第3章 程序流程控制

扫一扫

视频讲解

在 Python 程序中，对于语句的执行有三种基本的控制结构，即顺序结构、选择结构、循环结构。另外，当程序出错时，Python 使用异常处理流程进行处理（具体请参见第 7 章）。

扫一扫

知识图谱

3.1 程序的流程

3.1.1 输入、处理和输出

无论程序的规模如何，每个程序都可以分为以下三部分：程序通过输入接收待处理的数据（Input）；然后执行相应的处理（Process）；最后通过输出（Output）返回处理的结果。该过程通常称为 IPO 程序编写方法。其示意图如图 3-1 所示。

图 3-1 程序的输入、处理和输出示意图

（1）输入数据。输入是一个程序的开始。程序要处理的数据有多种来源，形成了多种输入方式，包括交互输入、参数输入、随机数据输入、文件输入、网络输入等。

（2）处理数据。处理是程序对输入数据进行计算产生输出结果的过程。计算问题的处理方法统称为“算法”。

（3）输出结果。输出是程序输出结果的方式。程序的输出方式包括控制台输出、图形输出、文件输出、网络输出等。

【例 3.1】 计算球体表面积和体积的程序的 IPO 描述。

```
输入(I):输入 r(球体的半径)
处理(P):计算球体的表面积 s = 4 * math.pi * r * r
        计算球体的体积 v = 4 * math.pi * r * r * r / 3
输出(O):输出 s 和 v
```

3.1.2 算法和数据结构

程序还可以使用以下公式描述：

$$\text{程序} = \text{算法} + \text{数据结构}$$

算法是执行特定任务的方法。数据结构是一种存储数据的方式，有助于求解特定的问题。算法通常与数据结构紧密相关。算法可以描述为“建立一个特定的数据结构，然后采用某种方式使用该数据结构”。

描述算法的最简单方法是使用自然语言描述。对于较复杂的算法，为了描述其细节，往往采用伪代码进行描述。伪代码是一种类似于程序设计语言的文本，其目的是为读者提供在代码中实现算法所需的结构和细节，而无须将算法局限于特定的程序设计语言。

【例 3.2】 求解两个整数最大公约数的算法的自然语言描述。

求解两个整数的最大公约数(Great Common Divisor，GCD)的一种算法是辗转相除法，又称欧几里得算法。辗转相除法算法的自然语言描述如下。

(1) 对于已知的两个正整数 m 和 n，使得 m > n。

(2) m 除以 n 得到余数 r。

(3) 若 r≠0，则令 m←n，n←r，重复步骤(2)，继续 m 除以 n 得到新的余数 r。若仍然 r≠0，则重复此过程，直到 r=0 为止。最后的 m 就是最大公约数。

【例 3.3】 求解两个整数最大公约数的辗转相除算法的伪代码描述。

```
// 求解 m 和 n 的最大公约数。GCD(m, n) = GCD(n, m Mod n)
GCD(m, n)
    While (n != 0)
        remainder = m Mod n        // 计算余数
        m = n
        n = remainder
    End While
    Return m
End GCD
```

说明：伪代码没有特定对应的语言规则。本书采用本例中类似 Python 缩进规则的伪代码描述。

【例 3.4】 求解两个整数最大公约数的辗转相除算法的 Python 代码实现(gcdTest.py)。

```
#求解 m 和 n 的最大公约数。GCD(m, n) = GCD(n, m Mod n)
def Mygcd(m,n):                              #定义函数
    if (m < n):
        m, n = n, m                          #m 和 n 的值交换
    while (n != 0):
        m, n = n, m % n
    return m
if __name__ == '__main__':                   #如果独立运行时，则运行测试代码
    print(24,36,"的最大公约数为:",Mygcd(24,36))
```

程序运行结果如图 3-2 所示。

```
24 36 的最大公约数为: 12
```

图 3-2 两个整数的最大公约数

3.1.3 程序流程图

程序流程图(Flow Chart)又称为程序框图，是描述程序运行具体步骤的图形表示。通过标准符号详细描述程序的输入、处理和输出过程，可以作为程序设计的最基本依据。

流程图的基本元素主要包括以下几种。

(1) 开始框和结束框(⬭)。表示程序的开始和结束。

(2) 输入框/输出框(▱)。表示输入和输出数据。

(3) 处理框(▭)。表示要执行的流程或处理。

(4) 判断框(◇)。表示条件判断，根据判断的结果执行不同的分支。

(5) 箭头线(↓)。表示程序或算法的走向。

【例 3.5】 使用程序流程图(如图 3-3 所示)描述计算所输入数据 a 的平方根的程序。

常用的程序结构包括顺序结构、选择结构和循环结构。本章将陆续展开阐述。

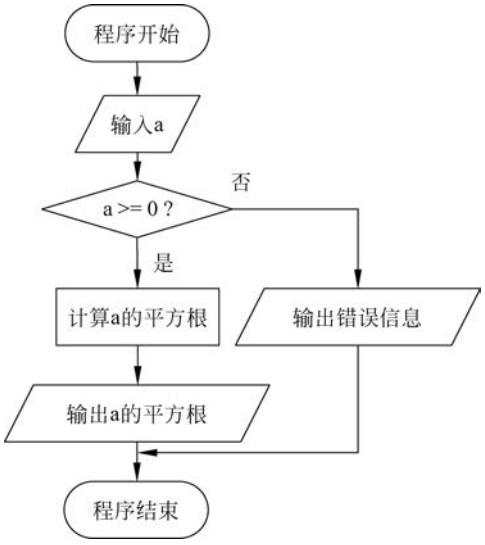


图 3-3 计算所输入数据 a 的平方根的流程图

3.2 顺序结构

若程序中的语句按各语句出现位置的先后次序执行,称之为顺序结构,参见图 3-4。在图 3-4 中先执行语句块 1,再执行语句块 2,最后执行语句块 3,三个语句块之间是顺序执行关系。

【例 3.6】 顺序结构示例(area.py):输入三角形三条边的边长(为简单起见,假设这三条边可以构成三角形),利用海伦公式计算三角形的面积。提示:计算三角形面积的海伦公式为 $s=\sqrt{h*(h-a)*(h-b)*(h-c)}$,其中,a、b、c 是三角形三边的边长,h 是三角形周长的一半。

```
import math                                          # 导入模块
a = float(input("请输入三角形的边长 a:"))             # 提示输入边长 a,并转换为浮点数
b = float(input("请输入三角形的边长 b:"))             # 提示输入边长 b,并转换为浮点数
c = float(input("请输入三角形的边长 c:"))             # 提示输入边长 c,并转换为浮点数
h = (a + b + c) / 2                                  # 三角形周长的一半
area = math.sqrt(h*(h-a)*(h-b)*(h-c))                # 三角形面积
print(str.format("三角形三边分别为:a={0},b={1},c={2}", a, b, c))
print(str.format("三角形的面积 = {0}", area))        # 输出三角形的面积
```

程序运行结果如图 3-5 所示。

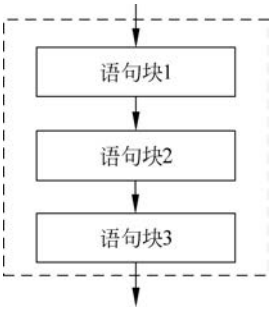


图 3-4 顺序结构示意图

```
请输入三角形的边长a: 3
请输入三角形的边长b: 4
请输入三角形的边长c: 5
三角形三边分别为: a=3.0, b=4.0, c=5.0
三角形的面积 = 6.0
```

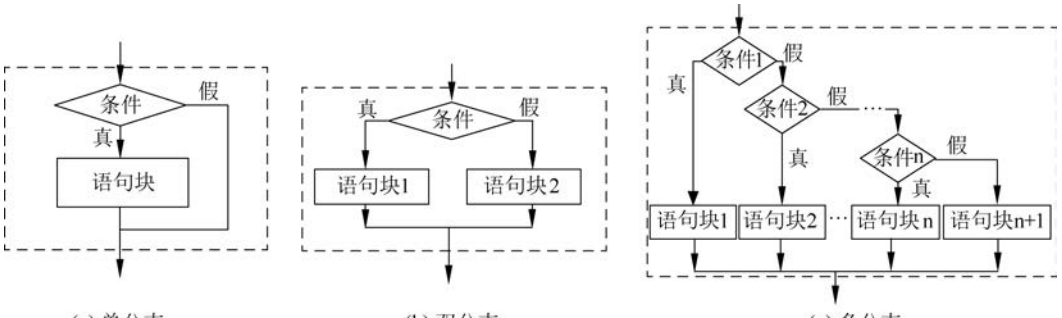
图 3-5 顺序结构求三角形面积的运行结果

3.3 选择结构

选择结构可以根据条件来控制代码的执行分支,也称为分支结构。Python 使用 if 语句来实现分支结构。

3.3.1 分支结构的形式

分支结构包含单分支、双分支和多分支等多种形式，流程如图 3-6(a)～(c)所示。



(a) 单分支　　(b) 双分支　　(c) 多分支

图 3-6　if 语句的选择结构

3.3.2 单分支结构

if 语句单分支结构的语法形式如下。

```
if (条件表达式):
    语句/语句块
```

其中：

(1) 条件表达式：可以是关系表达式、逻辑表达式、算术表达式等。

(2) 语句/语句块：可以是单个语句，也可以是多个语句。多个语句的缩进必须一致。

当条件表达式的值为真(True)时，执行 if 后的语句(块)，否则不做任何操作，控制将转到 if 语句的结束点。其流程如图 3-6(a)所示。

条件表达式可以是任意表达式，其最后评价结果为 bool 值 True(真)或 False(假)。如果表达式的结果为数值类型(0)、空字符串("")、空元组(())、空列表([])、空字典({})，其 bool 值为 False(假)；否则其 bool 值为 True(真)。例如，123、"abc"、(1,2)均为 True。

【例 3.7】　单分支结构示例(if_2desc.py)：输入两个整数 a 和 b，比较两者大小，使得 a 大于 b。

```
a = int(input("请输入第 1 个整数:"))                    # 输入整数 1
b = int(input("请输入第 2 个整数:"))                    # 输入整数 2
print(str.format("输入值:{0}, {1}", a, b))              # 显示所输入的两个整数
if (a < b):                                             # 如果 a 小于 b
    a, b = b, a                                         # a 和 b 交换
print(str.format("降序值:{0}, {1}", a, b))              # 输出降序排序结果
```

程序运行结果如图 3-7 所示。

```
请输入第1个整数: 23
请输入第2个整数: 34
输入值: 23, 34
降序值: 34, 23
```

图 3-7　单分支结构比较两数大小

3.3.3 双分支结构

if 语句双分支结构的语法形式如下。

```
if (条件表达式):
    语句/语句块 1
```

```
else:
    语句/语句块 2
```

当条件表达式的值为真(True)时,执行 if 后的语句(块)1,否则执行 else 后的语句(块)2,其流程如图 3-6(b)所示。

Python 提供了下列条件表达式来实现等价于其他语言的三元条件运算符((条件)? 语句 1: 语句 2)的功能:

```
条件为真时的值  if  (条件表达式) else 条件为假时的值
```

例如,如果 x≥0,则 y=x,否则 y=0,可以表述为:

```
y = x if (x >= 0) else 0
```

【例 3.8】 计算分段函数:$y=\begin{cases}\sin x+2\sqrt{x+e^4}-(x+1)^3 & x\geqslant 0\\ \ln(-5x)-\dfrac{|x^2-8x|}{7x}+e & x<0\end{cases}$

此分段函数有以下几种实现方式,请读者自行编程测试。

(1) 利用单分支结构实现。

```
if (x >= 0):
    y = math.sin(x) + 2 * math.sqrt(x + math.exp(4)) - math.pow(x + 1, 3)
if (x < 0):
    y = math.log(-5 * x) - math.fabs(x * x - 8 * x) / (7 * x) + math.e
```

(2) 利用双分支结构实现。

```
if (x >= 0):
    y = math.sin(x) + 2 * math.sqrt(x + math.exp(4)) - math.pow(x + 1, 3)
else:
    y = math.log(-5 * x) - math.fabs(x * x - 8 * x) / (7 * x) + math.e
```

(3) 利用条件运算语句实现。

```
y = (math.sin(x) + 2 * math.sqrt(x + math.exp(4)) - math.pow(x + 1, 3)) if ((x >= 0)) else \
    (math.log(-5 * x) - math.fabs(x * x - 8 * x) / (7 * x) + math.e)
```

3.3.4 多分支结构

if 语句多分支结构的语法形式如下。

```
if (条件表达式 1):
    语句/语句块 1
elif (条件表达式 2):
    语句/语句块 2
…
elif (条件表达式 n):
    语句/语句块 n
[else:
    语句/语句块 n+1;]
```

该语句的作用是根据不同条件表达式的值确定执行哪个语句(块),其流程如图 3-6(c)所示。

【例 3.9】 已知某课程的百分制分数 mark,将其转换为五级制(优、良、中、及格、不及

格)的评定等级 grade。评定条件如下：

$$成绩等级=\begin{cases}优 & mark\geqslant 90\\ 良 & 80\leqslant mark<90\\ 中 & 70\leqslant mark<80\\ 及格 & 60\leqslant mark<70\\ 不及格 & mark<60\end{cases}$$

根据评定条件，有以下 4 种不同的方法实现。

方法一：

```
mark = int(input("请输入分数: "))
if (mark >= 90):
    grade = "优"
elif (mark >= 80):
    grade = "良"
elif (mark >= 70):
    grade = "中"
elif (mark >= 60):
    grade = "及格"
else:
    grade = "不及格"
```

方法二：

```
mark = int(input("请输入分数: "))
if (mark >= 90):
    grade = "优"
elif (mark >= 80 and mark < 90):
    grade = "良"
elif (mark >= 70 and mark < 80):
    grade = "中"
elif (mark >= 60 and mark < 70):
    grade = "及格"
else:
    grade = "不及格"
```

方法三：

```
mark = int(input("请输入分数: "))
if (mark >= 90):
    grade = "优"
elif (80 <= mark < 90):
    grade = "良"
elif (70 <= mark < 80):
    grade = "中"
elif (60 <= mark < 70):
    grade = "及格"
else:
    grade = "不及格"
```

方法四：

```
mark = int(input("请输入分数: "))
if (mark >= 60):
    grade = "及格"
elif (mark >= 70):
    grade = "中"
elif (mark >= 80):
    grade = "良"
elif (mark >= 90):
    grade = "优"
else:
    grade = "不及格"
```

其中，方法一中使用关系运算符“>=”，按分数从大到小依次比较；方法二和方法三使用关系运算符和逻辑运算符表达完整的条件，即使语句顺序不按比较的分数从大到小依次书写，也可以得到正确的等级评定结果；方法四使用关系运算符“>=”，但按分数从小到大依次比较。

上述四种方法中，方法一、方法二和方法三正确，而且方法一最简捷明了，方法二和方法三虽然正确，但是存在冗余条件。方法四虽然语法没有错误，但是判断结果错误：根据 mark 分数所得等级评定结果只有“及格”和“不及格”两种，请读者根据程序流程自行分析原因。

【例 3.10】 已知坐标点(x,y)，判断其所在的象限(if_coordinate.py)。

```
x = int(input("请输入 x 坐标:"))
y = int(input("请输入 y 坐标:"))
if (x == 0 and y == 0): print("位于原点")
elif (x == 0): print("位于 y 轴")
elif (y == 0): print("位于 x 轴")
elif (x > 0 and y > 0): print("位于第一象限")
elif (x < 0 and y > 0): print("位于第二象限")
elif (x < 0 and y < 0): print("位于第三象限")
else: print("位于第四象限")
```

3.3.5 if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。一般形式如下。

```
if (条件表达式 1):
    if (条件表达式 11):
        语句 1
    [else:
        语句 2]
[else:
    if (条件表达式 21):
        语句 3
    [else:
        语句 4]]
```

内嵌 if

内嵌 if

【例 3.11】 计算分段函数：$y=\begin{cases}1 & x>0\\0 & x=0\\-1 & x<0\end{cases}$

此分段函数有以下几种实现方式，请读者判断哪些是正确的？并自行编程测试正确的实现方式。

```
方法一(多分支结构):
    if (x > 0):
        y = 1
    elif (x == 0):
        y = 0
    else:
        y = -1
```

```
方法二(if 语句嵌套结构):
    if (x >= 0):
        if (x > 0):
            y = 1
        else:
            y = 0
    else:
        y = -1
```

```
方法三:
    y = 1
    if (x != 0):
        if (x < 0):
            y = -1
    else:
        y = 0
```

```
方法四:
    y = 1
    if (x != 0):
        if (x < 0):
            y = -1
        else:
            y = 0
```

请读者画出每种方法相应的流程图，并进行分析测试。其中，方法一、方法二和方法三是正确的，而方法四是错误的。

3.3.6 if 语句典型示例代码

if 语句的典型示例代码如表 3-1 所示。当 if 或 else 的语句块仅包含一条语句时，该语句也可以直接写在关键字 if 或者 else 语句的同一行后面，以实现紧凑代码。

表 3-1 if 语句的典型示例代码

程序功能	代码片段
求绝对值	if a < 0: a = -a
a 和 b 按升序排序	if a > b: a, b = b, a

续表

程 序 功 能	代 码 片 段
求 a 和 b 的最大值	if a > b: maximum = a else: maximum = b
计算两个数相除的余数，如果除数为 0，则给出报错信息	if b == 0: print("除数为 0") else: print("余数为：" + a % b)

计算并输出一元二次方程的两个根。如果判别式 $b^2-4ac<0$，则显示“方程无实根”的提示信息	delta = b * b - 4.0 * a * c
	if delta < 0.0:
	　　print("方程无实根")
	else:
	　　d = math.sqrt(delta)
	　　print((-b + d)/(2.0 * a))
	　　print((-b - d)/(2.0 * a))

3.3.7 选择结构综合举例

【例 3.12】 输入三个数，按从大到小的顺序排序(if_3desc.py)。

先比较 a 和 b，使得 a > b；然后比较 a 和 c，使得 a > c，此时 a 最大；最后 b 和 c 比较，使得 b > c。

```
a = int(input("请输入整数 a:"))
b = int(input("请输入整数 b:"))
c = int(input("请输入整数 c:"))
if (a < b):
    a, b = b, a                          #a 和 b 交换，使得 a > b
if (a < c):
    a, c = c, a                          #a 和 c 交换，使得 a > c
if (b < c):
    b, c = c, b                          #b 和 c 交换，使得 b > c
print("排序结果(降序):", a, b, c)
```

程序运行结果如图 3-8 所示。

【例 3.13】 编程判断某一年是否为闰年(leapyear.py)。判断闰年的条件是年份能被 4 整除但不能被 100 整除，或者能被 400 整除，其判断流程参见图 3-9 所示。

```
请输入整数a: 3
请输入整数b: 2
请输入整数c: 5
排序结果（降序）： 5 3 2
```

图 3-8 三个数降序排序结果

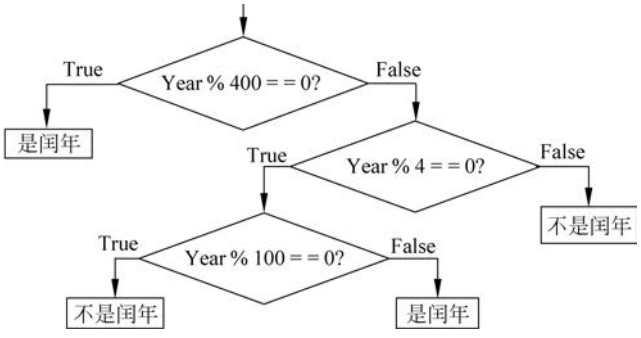


图 3-9 闰年的判断流程

方法一：使用一个逻辑表达式包含所有的闰年条件，相关语句如下。

```
if ((y % 4 == 0 and y % 100 != 0) or y % 400 == 0):
    print("是闰年")
else:
    print("不是闰年")
```

方法二：使用嵌套的 if 语句，相关语句如下。

```
if (y % 400 == 0):
    print("是闰年")
else:
    if (y % 4 == 0):
        if (y % 100 == 0):
            print("不是闰年")
        else:
            print("是闰年")
    else:
        print("不是闰年")
```

方法三：使用 if-elif 语句，相关语句如下。

```
if (y % 400 == 0):
    print("是闰年")
elif (y % 4 != 0):
    print("不是闰年")
elif (y % 100 == 0):
    print("不是闰年")
else:
    print("是闰年")
```

方法四：使用 calendar 模块的 isleap()函数来判断闰年，相关语句如下。

```
if (calendar.isleap(y)):
    print("是闰年")
else:
    print("不是闰年")
```

3.4 循环结构

循环结构用来重复执行一条或多条语句。使用循环结构可以减少源程序重复书写的工作量。许多算法需要使用到循环结构。Python 使用 for 语句和 while 语句来实现循环结构。

3.4.1 可迭代对象

可迭代对象(iterable)一次返回一个元素，因而适用于循环。Python 包括以下几种可迭代对象：序列(sequence)，例如字符串(str)、列表(list)、元组(tuple)等；字典(dict)；文件对象；迭代器对象(iterator)；生成器函数(generator)。

迭代器是一个对象，表示可迭代的数据集合，包括方法__iter__()和__next__()，可以实现迭代功能。

生成器是一个函数，使用 yield 语句，每次产生一个值，也可以用于循环迭代。

有关可迭代对象的详细信息请参见本书 9.8 节内容。

3.4.2 range 对象

在 Python 3 中，内置对象 range 是一个迭代器对象，迭代时产生指定范围的数字序列，其格式如下。

```
range(start, stop[, step])
```

range 返回的数值序列从 start 开始，到 stop 结束（不包含 stop）。如果指定了可选的步长 step，则序列按步长 step 增长。例如：

```
>>> for i in range(1,11): print(i, end = ' ')          #输出:1 2 3 4 5 6 7 8 9 10
>>> for i in range(1,11,3): print(i, end = ' ')        #输出:1 4 7 10
>>> list(range(5, 0, -1))                             #输出:[5, 4, 3, 2, 1]
```

注意：Python 2 中 range 的类型为函数，是一个生成器；Python 3 中 range 的类型为类，是一个迭代器。

3.4.3 for 循环

for 语句用于遍历可迭代对象集合中的元素，并对集合中的每个元素执行一次相关的嵌入语句。当集合中的所有元素完成迭代后，控制传递给 for 之后的下一个语句。for 语句的格式如下。

```
for 变量 in 可迭代对象集合:
    循环体语句/语句块
```

例如：

```
>>> for i in (1,2,3):
        print(i, i**2, i**3)
1 1 1
2 4 8
3 9 27
```

【例 3.14】 利用 for 循环求 1～100 中所有奇数的和以及偶数的和(for_sum1_100.py)。

```
sum_odd = 0; sum_even = 0                          #循环赋初值
for i in range(1, 101):                            #i = 1～100
    if i % 2 != 0:                                 #奇数
        sum_odd += i                               #奇数和
    else:                                          #偶数
        sum_even += i                              #偶数和
print("1～100 中所有奇数的和:", sum_odd)
print("1～100 中所有偶数的和:", sum_even)
```

程序运行结果如图 3-10 所示。

```
1~100中所有奇数的和:  2500
1~100中所有偶数的和:  2550
```

图 3-10 for 循环求和的运行结果

3.4.4 while 循环

与 for 循环一样，while 也是一个预测试的循环，但是 while 在循环开始前并不知道重复执行循环语句序列的次数。while 语句按不同条件执行循环语句（块）零次或多次。while 循环语句的格式如下。

```
while (条件表达式):
    循环体语句/语句块
```

while 循环的执行流程如图 3-11 所示。

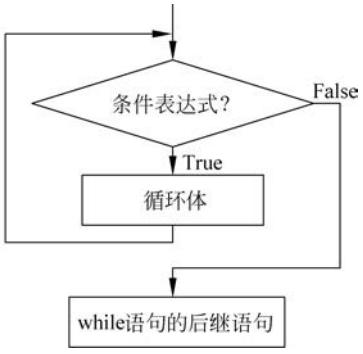


图 3-11 while 循环的执行流程

说明：

(1) while 循环语句的执行过程如下。

① 计算条件表达式。

② 如果条件表达式结果为 True，控制将转到循环语句(块)，即进入循环体。当到达循环语句序列的结束点时转①，即控制转到 while 语句的开始，继续循环。

③ 如果条件表达式结果为 False，退出 while 循环，即控制转到 while 循环语句的后继语句。

(2) 条件表达式是每次进入循环之前进行判断的条件，可以为关系表达式或逻辑表达式，其运算结果为 True(真)或 False(假)。条件表达式中必须包含控制循环的变量。

(3) 循环语句序列可以是一条语句，也可以是多条语句。

(4) 循环语句序列中至少应包含改变循环条件的语句，以使循环趋于结束，避免“死循环”。

【例 3.15】 利用 while 循环求 $\sum_{i=1}^{100} i$，以及 1～100 中所有奇数的和、偶数的和(while_sum.py)。

```
i = 1; sum_all = 0; sum_odd = 0; sum_even = 0
while (i <= 100):
    sum_all += i                          #所有数之和
    if (i % 2 == 0):                      #偶数
        sum_even += i                     #偶数和
    else:                                 #奇数
        sum_odd += i                      #奇数和
    i += 1                                #改变循环条件
print("和 = %d、奇数和 = %d、偶数和 = %d" % (sum_all, sum_odd, sum_even))
```

程序运行结果如图 3-12 所示。

【例 3.16】 用以下近似公式求自然对数的底数 e 的值，直到最后一项的绝对值小于 10^{-6} 为止(while_e.py)。

$$e \approx 1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!}$$

```
i = 1; e = 1; t = 1
while (1/t >= pow(10, -6)):
    t *= i
    e += 1 / t
    i += 1
print("e = ", e)
```

程序运行结果如图 3-13 所示。

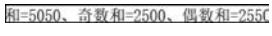


图 3-12 while 循环求和

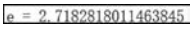


图 3-13 while 循环求自然对数

3.4.5 循环的嵌套

若在一个循环体内又包含另一个完整的循环结构，则称之为循环的嵌套。这种语句结构称为多重循环结构。内层循环中还可以包含新的循环，以形成多层循环结构。

在多层循环结构中两种循环语句（for 循环、while 循环）可以相互嵌套。多重循环的循环次数等于每一重循环次数的乘积。

【例 3.17】 利用嵌套循环打印运行效果如图 3-14 所示的九九乘法表（nest_for.py）。

```
for i in range(1, 10):                          #外循环
    s = ""
    for j in range(1, 10):                      #内循环
        s += str.format("{0:1}*{1:1}={2:<2} ", i, j, i * j)
    print(s)
```

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

图 3-14 九九乘法表运行效果图

思考：请修改程序，分别打印如图 3-15(a)和图 3-15(b)所示的九九乘法表。

```
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
```

(a) 下三角

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

(b) 上三角

图 3-15 九九乘法表的另外两种显示效果

3.4.6 break 语句

break 语句用于退出 for 循环、while 循环，即提前结束循环，接着执行循环语句的后继语句。注意，当多个 for、while 语句彼此嵌套时，break 语句只应用于最里层的语句，即 break 语句只能跳出最近的一层循环。

【例 3.18】 使用 break 语句终止循环（breakTest.py）。

```
while True:
    s = input('请输入字符串(按 Q 或者 q 结束):')
    if s.upper() == 'Q':
        break
    print('字符串的长度为:', len(s))
```

程序运行结果如图 3-16 所示。

```
请输入字符串（按Q或者q结束）：Hello, World!
字符串的长度为： 13
请输入字符串（按Q或者q结束）：您好！
字符串的长度为： 3
请输入字符串（按Q或者q结束）：q
```

图 3-16 break 语句的运行效果

【例 3.19】 编程判断所输入的任意一个正整数是否为素数（prime1.py 和 prime2.py）。

所谓素数（或称质数），是指除了 1 和该数本身，不能

被任何整数整除的正整数。判断一个正整数 m 是否为素数，只要判断 m 可否被 2～$\sqrt{m}$之中的任何一个整数整除，如果 m 不能被此范围中任何一个整数整除，m 即为素数，否则 m 为合数。

方法一（利用 for 循环和 break 语句）：

```
import math                                  #导入模块
m = int(input("请输入一个整数(>1):"))        #提示输入整数,并将字符串转换为整数
k = int(math.sqrt(m))                        #取整数 m 的平方根
for i in range(2, k + 2):                    #判断 m 可否被 2～m 平方根之中任何整数整除
    if m % i == 0:                           #整除运算的余数为 0,表示可以整除
        break                                #可以整除,肯定不是素数,结束循环
if i == k+1 :                                #m 不能被 2～m 平方根之中的任何一个整数整除
    print(m, "是素数!")                      #m 是素数
else:                                        #m 可以被 2～m 平方根之中的某个整数整除
    print(m, "是合数!")                      #m 是合数
```

方法二（利用 while 循环和 bool 变量）：

```
import math                                  #导入模块
m = int(input("请输入一个整数(>1):"))        #提示输入整数,并将字符串转换为整数
k = int(math.sqrt(m))                        #取整数 m 的平方根
flag = True                                  #先假设所输入的整数为素数
i = 2                                        #while 循环赋初值
while (i <= k and flag == True):             #判断循环条件
    if (m % i == 0):                         #整除运算余数为 0,表示可以整除
        flag = False                         #可以整除,肯定不是素数,结束循环
    else:
        i += 1                               #循环计数值加 1,继续循环
if (flag == True):                           #素数判断标志一直为 True
    print(m, "是素数!")                      #m 是素数
else:                                        #素数判断标志被修改为 False
    print(m, "是合数!")                      #m 是合数
```

3.4.7 continue 语句

continue 语句类似于 break，也必须在 for、while 循环中使用。但它结束本次循环，即跳过循环体内自 continue 下面尚未执行的语句，返回到循环的起始处，并根据循环条件判断是否执行下一次循环。

continue 语句与 break 语句的区别在于：continue 语句仅结束本次循环，并返回到循环的起始处，循环条件满足的话就开始执行下一次循环；而 break 语句则是结束循环，跳转到循环的后继语句执行。

与 break 语句相类似，当多个 for、while 语句彼此嵌套时，continue 语句只应用于最里层的语句。

【例 3.20】 使用 continue 语句跳过循环（continue_score.py）。要求输入若干学生成绩（按 Q 或 q 结束），如果成绩<0，则重新输入。统计学生人数和平均成绩。程序运行结果如图 3-17 所示。

```
num = 0; scores = 0;                         #初始化学生人数和成绩和
while True:                                  #循环条件一直为真
```

```
    s = input('请输入学生成绩(按 Q 或 q 结束):')    #提示输入成绩
    if s.upper() == 'Q':                          #输入了 Q 或 q
        break                                     #跳出循环
    if float(s) < 0:                              #成绩必须>=0
        continue                                  #跳转到循环开始,继续判断循环条件
    num += 1                                      #统计学生人数
    scores += float(s)                            #汇总成绩
#输出学生人数和平均成绩
print('学生人数为:{0},平均成绩为:{1}'.format(num,scores / num))
```

【例 3.21】 显示 100～200 中不能被 3 整除的数(continue_div3.py)。要求一行显示 10 个数。程序运行结果如图 3-18 所示。

```
j = 0                                             #控制一行显示的数值个数
print('100～200 不能被 3 整除的数为:')
for i in range(100, 200 + 1):                     #100～200 循环
    if (i % 3 == 0): continue                     #跳过能被 3 整除的数
    print(str.format("{0:<5}",i), end = "")       #每个数占 5 个位置,不足后面加空格,并且
                                                  #不换行
    j += 1                                        #一行输出的数值个数加 1
    if (j % 10 == 0): print()                     #一行显示 10 个数后换行
```

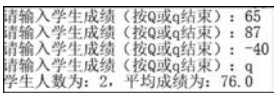


图 3-17 continue 语句的运行效果

图 3-18 显示 100～200 不能被 3 整除的数

3.4.8 死循环(无限循环)

如果 while 循环结构中循环控制条件一直为真,则循环将无限继续,程序将一直运行下去,从而形成死循环。

程序死循环时,会造成程序没有任何响应;或者造成不断输出(例如控制台输出,文件写入,打印输出等)。

在程序的循环体中,插入调试输出语句 print,可以判断程序是否为死循环。

注意:有的程序算法十分复杂,可能需要运行很长时间,但并不是死循环。

在大多数计算机系统中,用户可以使用快捷键 Ctrl+C 中止当前程序的运行。

【例 3.22】 死循环示例(infinite.py)。

```
import math                                       #导入模块
while True:                                       #循环条件一直为真
   num = float(input("请输入一个正数:"))          #提示输入一个正数(浮点数)
   print(str(num), "的平方根为:", math.sqrt(num))  #输出正数的平方根
print("Good bye!")
```

本程序因为循环条件为“while True”,所以将一直重复提示用户输入一个正数,计算并输出该数的平方根,从而形成死循环。所以,最后一句“print("Good bye!")”语句将没有机会执行。

3.4.9 else 子句

for 语句和 while 语句都可以附带一个 else 子句(可选)。如果 for、while 语句没有被 break 语句中止,则会执行 else 子句,否则不执行。其语法如下。

```
for 变量 in 可迭代对象集合:
    循环体语句(块)1
else:
    语句(块)2
```

或者:

```
while (条件表达式):
    循环体语句(块)1
else:
    语句(块)2
```

【例 3.23】 使用 for 语句的 else 子句(for_else.py)。程序运行结果如图 3-19 所示。

```
hobbies = ""                                        #空字符串
for i in range(1, 3 + 1):                           #循环 3 次
    s = input('请输入爱好之一(最多三个,按 Q 或 q 结束):')   #提示输入爱好
    if s.upper() == 'Q':                            #输入 Q 或 q,结束循环
        break                                       #跳出 for 循环,执行其后继语句
    hobbies += s + ''                               #(爱好)字符串拼接
else:                                               #for 语句的 else 子句
    print('您输入了三个爱好.')
print('您的爱好为:', hobbies)
```

3.4.10 enumerate()函数和循环

图 3-19 else 子句的运行结果

Python 语言的 for 循环直接迭代对象集合中的元素,如果需要在循环中使用索引下标访问集合元素,则可以使用内置的 enumerate()函数。

enumerate()函数用于将一个可遍历的数据对象(例如列表、元组或字符串)组合为一个索引序列,并返回一个可迭代对象,所以在 for 循环当中可以直接迭代下标和元素。

【例 3.24】 enumerate()函数和下标元素循环示例(enumerateTest.py)。程序运行结果如图 3-20 所示。

```
seasons = ['Spring', 'Summer', 'Autumn', 'Winter']      #一年四季的列表清单
for i, s in enumerate(seasons, start = 1):              #start 默认从 0 开始,现指定从 1 开始
    print("第{0}季节:{1}".format(i, s))
```

3.4.11 zip()函数和循环

图 3-20 enumerate()函数的运行结果

如果需要并行遍历多个可迭代对象,则可以使用 Python 内置函数 zip()。

zip()函数将多个可迭代的对象中对应的元素打包成一个个元组,然后返回一个可迭代对象。如果元素的个数不一致,则返回可迭代对象的长度与最短的对象相同。

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zip(x, y)                                  #输出:<zip object at 0x000001A68BE80900>
>>> list(zip(x, y))                           #输出:[(1, 4), (2, 5), (3, 6)]
```

【例 3.25】 zip 函数和并行循环示例(zipTest.py)。程序运行结果如图 3-21 所示。

```
evens = [0, 2, 4, 6, 8]                       #偶数
odds = [1, 3, 5, 7, 9]                        #奇数
for eache,eacho in zip(evens, odds):          #将多个可迭代对象中对应的元素打包
    print("{0}*{1}={2}".format(eache, eacho, eache*eacho))
```

0*1=0
2*3=6
4*5=20
6*7=42
8*9=72

图 3-21 zip 函数的运行结果

3.4.12 map()函数和循环

如果需要遍历可迭代对象,并使用指定函数处理对应的元素,则可以使用 Python 内置函数 map()。

map(func, seq1[, seq2,…])函数将 func 作用于 seq 中的每一个元素,并将所有的调用结果作为可迭代对象返回。如果 func 为 None,作用等同于 zip()函数。

例如,如果要返回列表中每个字符串的长度,可以使用内置的 map()函数和 len()函数。

```
>>> map(len, ['Spring', 'Summer', 'Fall', 'Winter'])
<map object at 0x0000021BA2716CB0>
>>> list(map(len, ['Spring', 'Summer', 'Fall', 'Winter']))        #输出:[6, 6, 4, 6]
```

【例 3.26】 map 函数和循环示例。

```
>>> list(map(abs, [-1, 0, 7, -8]))          #计算绝对值.输出:[1, 0, 7, 8]
>>> list(map(pow, range(5), range(5)))      #计算乘幂.输出:[1, 1, 4, 27, 256]
>>> list(map(ord,'abcdef'))                 #计算 ASCII 码.输出:[97, 98, 99, 100, 101, 102]
>>> list(map(lambda x,y:x+y,'abc','de'))    #字符串拼接.输出:['ad', 'be']
```

3.4.13 循环语句典型示例代码

使用 for 语句和 while 语句都能实现循环功能,具体选择哪种语法构造取决于程序员的偏好。循环语句的典型示例如表 3-2 所示。

表 3-2 for 语句和 while 语句的典型示例

功 能 示 例	实 现 代 码

输出 n 个数(0～n－1)的 2 的乘幂的值列表	power = 1
	for i in range(n):
	  print(str(i) + " " + str(power))
	  power *= 2

输出小于或等于 n 的最大的 2 的乘幂的值	power = 1 while 2*power <= n: power *= 2 print(power)
计算并输出 1+2+…+n 的累加和	total = 0 for i in range(1, n+1): total += i print(total)

续表

功 能 示 例	实 现 代 码
计算并输出 n 的阶乘(n! = 1×2×…×n)	factorial = 1 for i in range(1, n+1): factorial *= i print(factorial)

输出半径为 1～n 的圆的周长列表	for r in range(1, n+1):
	  print("r = " + str(r), end = " ")
	  print("p = " + str(2.0 * math.pi * r))

3.4.14 循环结构综合举例

【例 3.27】 使用牛顿迭代法求解平方根(sqrtNewton.py)。运行效果如图 3-22 所示。

计算一个正实数 a 的平方根,可以使用牛顿迭代法实现:首先假设 t=a,开始循环,如果 t=a/t(或小于容差),则 t 等于 a 的平方根,循环结束并返回结果;否则,将 t 和 a/t 的平均值赋值给 t,继续循环。

```
EPSILON = 1e-15                          #容差
a = float(input("请输入正实数 a:"))       #正实数 a
t = a                                    #假设平方根 t=a
while abs(t - a/t) > (EPSILON * t):
    t = (a/t + t) / 2.0                  #将 t 和 a/t 的平均值赋值给 t
print(t)                                 #输出 a 的平方根
```

【例 3.28】 显示 Fibonacci 数列(for_fibonacci.py):1、1、2、3、5、8、……的前 20 项。

$$即\begin{cases} F_1=1 & n=1 \\ F_2=1 & n=2 \\ F_n=F_{n-1}+F_{n-2} & n>=3 \end{cases}$$

要求每行显示 4 项。运行效果如图 3-23 所示。

图 3-22 使用牛顿迭代法求解平方根

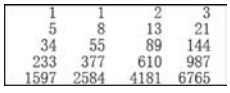


图 3-23 显示 Fibonacci 数列

相关语句如下。

```
f1 = 1; f2 = 1                                          #赋初值
for i in range(1, 11):                                  #i=1～10 循环
    print(str.format("{0:6}{1:6}", f1, f2), end = " ")  #每次输出两个数,每个数占 6 位,
                                                        #空格分隔
    if i % 2 == 0: print()                              #显示 4 项后换行
    f1 += f2; f2 += f1
```

3.5 综合应用:turtle 模块的复杂图形绘制

3.5.1 绘制正方形(改进版)

【例 3.29】 修改例 2.36 的代码,使用循环结构绘制正方形。

```
import turtle                          #导入 turtle 模块
p = turtle.Turtle()                    #创建海龟对象
p.color("red")                         #设置绘制时画笔的颜色
p.pensize(3)                           #定义绘制时画笔的线条宽度
turtle.speed(1)                        #定义绘图的速度("slowest"或者 1)
p.goto(0,0)                            #移动海龟到坐标原点(0,0)
for i in range(4):                     #绘制正方形的四条边
    p.forward(100)                     #向前移动 100
    p.right(90)                        #向右旋转 90 度
```

3.5.2 绘制圆形螺旋

【例 3.30】 使用海龟绘图分别绘制红、蓝、绿、黄四种颜色的圆形螺旋(spiral.py)。程序运行最终结果如图 3-24 所示。

```
import turtle                                    #导入 turtle 模块
p = turtle.Turtle()                              #创建海龟对象
p.speed(0)                                       #定义绘图的速度("fastest"或者 0 均表示最快)
colors = ["red", "blue", "green", "yellow"]      #红、蓝、绿、黄四种颜色
for i in range(100):                             #i = 0～99
    p.pencolor(colors[i%4])                      #设置画笔颜色(红或蓝或绿或黄)
    p.circle(i)                                  #画圆
    p.left(91)                                   #向左旋转 91 度
```

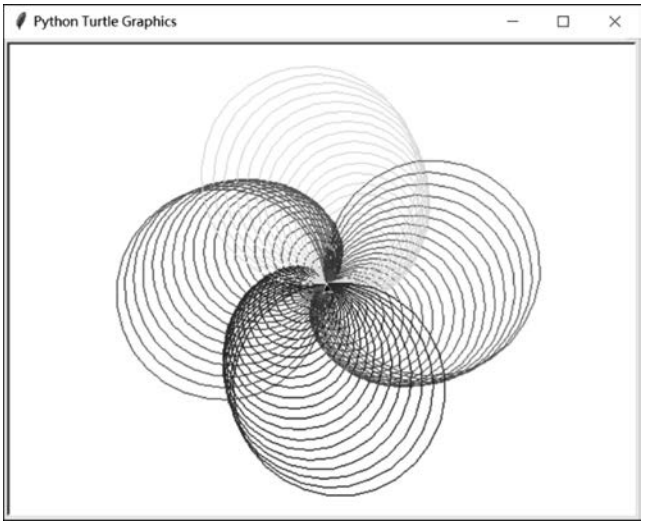


图 3-24 使用海龟绘图绘制四种颜色的圆形螺旋

3.6 习题

扫描下方二维码，下载习题电子版。

扫一扫

习题

3.7 上机实践

1. 完成本章中的例 3.1～例 3.30，熟悉 Python 语言的三种基本控制结构，即顺序结构、选择结构、循环结构。

2. 编写程序，计算 1＋2＋3＋…＋100 之和。

3. 编写程序，计算 10＋9＋8＋…＋1 之和。

4. 编写程序，计算 1＋3＋5＋7＋…＋99 之和。

5. 编写程序，计算 2＋4＋6＋8＋…＋100 之和。

6. 编写程序，使用不同的实现方法，输出 2000～2500 中的所有闰年，程序运行效果如图 3-25 所示。

7. 编写程序，计算 $S_n=1-3+5-7+9-11+\cdots$。

提示：可以使用 if i%2==0 的语句形式判断 i 是否为偶数。

8. 编写程序，计算 $S_n=1+1/2+1/3+\cdots$。

9. 编写程序，打印九九乘法表。要求输出九九乘法表的各种显示效果（上三角、下三角、矩形块等方式）。

10. 编写程序，输入三角形三条边，先判断是否可以构成三角形，如果可以，则进一步求三角形的周长和面积，否则报错“无法构成三角形!”。程序运行效果如图 3-26 所示（结果均保留一位小数）。

```
2000 2004 2008 2012 2016 2020 2024 2028 2032 2036 2040 2044 2048 2052 2056 2060
2064 2068 2072 2076 2080 2084 2088 2092 2096 2104 2108 2112 2116 2120 2124 2128
2132 2136 2140 2144 2148 2152 2156 2160 2164 2168 2172 2176 2180 2184 2188 2192
2196 2204 2208 2212 2216 2220 2224 2228 2232 2236 2240 2244 2248 2252 2256 2260
2264 2268 2272 2276 2280 2284 2288 2292 2296 2304 2308 2312 2316 2320 2324 2328
2332 2336 2340 2344 2348 2352 2356 2360 2364 2368 2372 2376 2380 2384 2388 2392
2396 2400 2404 2408 2412 2416 2420 2424 2428 2432 2436 2440 2444 2448 2452 2456
2460 2464 2468 2472 2476 2480 2484 2488 2492 2496
```

图 3-25 2000～2500 的所有闰年运行效果

```
请输入三角形的边A:    3
请输入三角形的边B:    4
请输入三角形的边C:    5
三角形三边分别为:  a=3.0, b=4.0, c=5.0
三角形的周长 = 12.0, 面积 = 6.0
```

图 3-26 三角形周长和面积运行效果

提示：

（1）3 个数可以构成三角形必须满足如下条件：每条边长均大于 0，并且任意两边之和大于第三边。

（2）已知三角形的三条边，则可以利用海伦公式计算三角形的面积＝$\sqrt{h*(h-a)*(h-b)*(h-c)}$，其中，h 为三角形周长的一半。

11. 编写程序，输入 x，根据如下公式，计算分段函数 y 的值。分别利用“单分支语句”、“双分支结构”以及“条件运算语句”等方法实现。程序运行效果如图 3-27 所示。

$$y=\begin{cases}\dfrac{x^2-3x}{x+1}+2\pi+\sin x & x\geqslant 0\\ \ln(-5x)+6\sqrt{|x|+e^4}-(x+1)^3 & x<0\end{cases}$$

```
请输入x:    -1
方法一: x = -1.0, y = 46.34793812414429
方法二: x = -1.0, y = 46.34793812414429
方法三: x = -1.0, y = 46.34793812414429
```

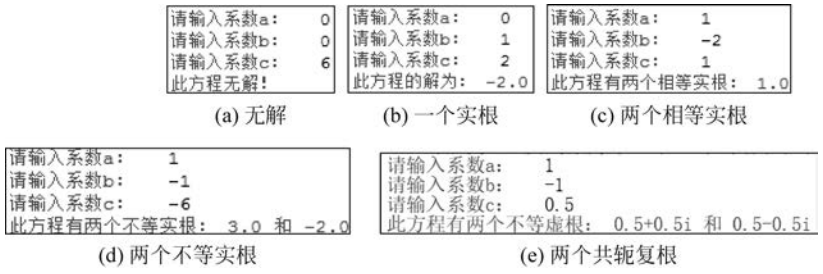
图 3-27 分段函数运行效果

12. 编写程序，输入一元二次方程的三个系数 a、b 和 c，求 $ax^2+bx+c=0$ 方程的解。程序运行效果如图 3-28 所示。

提示：

（1）方程 $ax^2+bx+c=0$ 的解有以下几种情况。

① a＝0 and b＝0，无解。



(a) 无解　(b) 一个实根　(c) 两个相等实根

(d) 两个不等实根　(e) 两个共轭复根

图 3-28　求解一元二次方程

② a=0　and　b!=0，有一个实根：$x=-\frac{c}{b}$。

③ $b^2-4ac=0$，有两个相等实根：$x_1=x_2=-\frac{b}{2a}$。

④ $b^2-4ac>0$，有两个不等实根：$x_1=-\frac{b}{2a}+\frac{\sqrt{b^2-4ac}}{2a}$，$x_2=-\frac{b}{2a}-\frac{\sqrt{b^2-4ac}}{2a}$。

⑤ $b^2-4ac<0$，有两个共轭复根：$x_1=-\frac{b}{2a}+\frac{\sqrt{4ac-b^2}}{2a}i$，$x_2=-\frac{b}{2a}-\frac{\sqrt{4ac-b^2}}{2a}i$。

(2) 可以利用“print(str.format("此方程有两个不等虚根：{0}+{1}i 和{0}-{1}i", realPart, imagPart))”的语句形式输出方程的两个共轭复根。

13. 编写程序，输入整数 n(n>=0)，分别利用 for 循环和 while 循环求 n!。程序运行效果如图 3-29 所示。

提示：

(1) n!=n×(n-1)×(n-2)×…×2×1。例如，5!=5×4×3×2×1=120。特别地，0!=1。

(2) 一般地，累乘的初值为 1，而累加的初值为 0。

(3) 如果输入的是负整数，则继续提示输入非负整数，直至 n>=0。

14. 编写程序，产生两个 0～100(包含 0 和 100)的随机整数 a 和 b，求这两个整数的最大公约数和最小公倍数。程序运行效果如图 3-30 所示。

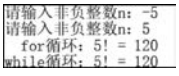


图 3-29　阶乘运行效果

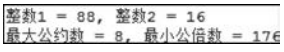


图 3-30　最大公约数和最小公倍数运行效果

提示：

(1) 用户可以利用“random.randint(0,100)”的语句形式生成 0～100(包含 0 和 100)的随机整数。

(2) 用户可以直接使用内置函数 gcd()求若干整数的最大公约数。

(3) 还可以利用“辗转相除法”求最大公约数，具体算法如下。

① 对于已知的两个正整数 m、n，使得 m>n。

② m 除以 n 得余数 r。

③ 若 r≠0，则令 m←n，n←r，继续相除得到新的余数 r。若仍然 r≠0，则重复此过程，

直到 r=0 为止。最后的 m 就是最大公约数。

(4) 当然,还可以利用“更相减损法”求最大公约数,具体算法如下。

“更相减损法”出自中国古代的《九章算术》,也是一种求最大公约数的算法,具体算法如下:

① 先判断两个正整数的大小,如果两数相等,则这个数本身就是所求的最大公约数。

② 如果两个正整数不相等,则用大数减去小数,然后用这个较小数与它们相减的结果相比较,如果相等,则这个差就是两个正整数的最大公约数,而如果不相等,则继续执行②操作。

程序实现可参见本章思考题第 7 题。

(5) 求得了最大公约数后,最小公倍数就是已知的两个正整数之积除以(整除)最大公约数后所得到的商。

15. 假设某理财产品一年期利率为 8.75%,编写程序计算多少年后一万元的这款一年期理财产品连本带息才能翻番? 程序运行效果如图 3-31 所示。

16. 编写程序,输入一个三位自然数,编写程序分别输出其百位数、十位数和个位数上的数字。程序运行效果如图 3-32 所示。要求利用“整除运算符//和求余数(求模)运算符%”“divmod()函数”和“map()函数”至少 3 种不同的编程方法来实现程序的功能。

```
9 年后一万元的一年期理财产品连本带息才能翻番
```

图 3-31 理财产品连本带息翻番的运行效果

```
请输入一个三位自然数: 368
方法一: 3 6 8
方法二: 3 6 8
方法三: 3 6 8
```

图 3-32 提取整数各位上数字的运行效果

17. 棋盘放麦粒。在印度有一个古老的传说,舍罕王打算奖赏国际象棋的发明人——宰相西萨·班·达依尔。国王问宰相想要什么,他对国王说:“陛下,请您在这张棋盘的第 1 个小格里,赏给我 1 粒麦子,在第 2 个小格里给 2 粒,第 3 小格里给 4 粒,以后每一小格都比前一小格加一倍的麦子数量。请您把这样摆满棋盘上所有 64 格的麦粒都赏给您的仆人吧!”国王觉得这要求太容易满足了,就命令给宰相这些麦粒。当人们把一袋一袋的麦子搬来开始计数时,国王才发现,就是把全印度甚至全世界的麦粒全拿来,也满足不了宰相的请求。编写程序显示宰相所要求得到的麦粒总数。程序运行效果如图 3-33 所示。

```
国际象棋64个格子中总的麦粒数量为: 18446744073709551615
```

图 3-33 棋盘放麦粒的运行效果

18. 自幂数探索。输入一个 1～10 的整数,编写程序输出相应的自幂数名称、数量和具体的数值,程序运行效果如图 3-34(a)所示。读者可进一步编程,输出 1～10 所有自幂数的名称、数量和具体的数值清单,程序运行效果如图 3-34(b)所示。

```
请输入自幂数位数(>=1并且<=10):
7
北斗七星数4个:
1741725
4210818
9800817
9926315
```

(a) 输出指定的自幂数

```
独身数9个:1 2 3 4 5 6 7 8 9
两位数没有自幂数!
水仙花数4个:153 370 371 407
玫瑰花数3个:1634 8208 9474
五角星数3个:54748 92727 93084
六合数1个:548834
北斗七星数4个:1741725 4210818 9800817 9926315
八仙数3个:24678050 24678051 88593477
九九重阳数4个:146511208 472335975 534494836 912985153
十全十美数1个:4679307774
```

(b) 1~10所有自幂数的清单

图 3-34 自幂数的运行效果

注意：算法使用逐个数字逐位枚举的方法，因此运行速度会比较慢。如果计算机的性能不够强劲的话，建议不要尝试超过 6 的自幂数，否则，用户可以使用 Ctrl＋C 组合键中止当前程序的运行。

提示：如果在一个固定的进制中，一个 n 位自然数等于自身各个数位上数字的 n 次幂之和，则称此数为自幂数。

例如，在十进制中，153 是一个三位数，各个数位上数字值的 3 次幂之和为 $1^3+5^3+3^3=153$，所以 153 是十进制中的自幂数。三位自幂数称为水仙花数。

在 n 进制中，所有小于 n 的正整数都为自幂数，比如二进制中 1 是自幂数，三进制中 1 和 2 都是自幂数，四进制中 1、2 和 3 都是自幂数，以此类推。

各种自幂数的名称及具体内容如下：

- 一位自幂数：独身数。共有 9 个：1、2、3、4、5、6、7、8、9。
- 二位自幂数：没有自幂数。
- 三位自幂数：水仙花数。共有 4 个：153、370、371、407。
- 四位自幂数：四叶玫瑰数。共有 3 个：1634、8208、9474。
- 五位自幂数：五角星数。共有 3 个：54748、92727、93084。
- 六位自幂数：六合数。只有 1 个：548834。
- 七位自幂数：北斗七星数。共有 4 个：1741725、4210818、9800817、9926315。
- 八位自幂数：八仙数。共有 3 个：24678050、24678051、88593477。
- 九位自幂数：九九重阳数。共有 4 个：146511208、472335975、534494836、912985153。
- 十位自幂数：十全十美数。只有 1 个：4679307774。

在十进制中最大的自幂数有 39 位，共有 88 个自幂数。

19. 编写程序，利用循环结构，使用 turtle 库的 turtle.fd()函数和 turtle.seth()函数绘制一个等边三角形，边长为 200 像素，程序运行效果如图 3-35 所示。

20. 编写程序，利用循环结构，使用 turtle 库的 turtle.right()函数和 turtle.fd()函数绘制一个菱形四边形，边长为 200 像素，程序运行效果如图 3-36 所示。

21. 编写程序，利用循环结构，使用 turtle 库的 turtle.fd()函数和 turtle.left()函数绘制一个正六边形，边长为 100 像素，程序运行效果如图 3-37 所示。

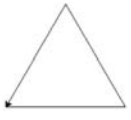


图 3-35　绘制等边三角形

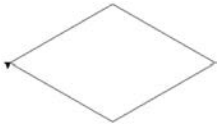


图 3-36　绘制菱形四边形

图 3-37　绘制正六边形

22. 编写程序，利用循环结构，使用 turtle 库的 turtle.circle()函数、turtle.seth()函数和 turtle.left()函数绘制一个四瓣花图形，从左上角花瓣开始，逆时针作画，程序运行效果如图 3-38 所示。

23. 编写程序，利用循环结构，使用 turtle 库的 turtle.right()函数和 turtle.circle()函数绘制一个四叶草，程序运行效果如图 3-39 所示。

24. 编写程序，利用循环结构，使用 turtle 库的 turtle.right()函数和 turtle.circle()函数绘制一个星星图形，程序运行效果如图 3-40 所示。

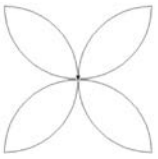


图 3-38　绘制四瓣花图形

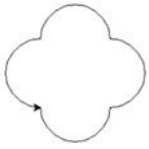


图 3-39　绘制四叶草图形

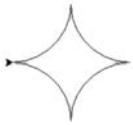


图 3-40　绘制星星图形

25. 编写程序，使用海龟绘图绘制一个由蓝色、黑色、红色、黄色和绿色五种颜色组成的奥运五环图形，程序运行效果如图 3-41 所示。

图 3-41　绘制奥运五环图形

3.8　案例研究：使用嵌套循环实现图像处理算法

在科学计算和各种算法中常常需要使用嵌套循环来处理数据。

例如，图像在计算机中是由像素点组成的二维数组，每个像素点的位置表示为两个整数的元组，像素的值根据图像模式由对应的元组组成（例如 RGB 模式表示为三个整数值组成的元组，分别表示构成颜色的红、蓝、绿的值，范围为 0～255）。

图像处理（例如复制、旋转、裁剪和平滑图像等）的算法根本上就是使用嵌套循环模式对这些像素进行处理。

本章案例研究使用 Python 第三方图像处理库 Pillow 中 PIL.Image 模块的 Image 类的方法 getpixel()和 putpixel()来读取和修改特定位置（loc）处的像素的颜色值（pix），然后使用嵌套循环实现图像处理的基本算法，目的是使学生深入了解 Python 数据结构和基本算法流程。

本章案例研究的解题思路和源代码等以电子版形式提供，具体请扫描如下二维码。

扫一扫

案例研究