文件。模块导入就是将软件包、模块、函数等程序加载到计算机内存, 方便程序快速调用。

Python 标准库模块繁多, 如果一次性将全部模块都导入内存, 这会占用很多系统资源, 导致程序运行效率很低。因此, Python 对模块和函数采用“加载常用函数, 其他需用再导入”的原则。所有函数都通过 API(Application Program Interface, 应用程序接口)调用, 本书提供了常用函数的调用方法和案例, 其他标准函数的调用方法可以用函数 help() 查看, 或者参考 Python 使用指南(见 <https://docs.python.org/3/library/index.html>)。

Python 程序中的函数有内置标准函数、导入标准函数、第三方软件包中的函数、自定义函数。内置函数(如 print()、len()、help()等)在 Python 启动时已经导入内存, 无须再次导入; 大部分标准函数模块(如 math、sys、time 等)都需要先导入再调用; 第三方软件包中所有模块和函数都需要先安装, 再导入和调用。

2. 模块绝对路径导入——import 语句

导入语句 import 可以导入一个目录下的某个模块, 模块绝对路径导入语法如下。

1	import 模块名	# 绝对路径导入, 导入软件包或模块
2	import 模块名 as 别名	# 绝对路径导入, 别名用于简化调用

- (1) 以上“模块名”包括库、包、模块等名称。
- (2) 同一模块导入多次只会执行一次,这防止了同一模块的多次执行。
- (3) 绝对路径导入时,调用时采用点命名形式,如“模块名.函数名()”。
- (4) 模块有哪些函数? 函数如何调用? 这些问题必须查阅软件包用户指南。
- (5) 如果当前目录下存在与导入模块同名的.py文件,就会将导入模块屏蔽。

【例 3-1】 根据勾股定理 $a^2 + b^2 = c^2$, 计算直角三角形边长。

案例分析: 用勾股定理求边长需要进行开方运算,这需要用数学模块 `math`。

1	<code>import math</code>	# 导入标准模块——数学计算
2	<code>a = float(input('输入直角三角形第 1 条边长:'))</code>	# 输入边长,转换为浮点数
3	<code>b = float(input('输入直角三角形第 2 条边长:'))</code>	
4	<code>c = math.sqrt(a*a + b*b)</code>	# 调用开方函数(需要写模块名 <code>math</code>)
5	<code>print('直角三角形的第 3 条边长为:', c)</code>	# 打印计算值
	<code>>>></code> 输入直角三角形第 1 条边长:3 输入直角三角形第 2 条边长:4 直角三角形的第 3 条边长为: 5.0	# 程序输出

程序说明: 程序第 4 行,函数 `math.sqrt()` 为点命名方法,`math` 为数学模块,`sqrt()` 为 `math` 模块的开方函数,它只能接受非负的实数。一个模块会有多个函数,标准函数的调用方法可以在 Python shell 窗口下用命令“`help(函数名)`”查看,如 `help(print)`。

【例 3-2】 导入第三方软件包 `Matplotlib` 中的画图子模块 `pyplot`。

1	<code>import matplotlib.pyplot</code>	# 导入第三方软件包——画图模块
---	---------------------------------------	------------------

案例分析: 执行 `D:\Python\Lib\site-packages\matplotlib\pyplot.py` 程序,其中 `D:\Python\Lib\site-packages\` 是软件包安装位置,这个路径在 Python 安装时已经设置好,无须重复说明; `matplotlib` 是软件包名(子目录); `pyplot` 是画图子模块名(文件)。

3. 函数相对路径导入——`from...import` 语句

函数相对路径导入语法如下。

1	<code>from 模块名 import 函数名</code>	# 函数相对路径导入,导入指定函数
2	<code>from 模块名 import *</code>	# 导入模块中所有函数和变量(慎用)

相对路径导入时,“模块名”是软件包中的模块(文件)或者子模块,“函数名”是模块中定义的函数、类、公共变量(如 `pi`)等,可以一次导入多个函数。

相对路径导入时,函数调用直接采用“函数名()”的形式,无须使用“模块名.函数名()”的格式。相对路径导入使用方便,缺点如下: 一是搞不清楚函数来自哪个模块; 二是容易造成同一程序中同名函数问题。

【例 3-3】 导入标准库数学模块中的开方函数。

1	<code>>>> from math import sqrt</code>	# 导入标准模块——导入 <code>math</code> 模块中的 <code>sqrt()</code> 函数
2	<code>>>> sqrt(2)</code>	# 调用函数对 2 进行开方运算(不需要写模块名 <code>math</code>)
	1.4142135623730951	

【例 3-4】 用 `from...import` 导入模块中的几个函数,而不是导入所有函数。

1	<code>from math import sqrt, sin, cos, pi</code>	# 导入标准模块——导入数学模块中的函数和常量
---	--	-------------------------

案例分析：语句导入了 math 模块中的 sqrt()(开方)、sin()(正弦)、cos()(余弦)三个函数,以及 math 模块中定义的常量 pi(圆周率,3.141 592 653 589 793)。

4. 对导入软件包取别名

导入软件包时,如果对包或模块取别名后,调用函数时书写会简化很多。

【例 3-5】 导入第三方绘图包 Matplotlib 中的 pyplot 模块,并取别名为 plt; 导入第三方科学计算包 NumPy,并取别名为 np。程序输出如图 3-2 所示。

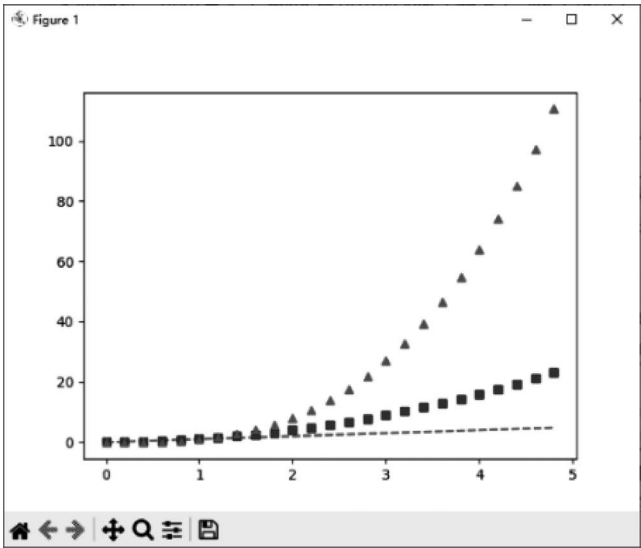


图 3-2 绘图程序输出

1	import matplotlib.pyplot as plt	# 导入第三方包——绘图包.绘图模块,别名为 plt
2	import numpy as np	# 导入第三方包——科学计算模块,别名为 np
3		
4	t = np.arange(0., 5., 0.2)	# 调用 NumPy 包中的 arange()函数
5	plt.plot(t, t, 'r-- ', t, t**2, 'bs', t, t**3, 'g')	# 调用时,用别名 plt 代替 matplotlib.pyplot
6	plt.show()	# 调用时,用别名 plt 代替 matplotlib.pyplot
	>>>	# 程序输出见图 3-2

程序说明如下：软件包 NumPy 的安装方法参见 1.1.4 节。

程序第 1 行,matplotlib 为第三方软件包名,pyplot 为子模块名; matplotlib.pyplot 表示只导入 Matplotlib 软件包中的绘图子模块 pyplot,程序第 5、6 行需要用到 pyplot 模块中的函数 plot()和 show(); matplotlib.pyplot 模块调用时可简写为 plt(别名)。

程序第 2 行,numpy 为导入软件包全部模块,NumPy 包别名为 np。

5. 语句 import 与 from...import 的区别

语句 import 相当于导入一个目录(绝对路径),这个目录下可能有很多模块,因此调用函数时,需要注明是哪个模块中的函数,如“模块名.函数名()”。

语句 from...import 相当于导入一个目录中的指定模块的指定函数,调用模块中的函数时直接使用“函数名()”就可以了,因为导入时指明了该函数所在模块。

语句 from...import * 是将一个模块中所有函数都导入进来。这种导入方式容易污染命名空间,应谨慎使用 from...import * 语句,避免程序中同名变量和同名函数的冲突。

3.1.2 赋值语句

1. 变量赋值常规方法

将值传送给变量的过程称为赋值。赋值语句中的“值”可以是数字或字符串,也可以是表达式、函数等。变量的数据类型在首次赋值时产生。赋值语句语法如下。

1	变量名 = 值或者表达式
---	--------------

赋值语句中,“=”是把等号右边的值或表达式赋给等号左边的变量名,这个变量名就代表了变量的值。

Python 赋值语句不允许嵌套赋值;不允许引用没有赋值的变量;不允许连续赋值;不允许不同数据类型用运算符赋值。试图这样做将触发程序异常。

【例 3-6】 错误的变量赋值方法。

1	>>> x = (y = 0)	# 异常,不允许赋值语句嵌套赋值
2	>>> print(y)	# 异常,不允许引用没有赋值的变量
3	>>> x = 2, y = 5	# 异常,一行多句时不能用逗号分隔
4	>>> s = '日期' + 2018	# 异常,不允许不同数据类型用运算符赋值
5	>>> 2 + 3 = c	# 异常,不允许赋值语句顺序颠倒

【例 3-7】 正确的变量赋值方法。

1	>>> path = 'd:\test\03\'	# 字符串赋值
2	>>> pi = 3.14159	# 浮点数赋值
3	>>> s = pi * (5.0 ** 2)	# 表达式赋值
4	>>> lst = ['张飞', '程序设计', 60]	# 列表赋值
5	>>> b = True	# 布尔值赋值

【例 3-8】 Python 中,一个变量名可以通过重复赋值,定义为不同的数据类型。

1	>>> ai = 1314	# 变量名 ai 赋值为整数
2	>>> ai = '一生一世'	# 变量名 ai 重新赋值为字符串

2. 变量赋值的特殊方法

【例 3-9】 元组赋值可以一次赋多个值。

1	>>> T = 1, 2, 3	# 元组赋值,允许一个变量赋多个值
2	>>> x, y, z = T	# 元组 T 中的值按顺序赋给多个变量
3	>>> T, x, y, z	# 输出元组
	((1, 2, 3), 1, 2, 3)	# T = (1, 2, 3), x = 1, y = 2, z = 3

【例 3-10】 变量链式赋值方法。

1	>>> n = 10; s = '字符'	# 一行多句时,按顺序赋值
2	>>> x = y = k = 0	# 赋值从右到左:k = 0, y = k, x = y
3	>>> x = y = student = {'姓名':'贾宝玉', '年龄':18}	# 允许变量重新赋值,并改变数据类型

【例 3-11】 变量增量赋值方法。

1	>>> x = 1	# x = 1(语义:将 1 赋值给变量名 x)
2	>>> x += 1	# x = 2(语义:将 x + 1 后赋值给 x;等价于 x = x + 1)
3	>>> x *= 3	# x = 6(语义:将 x * 3 后赋值给 x;等价于 x = x * 3)

4	>>> x -= 1	# x=5(语义:将 x-1 后赋值给 x;等价于 x=x-1)
5	>>> x %= 3	# x=2(模运算,语义:将 x 除 3 取余数赋值给 x)
6	>>> s = '世界,'	# s = '世界,'(语义:字符串赋值给变量名 s)
7	>>> s += '你好!'	# s = '世界,你好!'(语义:字符串连接)
8	>>> s *= 2	# s = '世界,你好! 世界,你好!'(语义:字符串重复)

【例 3-12】 变量交换赋值方法,变量交换在排序算法中应用很多。

1	>>> x, y = 10, 20	# 变量序列赋值(x=10,y=20)
2	>>> x, y = y, x	# 变量内容交换,x←y,y←x(x=20,y=10)
3	>>> a, b = '天上', '人间'	# 变量序列赋值(a='天上',b='人间')
4	>>> a, b = b, a	# 字符串变量交换,a←b,b←a(a='人间',b='天上')

3. 赋值语句应用案例

【例 3-13】 求根方法 1: 对方程 $44x^2 + 123x - 54 = 0$ 求根(比较例 3-24、例 3-28)。

案例说明如下。

一元二次方程标准式: $Ax^2 + Bx + C = 0$

一元二次方程判别式: $\Delta = B^2 - 4AC$

解:
$$\begin{cases} \Delta < 0 \text{ 时, 无解;} \\ \Delta = 0 \text{ 时, } x = B/(2A); \\ \Delta > 0 \text{ 时, } x_1 = -((B + \sqrt{\Delta})/(2A)), x_2 = -((B - \sqrt{\Delta})/(2A)). \end{cases}$$

1	>>> delta = 123 ** 2 - 4 * 44 * (-54)	# 符号(如 Δ)用读音代替(如 delta)
2	>>> x1 = (123 + delta ** 0.5)/(-2 * 44)	# 计算方程根: $x_1 = -((B + \sqrt{\Delta})/(2A))$
3	>>> x2 = (123 - delta ** 0.5)/(-2 * 44)	# 计算方程根: $x_2 = -((B - \sqrt{\Delta})/(2A))$
4	>>> print('方程根:x1 = ', x1)	
	方程根:x1 = -3.18123904902659	
5	>>> print('方程根:x2 = ', x2)	
	方程根:x2 = 0.3857845035720447	

程序说明: 程序第 2 行,用指数运算开方,即 $\text{delta} ** 0.5 = \text{delta} ** (1/2) = \sqrt{\text{delta}}$ 。

3.1.3 输入输出语句

输入是用户告诉程序需要的信息,输出是程序运行后告诉用户的结果,通常将输入输出简称为 I/O。Python 输入输出方式有字符串输入输出、图形用户界面输入输出、文件输入输出、网络输入输出等。字符串输入输出函数为 input() 和 print()。

1. 用函数 input() 读取键盘数据

函数 input() 是一个内置标准函数,函数 input() 的功能是从键盘读取输入数据,并返回一个字符串。如果希望输入数据为整数,则需要用函数 int() 将输入的数据转换为整数;如果希望输入数据为浮点数(实数),则需要用函数 float() 将数据转换为浮点数。

【例 3-14】 用函数 input() 读取键盘输入数据。

1	>>> s = input('请输入产品名称:')	# 从键盘读取字符串,赋值给变量 s
	请输入产品名称:计算机	
2	>>> x1 = input('请输入一个整数:')	# 从键盘读取字符串,赋值给变量 x1
	请输入一个整数:105	# 变量 x1 = '105',注意,105 是字符串
3	>>> x2 = int(input('请输入一个整数:'))	# 将输入字符串转换为整数,赋值给 x2
	请输入一个整数:105	# 变量 x2 = 105,注意,105 已转换为整数

4	>>> x3 = float(input('请输入浮点数:')) 请输入浮点数:88.66	# 将输入字符串转换为浮点数,赋值给 x3 # 变量 x3 = 88.66
---	--	--

2. 用函数 eval()读取键盘多个数据

【例 3-15】 用函数 eval()从键盘读取多个数据。

1	>>> a, b, c = eval(input('请输入 3 个数字:')) 请输入 3 个数字:1,2,3	# 同一行输入数据 # 数据之间用逗号分隔
2	>>> x, y = eval(input('数字 x= ')), eval(input('数字 y= ')) 数字 x= 1 数字 y= 2	# 分行输入数据

程序说明: 程序第 1 行,输入数据少于变量个数时,Python 会异常退出。

3. 交互环境的打印输出

【例 3-16】 Python shell 内置了打印功能,没有 print()函数也可以打印输出。

1	>>>'程序' + '设计' '程序设计'	# 打印输出字符串表达式
2	>>> 5200000 + 1314 5201314	# 打印输出数字表达式

4. 用函数 print()打印输出

函数 print()是一个内置的打印输出函数,它不是向打印机输出数据,而是向屏幕输出数据(参见 1.1.3 节)。函数 print()有多个参数,可以用来控制向屏幕的输出格式。

【例 3-17】 用函数 print()打印输出。

1	print(' \u20dd')	# 圆形符号的 UTF8 编码为\u20dd
2	print('~' * 12)	# 连续打印 12 个波纹字符串
3	print('海上生明月,天涯共此时。')	# 打印字符串
	>>> ○ ~~~~~ 海上生明月,天涯共此时。	# 程序输出

【例 3-18】 函数 print()的打印输出方法。

1	>>> love = 5201314	# 变量赋值
2	>>> print('爱情计算值 = ', love) 爱情计算值 = 5201314	# 输出提示信息和变量值
3	>>> print('学号\t 姓名\t 成绩')	# 用制表符'\t'控制输出空格
	学号 姓名 成绩	# 注意,\t 和\n 只有在引号内才起作用
4	>>> print('休言女子非英物\n夜夜龙泉壁上鸣')	# 用换行符'\n'控制换行输出
	休言女子非英物 夜夜龙泉壁上鸣	

5. 用占位符控制输出格式

字符按规定格式输出是一个经常遇到的问题。可以用%表示占位符,有几个%,后面就需要有几个变量顺序对应。常见的占位符有%d(整数)、%f(浮点数)、%s(字符串)、%x(十六进制整数)。不确定用什么占位符,参数%s 永远起作用,它会把任何数据类型转换为字符串。占位符前面的 x 表示占几位,如 10.2f 表示占 10 位,其中小数占 2 位。

【例 3-19】 用占位符控制输出格式。

1	>>> s1 = '布衣中,'; s2 = '问英雄。'	# 定义字符串
2	>>> '%s%s' % (s1, s2)	# 参数 % 为占位符, s 为字符串
	'布衣中,问英雄。'	
3	>>> '您好, %s, 您有 %s 元到账了。' % ('小明', 1000.00)	# 第 1 个 %s 对应 '小明', 其余类推
	'您好, 小明, 您有 1000.00 元到账了。'	
4	>>> print('教室面积是 %10d 平方米。' % (80))	# %10d 为右对齐, 占 10 位
	教室面积是 80 平方米。	
5	>>> print('教室面积是 %10.2f 平方米。' % (80))	# %10.2f 为右对齐, 占 10 位
	教室面积是 80.00 平方米。	# 其中小数占 2 位

6. 利用函数格式化输出

函数 `{:}.format()` 可以格式化输出, 函数用 `{:}` 来代替占位符 `%`, 语法如下。

1	{索引号:[填充字符][对齐][正负][小数位][宽度][类型]}.format(元素)
---	--

参数“索引号”说明 `{:}` 中的参数在 `format()` 中的索引号, 一般为 0; 如果没有则以默认顺序自动分配。如 `{0:*^10}'.format('说明')` 表示索引号为 0, 冒号是分隔符。

参数“填充字符”默认为空格, 如 `{: * ^10}` 表示用 10 个 * 号填充。

参数“对齐”有 `>` (右对齐)、`<` (左对齐)、`^` (居中), 如 `{0:<6}` 为右对齐, 宽度为 6 位。

参数“正负”为是否保留正负号, `+` 为正号, `-` 为负号, 如 `{:+.2f}` 为 2 位小数, 保留十号。

参数“小数位”为指定小数的精确到第几位, 如 `{:.2f}` 为精确到 2 位小数。

参数“宽度”为元素显示宽度位, 如果前面加 0, 则表示用 0 填充。

参数“类型”为指定元素类型, 如 `b` 为二进制, `x` 为十六进制。

返回值: 函数返回格式化后的字符串。

【例 3-20】 函数 `{:}.format()` 的格式化输出方法。

1	>>> '{0:*^10}'.format('说明')	# 0 为索引号, : 为分隔符, * 为填充, ^ 为居中对齐, 10 为宽度
	'**** 说明 ****'	# 注意, 宽度包括元素字符, 一个汉字为一个字符
2	>>> a = '孙悟空'; b = '猪八戒'	
3	>>> '{1}埋怨起{0}来'.format(a, b)	# 索引号 1 对应元素 b, 索引号 0 对应元素 a
	'猪八戒埋怨起孙悟空来'	

【例 3-21】 Python 3.6 以上版本对函数 `print()` 新增了一个 `f` 参数, 它调用函数 `s.format()` 对字符串进行格式化输出, 它可以将大括号内的变量替换为具体值。

1	>>> x = 100; y = 60; s = '优良'	# 变量赋值
2	>>> print(f'考试学生有{x}个, 成绩{s}的有{y}个。')	# 参数 f 为字符串格式化输出
	考试学生有 100 个, 成绩优良的有 60 个。	
3	>>> print(f'浮点数 1/3 的 4 位小数为{1/3:.4f}。')	# 符号 ".4f" 为输出 4 位小数
	浮点数 1/3 的 4 位小数为 0.3333。	

说明: 程序第 2 行与 `print('考试学生有{}个, 成绩{}的有{}个.'.format(x, s, y))` 语句等效。

3.2 选择结构

选择结构由 `if` 语句组成, 它根据条件表达式的值选择程序执行方向。

条件选择语句是判断某个条件是否成立,然后选择执行程序中的某些语句块。与顺序结构比较,选择语句使程序的执行不再完全按照语句的顺序执行,而是根据某种条件是否成立来决定程序执行的走向,它体现了程序具有基本的逻辑判断功能。

条件选择语句有单条件选择结构(见图 3-3)、双条件选择结构(见图 3-4)、多条件选择结构(见图 3-5)。条件选择语句有以下特征。

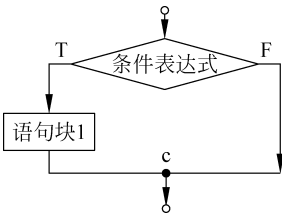


图 3-3 单条件选择结构

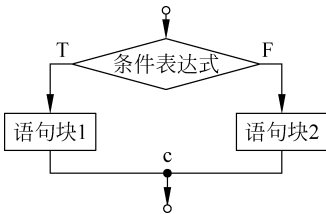


图 3-4 双条件选择结构

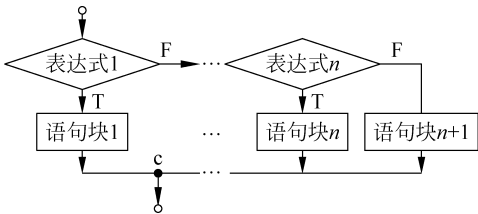


图 3-5 多条件选择结构

(1) 无论条件表达式的值为 T(True),或者为 F(False),一次只能执行一个分支方向的语句块。简单地说,程序不能同时执行语句块 1 和语句块 2。

(2) 无论执行哪一个语句块,都必须能脱离选择结构(见图 3-3~图 3-5 中 c 点)。

3.2.1 单条件选择结构

单条件选择结构很简单,如果条件表达式为真,则执行语句块 1,然后结束条件选择语句;如果条件表达式为假,则直接结束条件选择语句(语句结构见图 3-3)。由此可见,程序代码并不是全部都需要执行,选择语句使得程序具有了很大的灵活性。

【例 3-22】 单条件选择语句简单案例。

1	x = 80	# 变量赋值
2	if x >= 60:	# 条件选择语句,如果 x >= 60 为真
3	print('成绩及格')	# 则执行本语句,执行完后结束 if 语句
	>>>成绩及格	# 程序输出

3.2.2 双条件选择结构

双条件选择结构如图 3-4 所示,双条件选择语句的语法如下。

1	if 条件表达式:	# 条件表达式只允许用关系运算符"==" ;不允许用赋值运算符"="
2	语句块 1	# 如果条件表达式为 True,则执行语句块 1,执行完后结束 if 语句
3	else:	# 否则
4	语句块 2	# 如果条件表达式为 False,则执行语句块 2,执行完后结束 if 语句

在 if-else 语句中,保留字 if 可以理解为“如果”;条件表达式往往采用关系表达式(如 $x \geq 60$),条件表达式的值只有 True 或者 False;语句结尾的冒号(:)可以理解为“则”;保留字“else:”可以理解为“否则”。注意,if 语句后的冒号(:)不可省略,if 语句块可以有多行,但是 if 语句块内部必须缩进 4 个空格,并且保持垂直对齐。

Python 根据条件表达式的值为 True 还是为 False,来决定怎样执行 if-else 语句中的代码。如果条件表达式的值为 True,Python 就执行 if 语句块 1;如果条件表达式的值为 False,Python 将忽略语句块 1,选择执行语句块 2。

【例 3-23】 双条件选择语句简单案例。

1	x = 80	# 变量赋值
2	if x >= 60:	# 如果条件表达式 x >= 60 为真,则执行语句 3
3	print('成绩及格')	# 执行完本语句后,结束 if 语句(不执行语句 4 和语句 5)
4	else:	# 否则执行语句 5(x >= 60 为假时,跳过语句 3)
5	print('成绩不及格')	# 执行完本语句后,结束 if 语句
	>>>成绩及格	# 程序输出

程序说明:双条件选择程序结构适用于二选一的应用场景,两个语句块中总有一个语句块会被执行。如果有多种选择的要求,应当采用多条件选择语句结构。

【例 3-24】 求根方法 2:对方程 $44x^2 + 123x - 54 = 0$ 求根(比较例 3-13、例 3-28)。

案例分析:Python 可以用函数 input()读取用户输入信息,但是 Python 默认将输入信息保存为字符串形式。所以,需要用函数 float()将输入信息强制转换为浮点数类型,这样在计算时才可以避免出现错误。

1	import math	# 导入标准模块——数学计算
2		
3	print('请输入方程 A * x ** 2 + B * x + C = 0 的系数:')	
4	a = float(input('二次项系数 A = '))	# 将输入数据转换为浮点数
5	b = float(input('一次项系数 B = '))	
6	c = float(input('常数项系数 C = '))	
7	p = b * b - 4 * a * c	# 计算方程判别式
8	if p < 0:	# 如果判别式小于 0
9	print('方程无解')	
10	exit()	# 函数 exit()为退出程序
11	else:	
12	x1 = (-b + math.sqrt(p)) / (2 * a)	# 计算方程根 1
13	x2 = (-b - math.sqrt(p)) / (2 * a)	# 计算方程根 2
14	print(f'x1 = {x1} x2 = {x2}')	# f 参数为格式化输出
	>>>	# 程序输出
	请输入方程 A * x ** 2 + B * x + C = 0 的系数:	
	二次项系数 A = 44	
	一次项系数 B = 123	
	常数项系数 C = -54	
	x1 = 0.3857845035720447 x2 = -3.18123904902659	

三元运算是有三个操作数(左表达式值、条件表达式值、右表达式值)的程序语句。程序中经常用三元运算进行条件赋值,三元运算语法如下。

1	a = 左表达式值 if 条件表达式值 else 右表达式值
---	--------------------------------

三元运算语句中,首先进行条件判断,条件表达式的值为 True 时,将左表达式的值赋给变量;条件表达式的值为 False 时,将右表达式的值赋给变量。

【例 3-25】 条件选择语句的三元运算。

1	score = 80	# 变量赋值
2	s = '及格' if score >= 60 else '不及格'	# 左表达式值为'及格',条件表达式值为 True
3	print('三元运算结果:', s)	# 右表达式值为'不及格'
	>>>三元运算结果: 及格	# 程序输出

3.2.3 多条件选择结构

1. 多条件选择语句 if-elif

当条件选择有多个项目时,可以使用多条件选择语句 if-elif。elif 是 else if 的缩写,语句结构如图 3-5 所示,多条件选择语句的语法如下。

1	if 条件表达式 1:	# 判断条件表达式 1
2	语句块 1	# 若条件表达式 1 为 True 则执行语句块 1,执行完后结束 if-elif 语句
3	elif 条件表达式 2:	# 若条件表达式 1 不满足,则继续判断条件表达式 2
4	语句块 2	# 若条件表达式 2 为 True 则执行语句块 2,执行完后结束 if-elif 语句
5	elif 条件表达式 3:	# 若条件表达式 2 也不满足,则继续判断条件表达式 3
6	语句块 3	# 若条件表达式 3 为 True 则执行语句块 3,执行完后结束 if-elif 语句
7	else:	# 若条件表达式 1、2、3 都不满足(注意,此处没有条件表达式)
8	语句块 4	# 执行语句块 4,执行完后结束 if-elif 语句

以上语法中,if、elif 都需要写条件表达式,但是 else 不需要写条件表达式;else、elif 需要与 if 一起使用。注意,if、elif、else 行尾都有英文冒号(;)。

if-elif 语句从上往下判断,如果某个条件判断为 True,将该条件选择对应的语句块执行完后,忽略剩下的 elif 和 else 语句块,结束 if-elif 语句,即一次只执行一个分支。

在单数据多条件选择编程时,开关语句 switch-case 比 if-elif 结构更简洁清晰。大多数程序语言都提供了 switch-case 条件选择语句或者极其相似的语句,如 C/C++、Java、Go 等静态语言都支持 switch-case 语句结构;Ruby、Shell、Perl 语言中,也有类似的 case-when 结构。其实在 PEP-275(2001 年)、PEP-3103(2006 年)、PEP-622(2020 年)等标准中,讨论过引入 switch 开关语句,提出过几个 switch 语句的基本结构,然而核心开发者们似乎没有达成一致共识,最终导致提案流产。

2. 多条件选择语句 if-elif 应用案例

【例 3-26】 BMI(Body Mass Index,体重指数)是国际上常用衡量人体健康程度的指标(见表 3-1),过胖和过瘦都不利于身体健康。

表 3-1 BMI 参考标准

BMI 分类	世界卫生组织标准	中国参考标准	相关疾病发病的危险性
体重过低	BMI<18.5	BMI<18.5	低(其他疾病危险性增加)
正常范围	18.5≤BMI<25	18.5≤BMI<24	平均水平
超重	BMI≥25	BMI≥24	增加
肥胖前期	25≤BMI<30	24≤BMI<28	增加

续表

BMI 分类	世界卫生组织标准	中国参考标准	相关疾病发病的危险性
I 度肥胖	$30 \leq \text{BMI} < 35$	$28 \leq \text{BMI} < 30$	中度增加
II 度肥胖	$35 \leq \text{BMI} < 40$	$30 \leq \text{BMI} < 40$	严重增加
III 度肥胖	$\text{BMI} \geq 40.0$	$\text{BMI} \geq 40.0$	非常严重增加

BMI 计算方法为： $\text{BMI} = \text{体重}(\text{kg}) / \text{身高}^2(\text{m})$

案例分析：编程难点在于同时输出国际和国内对应的 BMI 分类。我们可以先对 BMI 指标进行分类，合并相同项，列出差别项，然后利用 if-elif 语句进行编程。

1	height = eval(input('请输入您的身高(米):'))	# 输入数据
2	weight = eval(input('请输入您的体重(公斤):'))	# 输入数据
3	BMI = weight / (height ** 2)	# 计算 BMI 值
4	print('您的 BMI 为:{:.2f}'.format(BMI))	# 输出 BMI 值
5	who = nat = ''	# who 为国际标准值
6	if BMI < 18.5:	# nat 为国内标准值
7	who, nat = '偏瘦', '偏瘦'	# 根据 BMI 值判断
8	elif 18.5 <= BMI < 24:	# 多重判断
9	who, nat = '正常', '正常'	
10	elif 24 <= BMI < 25:	
11	who, nat = '正常', '偏胖'	
12	elif 25 <= BMI < 28:	
13	who, nat = '偏胖', '偏胖'	
14	elif 28 <= BMI < 30:	
15	who, nat = '偏胖', '肥胖'	
16	else:	
17	who, nat = '肥胖', '肥胖'	
18	print(f'国际 BMI 标准:{who}; 国内 BMI 标准:{nat}')	# 打印结论
	>>> 请输入您的身高(米):1.75 请输入您的体重(公斤):76 您的 BMI 为:24.82 国际 BMI 标准:正常; 国内 BMI 标准:偏胖	# 程序输出

3.2.4 条件选择嵌套结构

在一个 if 语句块中嵌入另外一个 if 语句块称为 if 嵌套结构，语法如下。

1	if 条件表达式 1:	# 开始外层 if - else 语句
2	语句块 1	
3	if 条件表达式 2:	# 开始内层 if - else 嵌套语句
4	语句块 2	
5	else:	
6	语句块 3	# 结束内层 if - else 嵌套语句
7	else:	
8	语句块 4	# 结束外层 if - else 语句

【例 3-27】 用户输入一个整数，判断其是否能够被 2 或者 3 整除。

1	num = int(input('请输入一个整数:'))	
2	if num % 2 == 0:	# if 语句块 1 开始,% 为模运算
3	if num % 3 == 0:	# 内层 if 嵌套语句块 2 开始
4	print('输入的数字可以整除 2 和 3')	
5	else:	# if 语句块 2 部分
6	print('输入的数字可以整除 2,但不能整除 3')	# 内层 if 嵌套语句块 2 结束
7	else:	# if 语句块 1 部分
8	if num % 3 == 0:	# 内层 if 嵌套语句块 3 开始
9	print('输入的数字可以整除 3,但不能整除 2')	
10	else:	# if 语句块 3 部分
11	print('输入的数字不能整除 2 和 3')	# if 语句块 3,语句块 1 结束
	>>> 请输入一个数字:55 输入的数字不能整除 2 和 3	# 程序输出

程序说明：第 2~11 行为 if 语句块 1；第 3~6 行为 if 语句块 2；第 8~11 行为 if 语句块 3。条件嵌套结构的语句块难于理解,并且增加了程序测试的复杂度,应当尽量避免。

【例 3-28】 求根方法 3：对方程 $44x^2 + 123x - 54 = 0$ 求根(比较例 3-13、例 3-24)。

1	import math	# 导入标准模块——数学计算
2		
3	a = float(input('请输入二次项系数 a:'))	# 将输入信息转换为浮点数
4	b = float(input('请输入一次项系数 b:'))	
5	c = float(input('请输入常数项系数 c:'))	
6	if a != 0:	# 如果判别式不等于 0
7	delta = b**2 - 4 * a * c	# 则计算方程判别式
8	if delta < 0:	# 如果判别式小于 0
9	print('无根')	
10	elif delta == 0:	# 如果判别式等于 0
11	s = -b/(2 * a)	# 则计算方程唯一根
12	print('唯一根 x = ', s)	
13	else :	
14	root = math.sqrt(delta)	# 计算方程判别式
15	x1 = (- b + root)/(2 * a)	# 计算方程根 1
16	x2 = (- b - root)/(2 * a)	# 计算方程根 2
17	print(f'x1 = {x1} x2 = {x2}')	# f 参数为格式化输出
	>>> 请输入二次项系数 a:44 请输入一次项系数 b:123 请输入常数项系数 c: - 54 x1 = 0.3857845035720447 x2 = - 3.18123904902659	# 程序输出

3.3 循环结构

循环结构是重复执行一个或几个语句块(循环体),直到满足某一条件为止。循环是程序迭代的实现方法。Python 有两种基本循环结构：一种是计数循环,它用于循环次数确定的情况；另一种是条件循环,它用于循环次数不确定的情况。

3.3.1 计数循环

计数循环有序列循环(见图 3-6)、迭代器循环(见图 3-7)和列表推导式三种形式。

1. 序列循环

(1) 序列循环语法。序列可以是字符串、列表、元组、字典、集合等;迭代变量用于接收序列中取出的元素。序列循环的功能是将序列中的元素逐个取出,每循环一次,迭代变量会自动接收序列中的一个元素,这相当于给迭代变量循环赋值,序列中最后一个元素取完后自动结束循环。序列循环的语法如下。

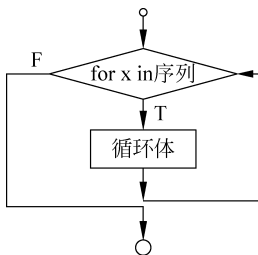


图 3-6 序列循环结构

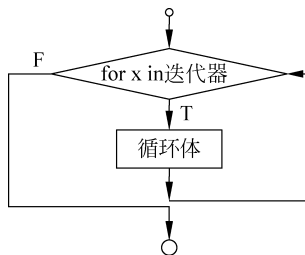


图 3-7 迭代器循环结构

1	for 迭代变量 in 序列;	# 将序列中每个元素逐个代入迭代变量
2	循环体	

说明: 迭代变量是临时变量,它的命名比较自由,如 x、i、n、k、s、_(下画线)等。

(2) 序列循环案例。

【例 3-29】 列表为['唐僧','孙悟空','猪八戒','沙和尚'],逐个输出列表中的元素。

1	names = ['唐僧', '孙悟空', '猪八戒', '沙和尚']	# 定义列表(序列)
2	for x in names:	# 列表 names 中的元素逐个代入迭代变量 x
3	print(x, end = '')	# 执行循环体,参数 end = ''为不换行
>>>唐僧 孙悟空 猪八戒 沙和尚		# 程序输出

【例 3-30】 对字符串中的元素,用打字机效果逐个输出。

1	import sys	# 导入标准模块——系统
2	from time import sleep	# 导入标准模块——睡眠函数
3		
4	poem = '''\	# 加\不空行输出;删除\空 1 行输出
5	pain past is pleasure.	# 定义字符串
6	过去的痛苦就是快乐。'''	
7	for char in poem:	# 把 poem 中元素逐个代入迭代变量 char
8	sleep(0.2)	# 睡眠 0.2 秒(用暂停形成打字机效果)
9	sys.stdout.write(char)	# 输出字符串
>>> pain past is pleasure...		# 程序输出(略)

(3) 序列循环执行过程。

步骤 1: 循环开始时,语句 for 内部计数器自动设置索引号=0,并读取序列(如列表等)中 0 号元素,如果序列为空,则循环自动结束并退出循环;

步骤 2: 如果序列不为空,则 for 语句读取指定元素,并将它复制到迭代变量;

- 步骤 3: 执行循环体内语句块,循环体执行完后,一次循环执行完毕;
- 步骤 4: for 语句内部计数器将索引号自动加 1,继续访问下一个元素;
- 步骤 5: for 语句自动判断,如果序列中存在下一个元素,重复执行步骤 2~5;
- 步骤 6: 如果序列中已经没有元素了,则 for 语句会自动退出当前循环语句。

2. 迭代器循环

(1) 整数序列生成函数。函数 range()用于生成顺序整数序列,语法如下。

1	range(起始值,终止值,步长)
---	-------------------

参数“起始值”默认从 0 开始,如 range(5)等价于 range(0,5),共 5 个元素。
参数“终止值”不包括本身,range()遵循“左闭右开”原则(取头不取尾)。
参数“步长”即每个整数的增量,默认为 1,如 range(5)等价于 range(0,5,1)。
返回值: 函数 range()返回一个[顺序整数列表]。

(2) 函数 range()作为 for 循环的迭代器(生成整数序列),语法如下。

1	for 迭代变量 in range(参数):	# 将迭代器中的元素逐个赋值给“迭代变量”
2	循环体	

【例 3-31】 一个球从 100 米高度自由落下,每次落地后反弹到原高度的一半再落下,球一共反弹了 10 次。输出每次反弹的高度,并计算球一共经过多少米路程。

案例分析: 设球反弹高度为 high,每次为原高度的一半(high/2),总高度(total)为 10 次累计反弹高度。利用迭代循环将 10 次反弹高度进行累加(total += high)。

1	high = 100	# 高度赋值
2	total = 0	# 总共经过路程
3	for i in range(10):	# 注意,range(10)一共循环 10(0~9)次
4	high /= 2	# 计算反弹高度(反弹高度每次除以 2)
5	total += high	# 计算路程和(等价于 total = total + high)
6	print('球第', i + 1, '次反弹高度:', high)	# 注意,索引号 i 从 0 开始,因此需要 + 1
7	print('球总共经过的路程为[米]:', total)	
	>>>球第 1 次反弹高度: 50.0...	# 程序输出(略)

(3) 遍历序列函数。函数 enumerate()也可以作为 for 循环的迭代器,函数的基本功能是遍历一个序列,它通过迭代器来实现,内存使用量很低,运行速度快。函数语法如下。

1	enumerate(序列, 索引号)
---	--------------------

参数“序列”可以是列表、字符串、集合等。
参数“索引号”默认从 0 开始,也可以从指定编号开始(没有意义,见例 3-32)。
返回值: 函数返回的迭代变量看上去是两个(如“索引号+元素值”),实际上是一个元组;迭代变量也可以只有一个,如“索引号”或者“元素值”。

【例 3-32】 用函数 enumerate()作为迭代器,循环打印列表元素。

1	lst = ['智者', '乐水', '仁者', '乐山']	# 定义列表
2	for x,y in enumerate(lst, 5):	# 两个迭代变量 x 和 y,索引号从 5 开始
3	print(f'{x}:{y}')	# 打印元素序列
	>>> 5:智者 6:乐水 7:仁者 8:乐山	# 程序输出

3. 列表推导式

列表推导式是将输出表达式(新列表)、for 循环和 if 条件表达式封装在一个列表语句中的方法,它可以对源列表做映射或过滤等操作。列表推导式语法如下。

1	变量名 = [输出表达式 for 迭代变量 in 列表]	# 语法 1
2	变量名 = [输出表达式 for 迭代变量 in 列表 if 条件表达式]	# 语法 2

【例 3-33】 利用列表推导式生成一个等差数列列表。

1	>>> lst = [1, 2, 3, 4, 5, 6, 7, 8]	# 定义源列表
2	>>> lst = [i * i for i in lst]	# 用源列表生成一个平方后的新列表(映射)
3	>>> lst	# 输出新列表
	[1, 4, 9, 16, 25, 36, 49, 64]	

【例 3-34】 成绩为[65,82,75,88,90],过滤出 80~90 的成绩(见图 3-8)。

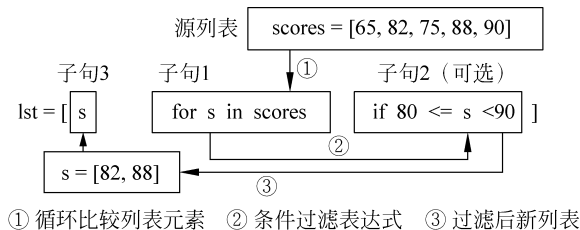


图 3-8 列表推导式执行过程示意图

1	>>> scores = [65, 82, 75, 88, 90]	# 定义成绩列表
2	>>> lst = [s for s in scores if 80 <= s < 90]	# 在[]内的语句为列表推导式(过滤)
3	>>> print('成绩过滤新列表:', lst)	# 打印新列表
	成绩过滤新列表: [82, 88]	

案例分析: 程序第 2 行的列表推导式看上去很复杂,难以理解,如果对复杂语句按子句逐个分解,理解起来就容易多了。如图 3-8 所示,列表推导式可以分为 3 个子句,子句 1 为列表循环,子句 2 为条件表达式(可选),子句 3 为输出表达式,各子句之间是嵌套关系。如图 3-8 所示,列表推导式的执行过程如下。

(1) 执行子句 1,从源列表 scores 中取出第一个元素(65)。

(2) 子句 2 对取出的元素进行条件选择,这时条件值为假,继续循环取第 2 个元素(82);这时条件值为真,将过滤出的元素存入子句 3 的表达式 s 中。

(3) 源列表所有元素循环判断完毕后,将子句 3 中的新列表赋值给变量 lst。

3.3.2 条件循环

条件循环有 while 条件循环(见图 3-9)和 while 永真循环(见图 3-10)两种结构。while 条件循环在运行前先判断条件表达式,若条件表达式的值为 True 则继续循环,若条件表达式的值为 False 则退出循环。while 永真循环则在循环开始设置条件为永真(True),然后在循环体内部设置条件判断语句,若条件表达式值为 True 则退出循环,否则继续循环。while 永真循环广泛应用于事件驱动程序设计和游戏程序设计。

说明: 图 3-10 中,“if 条件表达式”语句可以设置条件为真退出循环,也可以设置条件为假退出循环,这里采用条件为真退出。while 永真循环中,条件为真退出循环设计方法广泛

用于 GUI 程序设计、游戏程序设计等。

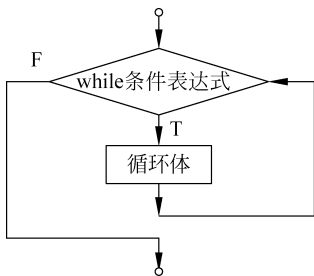


图 3-9 while 条件循环结构

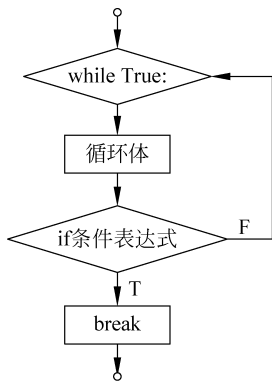


图 3-10 while 永真循环结构

while 条件循环语法如下。

1	while 条件表达式:	# 如果条件表达式为 True,则执行循环体
2	循环体	# 如果条件表达式为 False,则结束循环

while 永真循环语法如下。

1	while True:	# while 永真循环
2	循环体	
3	if 条件表达式:	# 如果条件表达式为 False,则继续循环
4	break	# 如果条件表达式为 True,则强制退出循环

【例 3-35】 用 while 条件循环计算 1~100 的累加和。

1	n = 100	# 定义循环终止条件
2	sum = 0	# sum 存放累加和,初始化变量
3	counter = 1	# 定义计数器变量,记录循环次数,并作为累加增量
4	while counter <= n:	# 循环判断,如果 counter <= n 为 True,则执行下面语句
5	sum = sum + counter	# 累加和(sum 值 + counter 值后,再存入 sum 单元)
6	counter += 1	# 循环计数(counter <= n 为 False 时结束循环)
7	print(f'1 到{n}之和 = {sum}')	# 循环外语句,打印累加和
>>> 1 到 100 之和 = 5050		# 程序输出

【例 3-36】 输出简单正三角形。

1	i = 1	# 变量初始化
2	while i <= 5:	# 循环终止条件
3	print('*' * i)	# 打印 * 号
4	i += 1	# 自加运算
>>>		
* ** *** **** *****		

【例 3-37】 输出简单倒三角形。

1	i = 5	# 变量初始化
2	while i >= 0:	# 循环终止条件
3	print('*' * i)	# 打印 * 号
4	i -= 1	# 自减运算
>>>		
***** **** *** ** *		

3.3.3 中止和跳出循环

1. 循环中止: continue 语句

在程序循环结构中,有时需要跳过循环序列中的某些部分,然后继续进行下一轮循环(注意,不是终止循环)。这时可以用 continue 语句实现这个功能。

【例 3-38】 字符串为“人间四月芳菲尽”,在输出中跳过“芳”字。

1	for s in '人间四月芳菲尽':	# 循环取出字符串中的字符
2	if s == '芳':	# 判断字符串如果是'芳'
3	continue	# 则跳过这个字符,回到循环开始处重新循环
4	print(s,end=" ")	
	>>>人间四月芳菲尽	# 程序输出

2. 强制跳出循环: break 语句

在循环体中,可以用 break 语句强制跳出当前 for 或 while 循环体。break 语句一般与 if 条件选择语句配合使用,在特定条件满足时,达到跳出当前循环体的目的。

【例 3-39】 字符串为“人间四月芳菲尽”,当遇到“芳”字时强制退出循环。

1	for s in '人间四月芳菲尽':	# 循环取出字符串中的字符
2	if s == '芳':	# 判断字符串如果是'芳'
3	break	# 则强制退出循环(比较与例 3-38 的区别)
4	print(s,end=" ")	
	>>>人间四月	# 程序输出

【例 3-40】 字典为{'宝玉': 85,'黛玉': 90,'宝钗': 88},根据输入的姓名(键),查找字典中对应的成绩(值),如果找到对应的成绩则结束程序,否则继续循环查找。

1	dict1 = {'宝玉': 85, '黛玉': 90, '宝钗': 88}	# 定义字典
2	while True:	# 永真循环
3	key = input('请输入用户名:')	# 输入姓名(键)
4	if key in dict1:	# 判断字典中的键
5	print(f'{key}的成绩是:', dict1.get(key))	# 打印查找到的值
6	break	# 强制退出循环
7	else:	# 否则
8	print('您输入的用户名不存在,请重新输入')	
9	continue	# 返回循环开始处
	>>>请输入用户名:	# 程序输出

程序说明: 程序第 5 行,函数 dict1.get(key)为获取字典中键对应的值,如果字典中不存在输入的 key,则返回一个空值(None)。

注意,break 语句是向下强制跳出循环;continue 语句暂停向下执行,继续向上循环。程序中要谨慎使用 break 和 continue 语句:一是它们本质上都是 goto 语句;二是 break 语句造成了循环模块存在两个出口,这增加了程序测试的复杂度。

【例 3-41】 《石头—剪刀—布》游戏中,玩家赢则结束程序,否则游戏循环进行。

1	import random	# 导入标准模块——随机数
2		
3	lst = ['石头', '剪刀', '布']	# 定义字符串列表

4	win = [['布', '石头'], ['石头', '剪刀'], ['剪刀', '布']]	# 定义输赢标准
5	while True:	# while 永真循环
6	computer = random.choice(lst)	# 生成随机数
7	people = input('请输入【石头,剪刀,布】:').strip()	# 玩家输入
8	if people not in lst:	# 判断输赢方
9	people = input('请重新输入【石头,剪刀,布】:').strip()	# 重新输入
10	continue	# 回到循环起始处
11	if computer == people:	# 判断输赢
12	print('平手,再玩一次!')	
13	elif [computer, people] in win:	# 用成员运算判断输赢
14	print('电脑获胜!')	
15	else:	# 否则,玩家获胜
16	print('你获胜啦!')	
17	break	# 玩家获胜则退出循环
	>>>	# 程序输出
	请输入【石头,剪刀,布】= 石头	# 玩家输入"石头"
	平手,再玩一次!	

程序说明:

程序第 8~17 行为条件选择语句块,其中第 15~17 行为强制退出 while 永真循环。程序第 13 行,表达式“[computer, people] in win”为成员运算,它用来判断某一元素(如 [computer, people])是否包含在变量中(如 win)。

3.3.4 程序的循环嵌套

循环嵌套就是一个循环体里面还有另外一个循环体,当两个以上的循环语句相互嵌套时,位于外层的循环结构简称为外循环,位于内层的循环结构简称为内循环。循环嵌套会导致代码的阅读性非常差,因此要避免出现三个以上的循环嵌套。

循环嵌套时,每执行一次外循环,都要进行一遍内循环。例如,读取一个 8 行 5 列的二维表格中的全部数据时,如果用循环遍历的方法读取表格内的数据,外循环 1 次时(读行),则内循环 5 次(读列),读取所有数据一共需要循环 $8 \times 5 = 40$ 次。

【例 3-42】 利用循环嵌套打印乘法口诀表。

案例分析:用外循环控制打印行,内循环控制打印列。乘法口诀表不需要重复打印,因此内循环迭代变量从第 1 列开始,到 $j+1$ 列结束(j 为列数)。

1	for i in range(1, 10):	# 外循环 9 次,打印 9 行(i 行变量)
2	for j in range(1, i+1):	# 内循环打印一行中的列(j 列变量)
3	print(f'{j} × {i} = {i * j} \t', end = '')	# 按格式打印乘法口诀(\t 为空格)
4	print()	# 换行
	>>>	# 程序输出
	1 × 1 = 1	
	1 × 2 = 2 2 × 2 = 4	
	1 × 3 = 3 2 × 3 = 6 3 × 3 = 9	
	1 × 4 = 4 2 × 4 = 8 3 × 4 = 12 4 × 4 = 16	
	1 × 5 = 5 2 × 5 = 10 3 × 5 = 15 4 × 5 = 20 5 × 5 = 25	
	1 × 6 = 6 2 × 6 = 12 3 × 6 = 18 4 × 6 = 24 5 × 6 = 30 6 × 6 = 36	
	1 × 7 = 7 2 × 7 = 14 3 × 7 = 21 4 × 7 = 28 5 × 7 = 35 6 × 7 = 42 7 × 7 = 49	
	1 × 8 = 8 2 × 8 = 16 3 × 8 = 24 4 × 8 = 32 5 × 8 = 40 6 × 8 = 48 7 × 8 = 56 8 × 8 = 64	
	1 × 9 = 9 2 × 9 = 18 3 × 9 = 27 4 × 9 = 36 5 × 9 = 45 6 × 9 = 54 7 × 9 = 63 8 × 9 = 72 9 × 9 = 81	

程序说明:

程序第 1 行,语句 `for i in range(1,10)` 为外循环,它控制行输出,共循环 9 次。

程序第 2 行,语句 `for j in range(1,i+1)` 为内循环,它控制一行中每个表达式(如 $1 \times 1=1$)的输出,由于迭代器为 `range(1,10)`,因此每行最多打印 9 个乘法口诀(9 列)。

程序第 3 行,语句 `print()` 中,参数 `f` 为格式控制; `{j}` 为乘数 1; `×` 为乘号字符; `{i}` 为乘数 2, `{i*j}` 为乘积; `\t` 为水平制表符(即空格); `end=""` 为不换行打印。

程序第 4 行,语句 `print()` 属于外循环,因此语句缩进与第 3 行 `for` 语句对齐,外循环每次循环中,它会执行一次。语句 `print()` 中没有任何参数,它仅起到换行的作用。

【例 3-43】 输出 1~100 的素数。

案例分析:公元前 250 年,古希腊数学家埃拉托色尼(Eratosthenes)提出了一个构造出不超过 n 的素数算法。它基于一个简单的性质:对正整数 n ,如果用 $2 \sim \sqrt{n}$ 的所有整数去除,均无法整除,则 n 为素数。 $\sqrt{n}$ 为内循环次数。

1	<code>import math</code>	# 导入标准模块——数学
2	<code>for n in range(2, 101):</code>	# 外循环,0 和 1 不是素数,n 从 2 开始
3	<code> for j in range(2, round(math.sqrt(n)) + 1):</code>	# 内循环,j 为 $2 \sim \sqrt{n}$ 的整数
4	<code> if n % j == 0:</code>	# 求余运算; $n \% j = 0$ 是合数(能整除)
5	<code> break</code>	# 退出内循环,返回外循环
6	<code> else:</code>	# 语句是 <code>for-else</code> 结构,不是 <code>if-else</code> 结构
7	<code> print('素数为', n)</code>	# $n \% j \neq 0$ 时,说明不能整除,n 为素数
	<code>>>>素数为:2 3 5</code>	# 程序输出(略)(注:输出为竖行)

程序说明:

程序第 3 行,变量 n 为外循环迭代变量;变量 j 为内循环迭代变量,值为 $2 \sim \sqrt{n}$ 的整数;`math.sqrt(n)` 为求 n 的开方值;`round()` 为取整数;`range()` 为定义顺序整数。

程序第 4 行,如果 n 和 j 求余为 0,则 n 不是素数;如果余数不为 0,则 n 为素数。

程序第 6 行,语句 `else` 与第 3 行的 `for` 配套,语句含义是如果循环正常结束,则执行 `else` 中的代码;如果循环中执行了 `break` 语句,则 `else` 中的代码将不再执行。

3.3.5 案例:用 BBP 公式求 π 值

BBP(由算法学家 David Bailey、Peter Borwein、Simon Plouffe 三人的姓名而得名)公式非常神奇,它可以计算圆周率中的任何一位。BBP 公式如下。

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

式中, k 是 π 需要计算的小数位。公式的计算结果是十六进制数。虽然可以将十六进制数转换为十进制数,但是当把它转换为十进制数时,计算结果会被前后位数上的数所影响(二进制数没有影响)。

【例 3-44】 方法 1: 利用 BBP 公式计算 π 值到小数点后第 100 位。

案例分析:为了进行比较,从网络(<http://pai.babihu.com/pi/100.html>)查找到圆周率的 100 位小数准确值为: $\pi=3.141\,592\,653\,589\,793\,238\,462\,643\,383\,279\,502\,884\,197\,169\,399\,375\,105\,820\,974\,944\,592\,307\,816\,406\,286\,208\,998\,628\,034\,825\,342\,117\,067\,9$ 。

1	N = int(input('请输入需要计算到小数点后第 n 位:'))	# 输入计算位数
2	pi = 0	# 初始化变量 pi 值
3	for k in range(N):	# 循环计算 pi 值
4	pi += 1/pow(16,k) * (4/(8 * k + 1) - 2/(8 * k + 4) - 1/(8 * k + 5) - 1/(8 * k + 6))	# BBP 公式
5	print(f'小数点后第 {N} 位的 pi 值为:', pi)	# 打印计算结果
	>>> 请输入需要计算到小数点后第 n 位:100 小数点后第 100 位的 pi 值为: 3.141592653589793	
		# 程序输出

由以上程序输出结果可见, pi 值只能精确到小数点后 15 位。可对程序进行修改, 如例 3-45 所示。

【例 3-45】 方法 2: 利用 BBP 公式计算 π 值到小数点后第 100 位。

案例分析: 对于浮点数的精确计算, 需要用到 Python 标准函数库中的精确计算模块 decimal, 以及计算精度设置函数 getcontext().prec。

1	from decimal import Decimal, getcontext	# 导入标准模块——精确计算函数
2		
3	getcontext().prec = 102	# 设精度为 102 位(很重要)
4	N = int(input('请输入需要计算到小数点后第 n 位:'))	# 输入计算位数
5	pi = Decimal(0)	# 初始化 pi 值为精确浮点数
6	for k in range(N):	# 循环计算 pi 值
7	pi += Decimal(1/pow(16,k) * (4/(8 * k + 1) - 2/(8 * k + 4) - 1/(8 * k + 5) - 1/(8 * k + 6)))	# BBP 公式
8	print(f'小数点后第 {N} 位的 pi 值为:', pi)	# 打印 pi 值
	>>> 请输入需要计算到小数点后第 n 位:100 小数点后第 100 位的 pi 值为: 3.1415926535 8979320958 1689672085 3760862142 8145036074 3119042289 7105864764 2051543596 3184949209 0956344708 9	
		# 程序输出

程序说明:

程序第 3 行, 函数 getcontext().prec 为设置在后续运算中的有效位数。

程序第 5 行, 函数 Decimal(0) 为设置参数 0 的精度为 102 位(默认 28 位)。参数可以为整数或者数字字符串(如 '0'), 但不能是浮点数, 因为浮点数本身就不准确。

程序分析: 从以上计算结果可以看到, 虽然可以计算到 100 位, 但是小数点后第 16 位以后已经与网络查找的 pi 值不相符了。问题出在哪里呢? 检查程序, 可以发现 N 和 k 没有设置浮点数精确计算, N 为循环计数序列, 不设置精确值问题不大; 但是 k 是迭代变量, 如果不设置浮点数精确计算, 将导致浮点数精确计算失败。可对程序进行修改, 如例 3-46 所示。

【例 3-46】 方法 3: 利用 BBP 公式计算 π 值到小数点后第 100 位。

1	from decimal import Decimal, getcontext	# 导入标准模块——精确计算函数
2		
3	getcontext().prec = 102	# 设计算精度为 102 位(很重要)
4	N = int(input('请输入需要计算到小数点后第 n 位:'))	# 输入计算位数
5	pi = Decimal(0)	# 初始化 pi 值为精确浮点数

6	for n in range(N):	# 循环计算 pi 精确值
7	k = Decimal(n)	# 迭代变量为精确浮点数(很重要)
8	pi += Decimal(1/pow(16,k) * (4/(8 * k + 1) - 2/(8 * k + 4) - 1/(8 * k + 5) - 1/(8 * k + 6)))	# BBP 公式
9	print(f'小数点后第{N}位的 pi 值为:', pi)	# 打印 pi 值
>>>		# 程序输出
请输入需要精确到小数点后第 n 位:100		
小数点后第 100 位的 pi 值为: 3. 1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679 8		# 校验计算结果正确

案例分析: 从例 3-44、例 3-45、例 3-46 三个程序, 可以总结出以下经验。

- (1) 一个优秀的程序需要反复调试, 很难一次就设计成功。
- (2) 例 3-45 程序运行正常, 但是程序结果错误, 说明程序的逻辑错误很难发现。
- (3) 高精度浮点运算时, 一定要注意迭代变量的积累误差。

习 题 3

3-1 模块导入有哪些原则?

3-2 说明语句 `import matplotlib.pyplot as plt` 各部分的功能。

3-3 赋值语句应当注意哪些问题?

3-4 编程: 从键盘输入三个随机整数, 将这三个数由小到大排序输出。

3-5 编程: 成绩 ≥ 85 分用“优”表示; 75~84 分用“良”表示; 60~74 分用“及格”表示; 60 分以下用“不及格”表示。从键盘输入一个成绩, 显示成绩的等级。

3-6 编程: 一对兔子从出生后第 3 个月起, 每个月都生一对兔子, 小兔子长到第 3 个月后, 每个月又生一对兔子。假设兔子都不死, 问 9 个月内每月兔子总数为多少?

3-7 编程: “水仙花数”指一个三位数, 各位数字立方和等于该数本身。如 153 是一个“水仙花数”, 因为 $153=1^3+5^3+3^3$ 。编程打印 100~1000 的所有“水仙花数”。

3-8 编程: 用循环嵌套的方法, 打印一个由 * 号组成的 6 行倒三角形。

3-9 编程: 改写例 3-41 的程序, 从键盘输入 1 时, 代替汉字“石头”; 输入 2 时, 代替汉字“剪刀”; 输入 3 时, 代替汉字“布”; 输入 q 时退出游戏。

3-10 编程: 列表为 [55, 85, 73, 94, 42, 88], 将低于 60 的元素用“不及格”代替, 大于或等于 60 的元素保留原值。用列表推导式编程, 输出 ['不及格', 85, 73, 94, '不及格', 88]。