

计算机程序是一组计算机能识别和执行的指令,是满足人们某种需求的信息化工具。Python 程序是用 Python 语言编写的一组相关命令的集合。程序以文件的形式存储在磁盘中,Python 程序文件的扩展名为“.py”。

下面以 IDLE 为例,介绍程序文件的一些常用操作方法。

- 创建: 在 IDLE 中,可以通过菜单命令 File→New File(快捷键 Ctrl+N)创建程序文件。
- 保存: 在程序文件窗口中,可以通过菜单命令 File→Save(快捷键 Ctrl+S)或者 File→Save As...(快捷键 Ctrl+Shift+S)对程序文件进行保存。
- 运行: 程序编写完成后,需要运行才能得到程序执行结果。可以在程序文件窗口通过菜单命令 Run→Run Module(快捷键 F5)运行程序。
- 中断: 如果程序执行过程中需要中断程序的执行,可以在 IDLE 交互式窗口中选择菜单命令 Shell→Interrupt Execution(快捷键 Ctrl+C)。

3.1 程序基础

3.1.1 Python 程序的构成

Python 程序文件一般包括注释、模块导入、函数定义和程序主体等几个部分,如图 3-1 所示。



图 3-1 Python 程序文件的基本构成

(1) 注释。注释是在代码中加入的一行或多行信息,用来对模块、函数、方法等进行说明,用以提升代码的可读性。注释是用于辅助程序阅读或编写的,编译器或解释器会略去注释部分的内容。Python 中的注释有单行注释和多行注释两种。Python 中的单行注释以#作为开始标记,可以单独占一行,也可以写在程序代码行的后面,#后的部分即为注释部分,一般用于对该行或该行中部分代码的解释。多行注释写在一对三个单引号('')或者一对三个双引号("''")之间。

(2) 模块导入。若程序中需要用到标准库或者第三方库,则需要先将库导入。具体方法参见第 1.6 节。

(3) 函数定义。函数定义部分一般能够完成一个相对独立的子功能,由程序主体或其他函数调用执行。

(4) 程序主体。程序主体是完成程序功能的主要部分,程序文件需要按照程序主体部分的语句来依次执行。

3.1.2 Python 中的缩进

Python 程序对大小写敏感,对缩进敏感。缩进指每一行代码开始前的空白区域,用来表示代码之间的包含和层次关系。Python 是通过缩进来识别代码块的,因此对缩进非常敏感,对代码格式要求也非常严格。Python 可以使用 Tab 键或 4 个空格缩进一级。

3.1.3 程序基本结构分类

程序在计算机上执行时,程序中的语句完成具体的操作并控制执行流程。程序中语句的执行顺序称为程序结构。

程序包含 3 种基本结构:顺序结构、分支结构和循环结构。顺序结构是指程序中的语句按照出现位置顺序执行;分支结构是指程序中的某些语句按照某个条件来决定执行与否;循环结构是指程序中的语句在某个条件下重复执行多次。

3.2 顺序结构

顺序结构是比较简单的一种结构,也是非常常用的一种结构,其语句是按照位置顺序执行的。如图 3-2 所示,顺序结构中的语句块 1、语句块 2 按位置顺序依次执行。

【例 3-1】 编写程序,通过输入正方体的边长 a ,求正方体的体积。

问题分析:程序的输入、处理和输出三部分分别可以表示如下。

输入:正方体的边长 a 。

处理:正方体的体积 $v=a^3$ 。

输出:正方体的体积 v 。

依据输入、输出和处理过程编写程序代码,参考代码如下:

```
1 # E3-1.py
2 # 计算正方体的体积
3 a = eval(input("输入正方体的边长 a:"))
```

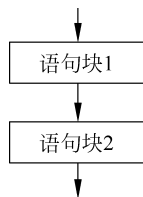


图 3-2 顺序结构流程图

```
4 v = a ** 3
5 print("正方体的体积为:", round(v, 2))
```

程序运行结果如下:

```
输入正方体的边长 a:3.5
正方体的体积: 42.88
```

【例 3-2】 用顺序结构编程求解一元二次方程 $y=3x^2+5x+7$, 要求通过输入 x 的值求得 y 的值。

问题分析: 该程序同样可以分为输入、处理和输出三部分表示。

输入: x 值。

处理: 计算 $y=3x^2+5x+7$ 的值。

输出: y 值。

参考代码如下:

```
1 #E3-2.py
2 #求解一元二次方程
3 x = eval(input("输入 x 的值:"))
4 y = 3 * x ** 2 + 5 * x + 7
5 print("y 的值是:", y)
```

程序运行结果如下:

```
输入 x 的值:3
y 的值是:49
```

【例 3-3】 编写程序, 输入 4 个数, 并求它们的平均值。

问题分析: 该例题仍然采用输入、处理、输出三个步骤, 同时可结合第 2 章对赋值符号的介绍, 注意多个数据输入时的处理方法。

参考代码如下:

```
1 #E3-3.py
2 #求平均值
3 x1, x2, x3, x4 = eval(input("输入 4 个数(逗号分隔):"))
4 avg = (x1 + x2 + x3 + x4) / 4
5 print(avg)
```

程序运行结果如下:

```
输入 4 个数(逗号分隔):15,23,6,7
12.75
```

【例 3-4】 用 turtle 绘制图 3-3 所示的扇形, 其中扇形的半径为 200, 圆心角为 120° , 填充颜色为红色。

问题分析: 将画布原点作为扇形圆心, 并以该圆心作为绘图的起始点, 将默认方向作为绘图的起始方向。按照边、弧线、边的



图 3-3 用 turtle 绘制扇形

绘制顺序即可绘制图 3-3 所示的图形,绘制的同时要注意图形颜色的填充。

参考代码如下:

```
1 #E3-4.py
2 #用 turtle 绘制扇形
3 import turtle
4 turtle.hideturtle()
5 turtle.fillcolor("red")
6 turtle.begin_fill()
7 turtle.fd(200)
8 turtle.left(90)
9 turtle.circle(200,120)
10 turtle.left(90)
11 turtle.fd(200)
12 turtle.end_fill()
```

3.3 分支结构

分支结构也称为选择结构,该结构可以通过判断某些特定条件是否满足,来决定下一步的执行流程。分支结构是非常重要的—种结构。常见的分支结构有单路分支结构、双路分支结构和多路分支结构。

3.3.1 单路分支结构

单路分支结构是分支结构中最简单的一种形式,其语法格式如下所示。

```
if <条件表达式>:
    <语句块>
```

其中:

- 单路分支结构以关键字 if 开头,后接<条件表达式>。
- <条件表达式>可以是关系表达式、逻辑表达式、算术表达式等。
- 冒号“:”表示一个语句块的开始,不能缺少。
- <语句块>可以是单个语句,也可以是多个语句。相对于“if”的位置,<语句块>应缩进 4 个字符。

功能:当<条件表达式>的值为 True 时,执行语句块;若为 False 则不做任何操作,其流程图如图 3-4 所示。

若<条件表达式>的计算结果不是布尔值时,认定数值 0、空字符串、空元组、空列表、空字典均为 False,其余为 True。该判断方法适用于后续所有分支结构和循环结构中对<条件表达式>的判断。

【例 3-5】 输入一个整数,如果是偶数则输出“这是个偶数”,否则无输出。

问题分析:这是一个典型的单路分支的情况,只需考虑条件满足时所要执行的语句。

参考代码如下:

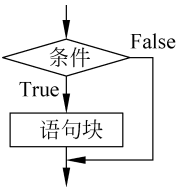


图 3-4 单路分支结构流程图

```

1  #E3-5.py
2  #单分支输出"这是个偶数"
3  s = eval(input("请输入一个整数:"))
4  if s%2 == 0:
5      print("这是个偶数")

```

运行程序,输入一个偶数,程序运行结果如下:

```

请输入一个整数:6
这是个偶数

```

再次运行程序,输入一个奇数,程序运行结果如下:

```

请输入一个整数:3

```

【例 3-6】 输入腋下体温值 t (单位: $^{\circ}\text{C}$),根据如下情况分别输出:

$t < 36.1$	输出: 您的体温偏低
$36.1 \leq t \leq 37$	输出: 您的体温正常
$t > 37$	输出: 您的体温偏高

问题分析: 题目列出了 3 个区间段的取值对应的不同输出,因此可以用三条单路分支语句分别与题目所述 3 种情况对应。

参考代码如下:

```

1  #E3-6.py
2  #体温判断
3  t = eval(input("请输入腋下体温值:"))
4  if t < 36.1:
5      print("您的体温偏低")
6  if 36.1 <= t <= 37:
7      print("您的体温正常")
8  if t > 37:
9      print("您的体温偏高")

```

运行程序,输入 35,程序运行结果如下:

```

请输入腋下体温值:35
您的体温偏低

```

运行程序,输入 36.7,程序运行结果如下:

```

请输入腋下体温值:36.7
您的体温正常

```

运行程序,输入 38.5,程序运行结果如下:

```

请输入腋下体温值:38.5
您的体温偏高

```

【例 3-7】 输入两个整数,并分别存放在 x 、 y 两个变量中,请将这两个数由小到大输出。

问题分析: 假设输出的结果也保存在 x 、 y 这两个变量中,并且输出时满足 $x \leq y$ 。为了

达到这一目标,可以对输入的 x 、 y 值进行判断:如果 $x > y$,则交换 x 和 y 中的值,否则不交换。最后输出 x 、 y 。

参考代码如下:

```
1 #E3-7.py
2 #两个数排序
3 x,y = eval(input("请输入 x,y:"))
4 if x > y:
5     x,y = y,x
6 print(x,y)
```

运行程序,依次输入 5、3,程序运行结果如下:

```
请输入 x,y:5,3
3 5
```

【例 3-8】 输入三个整数,分别存放在 x 、 y 和 z 三个变量中,请将这三个数由小到大输出。

问题分析:与例 3-7 相似,假设输出的结果也保存在 x 、 y 和 z 这三个变量中,并且输出时 $x \leq y \leq z$ 。为了达到这一目标,可按如下步骤对输入的 x 、 y 和 z 进行比较和交换:如果 $x > y$,则交换 x 和 y 中的值,否则不交换;如果 $x > z$,则交换 x 和 z 中的值,否则不交换;如果 $y > z$,则交换 y 和 z 中的值,否则不交换。最后输出 x 、 y 、 z 。

参考代码如下:

```
1 #E3-8.py
2 #三个整数排序
3 x,y,z = eval(input("输入三个数(用逗号分隔):"))
4 if x > y:
5     x,y = y,x
6 if x > z:
7     x,z = z,x
8 if y > z:
9     y,z = z,y
10 print(x,y,z)
```

运行程序,依次输入 3、16、-5,程序运行结果如下:

```
输入三个数(用逗号分隔):3,16,-5
-5 3 16
```

思考:若第 6 行代码改为“if $y > z$:",那么后续代码应如何编写?

3.3.2 双路分支结构

双路分支结构是程序中比较常用的一种形式,其语法格式如下所示。

```
if <条件表达式>:
    <语句块 1>
else:
    <语句块 2>
```

其中:

- 双路分支结构以关键字 if 开头,后接<条件表达式>。
- <条件表达式>可以是关系表达式、逻辑表达式、算术表达式等。
- 冒号“:”表示一个语句块的开始,不能缺少。
- <语句块 1>、<语句块 2>可以是单个语句,也可以是多个语句。<语句块 1>、<语句块 2>相对于 if 和 else 的位置应缩进 4 个字符。

功能:当<条件表达式>的值为 True 时,执行 if 后的<语句块 1>,继而双路分支语句结束;若<条件表达式>的值为 False 则执行 else 后的<语句块 2>,双路分支语句结束,其流程图如图 3-5 所示。

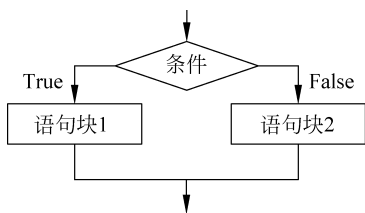


图 3-5 双路分支结构流程图

【例 3-9】 输入一个整数,如果是偶数则输出“这是个偶数”,否则输出“这是个奇数”。

问题分析:本题目是对两种情况的不同处理,因此满足双路分支结构。

参考代码如下:

```

1  #E3-9.py
2  #双路分支判断奇偶
3  s = eval(input("请输入一个整数:"))
4  if s % 2 == 0:
5      print("这是个偶数")
6  else:
7      print("这是个奇数")
  
```

运行程序,输入整数 6,程序运行结果如下:

```

请输入一个整数:6
这是个偶数
  
```

运行程序,输入整数 3,程序运行结果如下:

```

请输入一个整数:3
这是个奇数
  
```

【例 3-10】 输入 x 的值,计算 $y = \begin{cases} \sqrt{x^2 - 25} & x \leq -5 \text{ 或 } x \geq 5 \\ \sqrt{25 - x^2} & -5 < x < 5 \end{cases}$ 中 y 的值。

问题分析:题目中给出的是分成两段的分段函数,且自变量取值范围的并集为全集,因此可以采用双路分支结构。

参考代码如下:

```

1  #E3-10.py
2  #按条件求解表达式的值
3  x = eval(input("输入一个整数:"))
4  if x <= -5 or x >= 5:
5      y = (x**2 - 25)**0.5
6  else:
  
```

```

7     y = (25 - x ** 2) ** 0.5
8     print(y)

```

运行程序,输入整数-15,程序运行结果如下:

```

输入一个整数:-15
14.142135623730951

```

运行程序,输入整数 2,程序运行结果如下:

```

输入一个整数:2
4.58257569495584

```

运行程序,输入整数 15,程序运行结果如下:

```

输入一个整数:15
14.142135623730951

```

注意: 此处也可以通过导入 math 库,利用 sqrt()函数求解平方根。

【例 3-11】 输入一个数,利用 turtle 库绘制如图 3-6 所示的圆形,当输入的数是偶数时,填充颜色为蓝色;当输入的数是奇数时,填充颜色为红色。画笔大小为 5,圆的半径为 100。

问题分析: 该程序将例 3-9 的情形与 turtle 绘图功能相结合。在不同分支下设置不同的填充颜色。

参考代码如下:

```

1  #E3-11.py
2  #用 turtle 绘制圆形
3  import turtle
4  x = eval(input("输入一个整数:"))
5  if x % 2 == 0:
6      turtle.fillcolor("blue")
7  else:
8      turtle.fillcolor("red")
9  turtle.pensize(5)
10 turtle.hideturtle()
11 turtle.begin_fill()
12 turtle.circle(100)
13 turtle.end_fill()

```

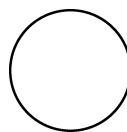


图 3-6 用 turtle 绘制圆形

运行程序文件,当输入偶数时,绘制的是蓝色的圆形;当输入奇数时,绘制的是红色的圆形。

对于一些语句结构简单的情形,双路分支结构还可以用表达式的形式实现,语法格式如下所示。

<表达式 1> if <条件表达式> else <表达式 2>

功能: 当<条件表达式>的值为 True 时,执行<表达式 1>;当<条件表达式>的值为 False 时,执行<表达式 2>。

【例 3-12】 输入一个数,如果这个数是 10,则输出“猜对了”,否则输出“猜错了”。

问题分析：此问题既可以选用常规的双路分支结构表示，又可以选用双路分支结构的表达式形式进行表示。此处选用双路分支的表达式形式表示。

参考代码如下：

```
1 #E3-12.py
2 #猜数
3 guess = eval(input())
4 y = "猜对了" if guess == 10 else "猜错了"
5 print(y)
```

运行程序，输入数字 10，程序运行结果如下：

```
10
猜对了
```

运行程序，输入数字 5，程序运行结果如下：

```
5
猜错了
```

【例 3-13】 用双路分支的表达式形式改写例 3-9。

参考代码如下：

```
1 #E3-13.py
2 #表达式形式判断奇偶
3 s = eval(input("请输入一个整数:"))
4 y = "偶数" if s % 2 == 0 else "奇数"
5 print(y)
```

运行程序，输入整数 6，程序运行结果如下：

```
请输入一个整数:6
偶数
```

运行程序，输入整数 3，程序运行结果如下：

```
请输入一个整数:3
奇数
```

3.3.3 多路分支结构

多路分支结构是双路分支结构的扩展，其语法格式如下所示。

```
if <条件表达式 1>:
    <语句块 1>
elif <条件表达式 2>:
    <语句块 2>
.....
elif <条件表达式 n>:
    <语句块 n>
[else:
    <语句块 n+1>]
```

其中：

- 多路分支结构以关键字 if 开头,后接<条件表达式>。
- <条件表达式 1>、<条件表达式 2>、……、<条件表达式 n>可以是关系表达式、逻辑表达式、算术表达式等。
- 冒号“:”表示一个语句块的开始,不能缺少。
- <语句块 1>、<语句块 2>、……、<语句块 n>可以是单个语句,也可以是多个语句。
<语句块 1>、<语句块 2>、……、<语句块 n>相对于 if、elif 和 else 的位置缩进 4 个字符。
- 方括号部分可以省略。

功能：当<条件表达式 1>的值为 True 时,执行 if 后的<语句块 1>,继而多路分支语句结束；若<条件表达式 1>的值为 False,则继续判断 elif 后<条件表达式 2>的值。若<条件表达式 2>的值为 True,执行<语句块 2>,多路分支语句结束；若为 False,则继续判断 elif 后<条件表达式 3>的值,以此类推。如果所有条件表达式均不成立,则执行 else 部分的<语句块 n+1>,多路分支语句结束,其流程图如图 3-7 所示。

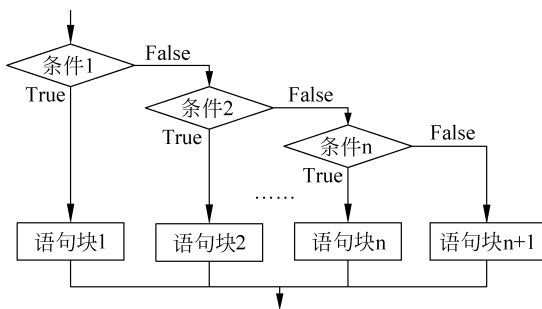


图 3-7 多路分支结构流程图

注意：若<条件表达式 k>成立且执行了其对应的语句块,此时多路分支结构结束,<条件表达式 k>后续的<条件表达式>即便成立也不会执行,并且可以断定<条件表达式 k>之前的<条件表达式>都不成立。

【例 3-14】 编写程序：输入学生成绩，根据成绩所在区间进行分类输出。

90 分以上	输出：优秀
80～89 分	输出：良好
70～79 分	输出：中等
60～69 分	输出：及格
低于 60 分	输出：不及格

问题分析：该题目将学生成绩分成 5 种情况进行分类输出,因此符合多路分支结构的特点。此外,考虑到多路分支结构中前面分支的<条件表达式>不成立才会执行后面<条件表达式>成立的分支语句,因此可适当对每个区间的条件表达式进行简化。

参考代码如下：

```
1 # E3-14.py
2 # 根据成绩区间分类输出
3 score = eval(input("请输入学生成绩:"))
```

```
4 if score >= 90:
5     print("优秀")
6 elif score >= 80:
7     print("良好")
8 elif score >= 70:
9     print("中等")
10 elif score >= 60:
11     print("及格")
12 else:
13     print("不及格")
```

运行程序,输入学生成绩为 69,程序运行结果如下:

```
请输入学生成绩:69
及格
```

运行程序,输入学生成绩为 75,程序运行结果如下:

```
请输入学生成绩:75
中等
```

运行程序,输入学生成绩为 97,程序运行结果如下:

```
请输入学生成绩:97
优秀
```

【例 3-15】 从键盘输入一个字符,赋值给变量 ch,判断它是英文字母、数字或其他字符。

根据题目要求,分析输入字符的 3 种情况:

(1) 英文字母: "a" <= ch <= "z" 或 "A" <= ch <= "Z"。

(2) 数字字符: "0" <= ch <= "9"。

(3) 其他字符: 排除(1)和(2)两种情况的情况。

参考代码如下:

```
1 # E3-15.py
2 # 判断输入字符的种类
3 ch = input("请输入一个字符:")
4 if "a" <= ch <= "z" or "A" <= ch <= "Z":
5     print("英文字母")
6 elif "0" <= ch <= "9":
7     print("数字")
8 else:
9     print("其他字符")
```

运行程序,输入字母 m,程序运行结果如下:

```
请输入一个字符:m
英文字母
```

运行程序,输入数字 7,程序运行结果如下:

```
请输入一个字符:7
数字
```

运行程序,输入其他字符*,程序运行结果如下:

```
请输入一个字符:*
其他字符
```

【例 3-16】 编写程序,输入员工号和该员工的工作时数,计算并输出员工的应发工资。工资计算方法如下:

- (1) 月工作时数超过 150 小时者,超过部分加发 15%。
- (2) 月工作时数低于 100 小时者,扣发 500 元。
- (3) 其余情况按每小时 60 元发放。

问题分析:该题目共描述了 3 种计算方法,符合多路分支结构特点。输入的员工号与应发工资的计算没有关联,只需要根据员工的工作时数,按题目要求分段计算应发工资即可。

参考代码如下:

```
1 #E3-16.py
2 # 计算不同情况下员工工资
3 num = input("员工号:")
4 h = eval(input("工作时数:"))
5 s = 0
6 if h > 150:
7     s = (h + (h - 150) * 0.15) * 60
8 elif h < 100:
9     s = h * 60 - 500
10 else:
11     s = h * 60
12 print("员工的应发工资是",s))
```

输入员工号 3001、工作时数 160,程序运行结果如下:

```
员工号:3001
工作时数:160
员工的应发工资是 9690.0
```

输入员工号 3002、工作时数 120,程序运行结果如下:

```
员工号:3002
工作时数:120
员工的应发工资是 7200
```

输入员工号 3003、工作时数 85,程序运行结果如下:

```
员工号:3003
工作时数:85
员工的应发工资是 4600
```

3.4 循环结构

在程序设计过程中,经常需要将一些代码按照要求重复多次执行,这时就需要用到循环结构。Python 有两种类型的循环语句,分别是 for 循环和 while 循环。for 循环用于确定次数的循环,while 循环多用于非确定次数的循环。

3.4.1 for 循环结构

1. for 语句的一般格式

for 循环结构是循环控制结构中使用较广泛的一种循环控制语句。for 循环以遍历序列对象的方式构造循环,特别适用于循环次数确定的情况,其语法格式如下所示。

```
for <变量> in <序列对象>:
    <循环体>
```

其中:

- <变量>即循环变量,用于存放从序列对象中读取出来的元素。<变量>能够控制循环次数,也可以参与到循环体中。
- <序列对象>可以是字符串、列表、元组、集合、文件等,也可以使用 range() 产生。
- <循环体>中的语句是需要重复执行的部分,相对于 for 要缩进 4 个字符。

功能: 尝试从<序列对象>中选取元素,如果有元素待选取,则执行<循环体>,并在执行<循环体>后,继续尝试从<序列对象>中选取元素;如果<序列对象>没有元素待选取,则结束循环,其流程图如图 3-8 所示。

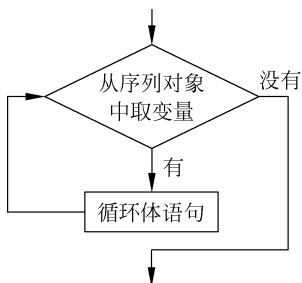


图 3-8 for 循环流程图

2. range()

range() 是 Python 的内建对象,可以创建一个整数序列,常用在 for 循环中,也可以通过 list()、tuple()、set() 等函数将 range() 生成的数据分别转换成列表、元组、集合等类型数据,range() 语法格式如下:

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

其中:

- start 表示计数开始的数值,缺省时为 0。
- stop 表示结束但不包括的数值。
- step 表示步长,可正可负,缺省时为 1。

功能: range() 表示生成一个从 start 值开始,到 stop 值结束(但不包含 stop)的 range 对象。

例如:

range(5) 等价于 range(0,5,1),生成的整数序列为 0,1,2,3,4。

range(2,18,3) 生成的整数序列为 2,5,8,11,14,17。

range(-10,0) 生成的整数序列为 -10,-9,-8,-7,-6,-5,-4,-3,-2,-1。

range(10,1,-2) 生成的整数序列为 10,8,6,4,2。

说明：range()返回的是一个可迭代对象，其类型为“range”，所以使用 print()函数不会输出具体数值。

3. for 语句的应用

for 语句经常与 range()配合使用，用于遍历由 range()生成的整数序列。

【例 3-17】 循环输出 0~5 的整数。

问题分析：可以利用 range(6)生成 0~5 的整数序列，同时结合 for 循环输出该序列。此处 range()的参数设定不唯一。

参考代码如下：

```
1 #E3-17.py
2 #循环输出数值 0~5
3 for i in range(6):
4     print(i,end=' ')
```

程序运行结果如下：

```
0,1,2,3,4,5,
```

【例 3-18】 循环输出 1~10 的奇数。

问题分析：可以利用 range(1,11,2)生成 1~10 的奇数，同时结合 for 循环输出。此处 range()的参数设定不唯一。

参考代码如下：

```
1 #E3-18.py
2 #循环输出 1~10 的奇数
3 for i in range(1,11,2):
4     print(i,end=' ')
```

程序运行结果如下：

```
1 3 5 7 9
```

【例 3-19】 利用循环求解 1~100 的累加和。

问题分析：利用 range()构造 1~100 的整数序列。定义变量 s，并赋初值为 0，用于存放循环累加的结果。每循环一次循环变量 i 会从生成的序列中取一个值，同时将变量 i 的值累加到变量 s 中。当循环结束时，s 中保存的就是 1~100 的累加和。

参考代码如下：

```
1 #E3-19.py
2 #求 1~100 的累加和
3 s = 0
4 for i in range(1,101):
5     s += i
6 print("1 + 2 + 3 + ... + 100 = ",s)
```

程序运行结果如下：

$$1 + 2 + 3 + \dots + 100 = 5050$$

【例 3-20】 使用 turtle 库绘制红色五角星图形,效果如图 3-9 所示,五角星非相邻两个顶点间的距离为 200。

问题分析:如图 3-10 所示,绘制过程以 S 为起始点绘制直线,长度为 200,此时到达五角星的另外一个顶点,然后向右转 144° ,再继续绘制直线,长度为 200,如此反复 5 次,最终绘制出五角星。绘制顺序如图中箭头方向所示。



图 3-9 用 turtle 绘制五角星

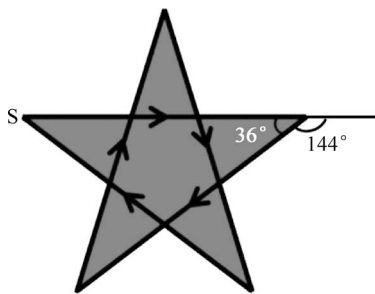


图 3-10 绘制五角星的方法

参考代码如下:

```
1 #E3-20.py
2 #绘制五角星
3 from turtle import *
4 setup(400,400)
5 penup()
6 goto(-100,50)
7 pendown()
8 color("red")
9 begin_fill()
10 for i in range(5):
11     forward(200)
12     right(144)
13 end_fill()
14 hideturtle()
15 done()
```

for 语句还经常用于字符串、列表、元组、字典、集合、文件等的遍历,这些内容将在后续章节中介绍。这里只给出几个简单的实例。

【例 3-21】 利用 for 循环遍历字符串 "PythonStudy", 输出字符 "y" 出现的个数。

问题分析:为了统计字符 "y" 出现的次数,可以定义一个变量 n 用于计数。同时用 for 循环遍历字符串 "PythonStudy", 并利用 if 单路分支结构对循环变量 i 的值进行判断: 如果 i 的值为 "y", 则 n 值增 1。循环结束时变量 n 的值即为字符 "y" 出现的次数。

参考代码如下:

```
1 #E3-21.py
2 #统计某字符出现的个数
3 n = 0
4 for i in "PythonStudy":
```

```

5     if i == 'y':
6         n += 1
7     print("n 的个数为",n)

```

程序运行结果如下：

```
n 的个数为 2
```

【例 3-22】 循环输出列表["11350675","白萌萌","女",660]中的元素。

问题分析：利用 for 循环对列表进行遍历时，循环变量会访问列表中的每一个元素。

参考代码如下：

```

1  # E3 - 22.py
2  # 列表的遍历
3  for st in ["11350675","白萌萌","女",660]:
4      print(st,end = ' - ')

```

程序运行结果如下：

```
11350675 - 白萌萌 - 女 - 660 -
```

与双路分支结构相似，对于一些功能简单的情形，for 循环结构也可以用表达式的形式实现，语法格式如下所示。

[<表达式> for <变量> in <序列对象>]

功能：将每次循环时<表达式>的值作为元素汇集并生成一个新的列表。该语法格式两端的“[]”也可以用“()”“{}”替换，分别用于生成元组、集合。

【例 3-23】 已知公式 $y=3x^2-1$ ，其中 $x \in [1, 7, 12, 21, 56]$ ，求由 y 值组成的列表。

参考代码如下：

```

1  # E3 - 23.py
2  # 表达式形式的 for 语句举例
3  y = [3 * x ** 2 - 1 for x in [1, 7, 12, 21, 56]]
4  print(y)

```

程序运行结果如下：

```
[2, 146, 431, 1322, 9407]
```

3.4.2 while 循环结构

while 循环结构主要用于构造不确定运行次数的循环。while 循环结构也可以完成 for 循环结构的功能，但不如 for 循环结构简单直观。while 循环的语法格式如下所示。

```
while <条件表达式>:
    <循环体>
```

- <条件表达式>可以是关系表达式、逻辑表达式、算术表达式等。
- <循环体>中的语句是需要重复执行的部分。

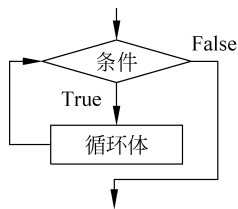


图 3-11 while 循环流程图

功能: while 循环结构首先判断<条件表达式>的值,如果为 True,则重复执行<循环体>,直到<条件表达式>的值为 False 时,结束循环,其流程图如图 3-11 所示。

说明: 在 while 语句的循环体中一定要包含改变测试条件的语句或 break 语句,以避免死循环的出现,保证循环能够结束。

【例 3-24】 用 while 循环改写例 3-17。

问题分析: while 循环也可以解决循环次数确定的情况,但需要在循环之前额外定义控制循环次数的变量,并在循环内部增加改变该变量值的语句。

参考代码如下:

```

1  # E3-24.py
2  # 改写例 3-17
3  i = 0
4  while i < 6:
5      print(i, end = ', ')
6      i = i + 1
  
```

思考: 若此处没有语句 `i=i+1`,则代码执行时会出现什么问题?

【例 3-25】 从键盘输入若干个数,求所有正数之和。当输入 0 或负数时,程序结束。

问题分析: 此处涉及求累加和的问题,但由于循环次数不确定,因此选用 while 循环求解。为保证输入若干个数,循环体中需要有输入语句,使得每次循环都会输入一个新的数据,此处用变量 `x` 表示。同时,执行循环体前需要对条件表达式进行判断,因此在循环体前还需要一条输入语句作为 `x` 的初始值。

参考代码如下:

```

1  # E3-25.py
2  # 求若干个数的和
3  s = 0
4  x = eval(input("请输入一个正整数:"))
5  while x > 0:
6      s = s + x
7      x = eval(input("请输入一个正整数:"))
8  print("s = ", s)
  
```

运行程序,依次输入 15、3、48、26、0,程序运行结果如下:

```

请输入一个正整数:15
请输入一个正整数:3
请输入一个正整数:48
请输入一个正整数:26
请输入一个正整数:0
s = 92
  
```

3.4.3 break、continue 和 pass 语句

break、continue 和 pass 语句在 for 循环和 while 循环中都可以使用,并且在循环体中一

般与选择结构配合使用,以达到满足特定条件时执行的效果。

(1) break 语句。break 语句的作用是终止循环的执行,如果循环中执行了 break 语句,循环就会立即终止。

(2) continue 语句。continue 语句的作用是立即结束本次循环,开始下一轮循环,也就是说,跳过循环体中在 continue 语句之后的所有语句,并根据条件判断是否继续下一轮循环。对于 for 循环,执行 continue 语句后将<序列对象>中的下一个元素赋值给<变量>;对于 while 循环,执行 continue 语句后将转到<条件表达式>判断部分。

(3) pass 语句。pass 语句是空语句,不做任何事情,一般用作占位语句,保证程序结构的完整。例如,如果只是想利用循环延迟后续程序的执行时间,可以让程序循环一定次数却什么都不做,然而循环体可以包含一个语句或多个语句,但是却不可以没有任何语句,此时就可以使用 pass 语句。与 break 和 continue 语句不同的是,pass 语句不仅可以用在循环结构中,也可以用在顺序结构和选择结构中。

【例 3-26】 将字符串“BIYE”中的字符“I”去掉,并输出其他字符。

参考代码如下:

```
1 #E3-26.py
2 # 去掉字符并输出
3 for k in "BIYE":
4     if k == "I":
5         continue
6     print(k, end = "")
```

程序运行结果如下:

```
BYE
```

扩展:若用 break 代替例 3-26 中的 continue,会有什么样的结果?

针对这一问题,改写程序如下所示:

```
1 #E3-26 扩展.py
2 # 去掉字符并输出
3 for k in "BIYE":
4     if k == "I":
5         break
6     print(k, end = "")
```

程序运行结果如下:

```
B
```

分析:在例 3-26 中,程序执行到 continue 时,会跳转到下一次循环,由于字符“I”后面还有其他字符,所以会继续执行。若将 continue 替换为 break,由于 break 语句会跳出循环,所以程序执行结束,后面的字符不再访问。

3.4.4 循环结构的 else 语句

无论是 for 循环还是 while 循环都支持 else 语句,具体格式如下:

for <变量> in <序列对象>:	while <条件表达式>:
<循环体>	<循环体>
[else:	[else:
<语句块>]	<语句块>]

else 所在的方括号部分可以省略。如果循环的结束是由于 for 循环中对<序列对象>的遍历已完成,或者是由于 while 循环的<条件表达式>不成立而自然结束,则执行 else 结构中的<语句块>;如果循环是因为执行了 break 语句而导致提前结束,则不执行 else 中的<语句块>。

由于两种循环的 else 语句部分用法相似,所以我们以 for 循环中的 else 语句为例进行介绍。

【例 3-27】 将字符串“BIYE”中的字符“I”去掉,并输出其他字符。若 for 循环完成遍历,则输出字符串“毕业,再见!”。

参考代码如下:

```
1 #E3-27.py
2 # 去掉字符并输出
3 for k in "BIYE":
4     if k == "I":
5         continue
6     print(k,end=" - ")
7 else:
8     print("毕业,再见!")
```

程序运行结果如下:

B - Y - E - 毕业,再见!

扩展:若用 break 代替例 3-27 中的 continue,会有什么样的结果?

针对这一问题,改写程序如下所示:

```
1 #E3-27 扩展 1.py
2 # 去掉字符并输出
3 for k in "BIYE":
4     if k == "I":
5         break
6     print(k,end=" - ")
7 else:
8     print("毕业,再见!")
```

程序运行结果如下:

B -

分析:如果循环是由 break 语句结束的,则 else 部分的语句不执行,所以程序的运行结果中没有输出字符串“毕业,再见!”

扩展:若用 pass 代替例 3-27 中的 continue,会有什么样的结果?

针对这一问题,改写程序如下所示:

```

1  # E3-27 扩展 2.py
2  # 去掉字符并输出
3  for k in "BIYE":
4      if k == "I":
5          pass
6      print(k, end= " - ")
7  else:
8      print("毕业,再见!")

```

程序运行结果如下：

```
B - I - Y - E - 毕业,再见!
```

分析：pass 语句是一个空语句，在本程序中去掉 if 分支部分也不影响最后的结果；由于没有 break 语句，else 部分也会正常执行。

3.5 嵌套程序

无论是分支结构还是循环结构，都允许嵌套。嵌套就是分支内还有分支，循环内还有循环或者分支内有循环，循环内有分支等。在设计嵌套程序时，要特别注意内层结构和外层结构之间的嵌套关系，以及各语句放置的位置。要保证外层结构完全包含内层结构，不允许出现交叉。

【例 3-28】 输入一个数，如果该数是偶数，则绘制一个圆形（半径为 100，用红色填充），否则逆时针绘制一个正方形（边长为 200，用黄色填充）。

参考代码如下：

```

1  # E3-28.py
2  # 根据输入绘制不同形状
3  from turtle import *
4
5  x = eval(input("输入一个数:"))
6  begin_fill()
7  if x % 2 == 0:
8      fillcolor('red')
9      circle(200)
10 else:
11     fillcolor("yellow")
12     for i in range(4):
13         fd(200)
14         left(90)
15 end_fill()

```

当输入一个偶数时，程序会绘制一个红色圆形；当输入一个奇数时，程序会绘制一个黄色正方形。这段程序用双路分支结构判断输入数据的奇偶，if 部分的程序用于绘制圆形，else 部分用于绘制正方形。总体来讲，该段程序是一个嵌套程序，具体位置是在双路分支结构的 else 部分嵌套了一个 for 循环结构。

对于 while 循环,如果<条件表达式>的值恒为 True,我们称这种循环结构为恒真循环,也称为无穷循环。为了保证恒真循环不是死循环,通常会在恒真循环内部嵌套带有 if 分支结构的 break 语句。

【例 3-29】 按“编号、姓名、性别、年龄”的顺序依次输入若干条记录,将这些记录保存在一个字符串变量 sinput 中,并输出。

问题分析:该题可以利用在 while 循环中嵌套 if 单路分支结构的形式完成。以字符串的形式输入记录,然后通过“+”运算将所有输入字符串连接在一起,并存储在 sinput 变量中。

参考代码如下:

```
1  #E3-29.py
2  '''
3  利用恒真循环实现数据多次输入
4  将数据存储在字符串变量中
5  '''
6
7  #输入
8  s = ''
9  print('按以下格式输入记录:')
10 print('编号 姓名 性别 年龄')
11 while True:
12     x = input('输入一条记录:')
13     if x == '':
14         break
15     s = s + x + '\n'
16
17 #输出
18 print(s)
```

程序运行结果如下:

```
按以下格式输入记录:
编号 姓名 性别 年龄
输入一条记录:001 王梦 女 20
输入一条记录:002 孙凯 男 19
输入一条记录:003 吴越 女 19
输入一条记录:
001 王梦 女 20
002 孙凯 男 19
003 吴越 女 19
```

【例 3-30】 用恒真循环改写例 3-25。

问题分析:例 3-25 中使用了两次输入语句:一次在循环前用于初始化 x 变量,另一次是在循环体中用于多次输入 x 的值。由于恒真循环的条件始终为 True,因此恒真循环的第一次循环必然会执行,所以可以将例 3-25 中循环前用于初始化变量 x 的输入语句删除,只保留循环体中的输入语句即可。为了保证恒真循环能够结束,需要在循环体中嵌套一个单路分支结构,用于判断当输入的 x 小于 0 时结束循环。

参考代码如下：

```
1 #E3-30.py
2 #用恒真循环改写例3-25
3 s = 0
4 while True:
5     x = eval(input("请输入一个正整数:"))
6     if x <= 0:
7         break
8     s = s + x
9 print("s = ", s)
```

循环内嵌套循环也是常用的一种嵌套程序模式。这种嵌套模式需要特别注意内层循环和外层循环之间变量的关系。

【例 3-31】 利用循环输出九九乘法表。

问题分析：可以先用占位符代替乘法表中的每一个算式，然后用占位符构造出九九乘法表直角三角形的外层，再将占位符转换回每一个具体的算式。构造外层时，可以先考虑每行输出算式的个数，再考虑多行输出的情况，从而构造出外层循环变量和内层循环变量的函数关系，并参照此函数关系设置 range() 的参数。

参考代码如下：

```
1 #E3-31.py
2 #九九乘法表
3 for i in range(1,10,1):
4     for j in range(1,i+1,1):
5         print('{ } * { } = { :<2d}'.format(i,j,i*j),end=' ')
6     print('')
```

程序运行结果如下：

```
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
```

【例 3-32】 利用循环输出由 * 号组成的图形，如图 3-12 所示。

问题分析：可以采用 for 循环嵌套的形式输出图形。外层 for 循环决定了输出图形的行数。有两个内层 for 循环：第一个用于循环输出空格，循环次数决定了输出空格的数量；第二个用于循环输出 * 号，循环次数决定了输出 * 号的数量。内层循环的循环次数可以通过构造每行的行号分别与所在行的空格数和 * 号数的对应函数关系来确定。

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

图 3-12 由 * 号组成的图形

参考代码如下：

```
1 #E3-32.py
2 # 输出图形
3 for i in range(4):
4     # 输出空格
5     for j in range(3-i):
6         print(' ',end='')
7     # 输出 * 号
8     for j in range(2*i+1):
9         print(" * ",end='')
10    print('')
```

程序运行结果如下：

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

3.6 程序的异常处理

对于程序设计者而言,在编写程序过程中可能会出现一些错误,导致程序出错或终止。通常错误可以分为如下3类。

(1) 运行时错误。在运行时产生的错误称为异常,如果在程序中引发了未经处理的异常,程序就会由于异常而终止运行。例如除数为0、名字错误、磁盘空间不足等。

(2) 语法错误。语法错误也称编译错误或解析错误,是指违背语法规则而引发的错误。通常情况下,Python解释器会指出错误的位置和类型。例如函数括号的重复或缺失,分支或循环结构中冒号的缺失等。

(3) 逻辑错误。逻辑错误不会导致解释器报错,但执行结果不正确。其主要原因是代码的逻辑错误导致的。例如:算术表达式中的“+”错误地写成了“-”号,表达式“i+=1”错误地写成了“i+=2”,或者语句先后顺序颠倒等问题造成的逻辑错误。

在发生异常时,Python解释器会返回异常信息。

【例 3-33】 异常发生举例。

```
1 #E3-33.py
2 # 异常发生举例
3 x=3
4 y=int(input("输入一个整数:"))
5 print(x+y)
```

程序运行结果如下：

```
输入一个整数: 3.1
Traceback (most recent call last):
  File "D:/ Eg /E-33.py", line 4, in <module>
    y=int(input("输入一个整数:"))
ValueError: invalid literal for int() with base 10: '3.1'
```

该结果出现了异常,这是由于输入的数据是一个浮点数而非整数,int 函数在对输入数据进行转换时发生了异常。

该异常运行结果中的信息说明如下。

- Traceback: 异常回溯标记。
- D:/Eg/E3-33.py: 异常文件路径。根据文件保存的位置的不同而不同。
- line 4: 异常产生的代码行数。根据异常产生的位置不同,“line”后的数字也会不同。
- ValueError: 异常类型。根据异常发生的情况进行分类,常见异常类型有很多种,例如 ZeroDivisionError、NameError 等。
- invalid literal for int() with base 10: '3.1': 异常内容提示。不同的异常情况会有不同的提示。

Python 语言采用结构化的异常处理机制捕获和处理异常。异常处理的语法格式如下所示。

```
try:
    <语句块 0>
except[ <异常类型 1>]:
    <语句块 1>
[except[ <异常类型 2>]:
    <语句块 2>
.....
except[ <异常类型 n>]:
    <语句块 n>]
[else:
    <语句块 n+1>]
[finally:
    <语句块 n+2>]
```

其中:

- 异常处理以 try 开头。
- <语句块 0>中的语句是可能发生异常的语句,若有异常,则执行到发生异常前。
- <异常类型 1>和<语句块 1>: 将程序运行时产生的异常类型与<异常类型 1>进行比对,若一致则执行对应的<语句块 1>。若<异常类型 1>缺省,则所有情况的异常都可以在对应<语句块 1>中执行。其余异常类型和语句块的处理方式与<异常类型 1>/<语句块 1>部分的处理方式相同。
- else: 不发生异常时执行<语句块 n+1>。
- finally: 发不发生异常都执行<语句块 n+2>。

最简单的捕捉和处理异常的形式是采用一个 try-except 结构来完成。

【例 3-34】 利用简单的 try-except 结构处理例 3-33 中的异常。

```
1 # E3-34.py
2 # 简单异常处理举例
3 x = 3
4 try:
5     y = int(input("输入一个整数:"))
6     print(x + y)
7 except ValueError:
8     print("输入错误,请输入一个整数")
```


输入数值 3.1,程序运行结果如下:

输入一个整数:3.1
输入错误,请输入一个整数

输入数值 5,程序运行结果如下:

输入一个整数:5
8

常见异常类型如表 3-1 所示。

表 3-1 常见异常类型

异常类型	描 述
AttributeError	调用未知对象属性时引发的异常
EOFError	发现一个不期望的文件或输入结束时引发的异常
IndexError	使用序列中不存在的索引时引发的异常
IOError	I/O 操作引发的异常
KeyError	使用字典中不存在的关键字引发的异常
NameError	使用不存在的变量名引发的异常
ValueError	参数错误引发的异常
ZeroDivisionError	除数为 0 引发的异常

【例 3-35】 下面两段程序中,假设有可能出现两种异常,一种是由于输入异常导致,另一种是由于除数为 0 导致。输入异常是在程序运行时输入非数值型数据产生的,除数为 0 异常是在程序运行时输入数值 0 产生的。对比以下两种处理方法,分析程序结果及原因。

```
1 #E3-35 方法一.py
2 #方法一
3 x = 2
4 try:
5     y = eval(input("y = "))
6     z = x/y
7     print(z)
8 except:
9     print('除数为 0')
```

```
1 #E3-35 方法二.py
2 #方法二
3 x = 2
4 try:
5     y = eval(input("y = "))
6     z = x/y
7     print(z)
8 except ZeroDivisionError:
9     print('除数为 0')
```

解析:“方法一”和“方法二”的不同点就在于 except 后的异常类型是否标注。针对两种不同的异常类型,分别进行分析。

(1) 除数为 0 产生的异常。

在输入时,若 y 的值输入为 0,则应产生除数为 0 的异常。

“方法一”运行时,输入 y 的值为 0,运行结果如下:

```
y = 0
除数为 0
```

“方法二”运行时,输入 y 的值为 0,运行结果如下:

```
y = 0
除数为 0
```

(2) 输入错误产生的异常。

在输入时,若 y 的值输入为非数值类型数据,则应产生异常。

“方法一”运行时,输入 y 的值为字母 k ,运行结果如下:

```
y = k
除数为 0
```

“方法二”运行时,同样输入 y 的值为字母 k ,运行结果如下:

```
y = k
Traceback (most recent call last):
  File "D:/Eg/E-35 方法二.py", line 5, in <module>
    y = eval(input("y = "))
  File "<string>", line 1, in <module>
NameError: name 'k' is not defined
```

我们看到,两种方法中输入 y 的值都是非数值型数据 k ,但结果却不相同。“方法一”中并没有解决输入的 y 值是非数值的问题,而是仍然认为输入的除数为 0,这显然是错误的。导致这一错误的原因是 `except` 后的异常类型是否给出。如果没有明确给出异常类型,则认为程序中产生的所有异常都可以在该 `except` 后的语句中被处理。因此虽然输入非数值数据导致的异常并不应该打印“除数为 0”,但解释器也错误地认为可以用这种方法来解决。而“方法二”则明确了该 `except` 语句能够解决的异常类型为 `ZeroDivisionError`,所以这部分语句只处理除数为 0 的异常,而输入非数值数据异常发生时,并没有将该异常捕获,而是将异常抛出。

【例 3-36】 改写例 3-35 使其能够正确处理两种类型的异常。

参考代码如下:

```
1  #E3-36.py
2  #改进例 3-35
3  x = 2
4  try:
5      y = eval(input("y = "))
6      z = x/y
7      print(z)
8  except NameError:
9      print("输入错误,请输入一个数")
10 except ZeroDivisionError:
11     print('除数为 0')
```

```
12     z = x/(y + 0.0001)
13     print(z)
14 else:
15     print("没有异常发生")
16 finally:
17     print("计算完成")
```

运行程序,为 y 的值输入 0,程序运行结果如下:

```
y = 0
除数为 0
20000.0
计算完成
```

运行程序,为 y 的值输入 k,程序运行结果如下:

```
y = k
输入错误,请输入一个数
计算完成
```

运行程序,为 y 的值输入 3,程序运行结果如下:

```
y = 3
0.6666666666666666
没有异常发生
计算完成
```

3.7 random 库

random 库是 Python 的一个标准库,包含多种生成随机数的函数。random 库中的大部分函数都基于 random() 函数,该函数使用梅森旋转算法(Mersenne twister)生成随机数。梅森旋转算法是一个伪随机数发生算法,所以由 random() 函数和基于 random() 函数生成随机数的函数所生成的随机数都是伪随机数。

使用 random 库的功能前,需要先导入 random 库。

random 库包含两类函数,分别是基本随机函数和扩展随机函数,如表 3-2 和表 3-3 所示。

表 3-2 random 库基本随机函数

函 数	功 能 描 述	表达式实例 & 实例结果
seed(a)	用于初始化随机数种子,a 缺省时默认为当前系统时间。只要确定了随机种子,每一次产生的随机序列都是确定的	实例: random.seed(5) 结果: 无显示结果
random()	用于生成一个位于[0.0,1.0]的随机小数	实例: random.random() 结果: 0.6229016948897019 (此结果前提是执行语句 random.seed(5))

表 3-3 random 库扩展随机函数

函 数	功 能 描 述	表达式实例 & 实例结果
uniform(a,b)	用于生成一个位于[a,b]的随机小数	实例: random.uniform(3,7) 结果: 5.967147957042918 (此结果随机, 在[3,7]中选取任一小数)
randint(a,b)	用于生成一个位于[a,b]的整数, 要求 $a \leq b$	实例: random.randint(3,7) 结果: 6 (此结果随机, 在[3,7]中选取任一整数)
		实例: random.randint(7,3) 结果: ValueError
randrange ([start,] stop[,step])	用于生成一个位于[start,stop)且以step为步长的随机整数。start缺省时的默认值为0,step缺省时的默认值为1。要求 $start \leq stop$ 时,step为正; $start > stop$ 时,step为负。有参数step时,start不可以缺省	实例: random.randrange(1,10,2) 结果: 3 (此结果随机, 在[1,10)中选取步长为2的任一整数)
		实例: random.randrange(10,2) 结果: ValueError
		实例: random.randrange(10) 结果: 8 (此结果随机, 在[0,10)中选取任一整数)
choice(seq)	用于从序列 seq 中随机选取一个元素	实例: random.choice([3,7,8,11,23]) 结果: 7 (此结果随机, 在给定参数列表中选取任一元素)
		实例: random.choice('Python') 结果: 't' (此结果随机, 范围是'Python'中的任一字母)
		实例: random.choice(['Python','VC','VB','Java']) 结果: 'VC' (此结果随机, 在给定参数列表中选取任一元素)
shuffle(seq)	用于将序列 seq 的顺序重新随机排列,并返回重新排列后的结果	实例: s=[3,7,8,11,23] random.shuffle(s) print(s) 结果: [3,8,23,11,7] (此结果随机)
sample(seq,k)	用于从序列 seq 中随机选取 k 个元素组成新的序列	实例: random.sample([3,7,8,11,23],3) 结果: [11,8,23] (此结果随机)
getrandbits(k)	用于生成一个 kb 长的随机整数	实例: random.getrandbits(6) 结果: 38 (对应的二进制数为 100110,6b 长。此结果随机)

【例 3-37】 班级举行抽奖活动,参与抽奖活动的学生有 10 名,分别是王东明、张帅阳、郝晴、潘盼盼、魏晓娟、孙清成、于胜、李丹、王文菲、高爽。利用随机函数抽取 3 名幸运之星。

参考代码如下:

```

1  # E3-37.py
2  # 抽取幸运之星
3  import random

```

```

4  namelist = ['王东明', '张帅阳', '郝晴', '潘盼盼', '魏晓娟', '孙清成', '于胜',
5  '李丹', '王文菲', '高爽']
6  print(random.sample(namelist, 3))

```

程序运行的一个随机结果如下：

```
['李丹', '王文菲', '魏晓娟']
```

【例 3-38】 猜数字游戏：程序随机生成 1~5 中的一个整数，参与者输入猜想的数值，若猜中，则输出“猜中啦”；否则输出“不对哦”。

参考代码如下：

```

1  #E3-38.py
2  #猜数字游戏
3  import random
4  num = random.randint(1,5)
5  guess = int(input('请输入 1-5 中的整数:'))
6  print('答案是{},您猜的数字是{}'.format(num,guess))
7  if guess == num:
8      print('猜对啦')
9  else:
10     print('不对哦')

```

按要求输入一个整数，程序会生成一个待匹配的随机数，运行结果如下：

```

请输入 1-5 中的整数:3
答案是 2,您猜的数字是 3
不对哦

```

【例 3-39】 猜数字游戏升级版：程序随机生成 1~5 中的一个整数，参与者输入猜想的数，若猜中，则输出“猜中啦”；否则输出“不对哦”。循环该过程，直到用户不想玩为止。

参考代码如下：

```

1  #E3-39.py
2  #循环猜数字游戏
3  import random
4  while True:
5      num = random.randint(1,5)
6      guess = int(input('请输入 1-5 中的整数:'))
7      print('答案是{},您猜的数字是{}'.format(num,guess))
8      if guess == num:
9          print('猜对啦')
10     else:
11         print('不对哦')
12     yn = input('接着玩按 y -- ->')
13     if yn != 'y' and yn != 'Y':
14         break

```

这段代码将例 3-38 的程序嵌套到一个 while 恒真循环中，并添加了用户意愿交互，提升用户体验。根据提示依次输入 3、y、5、n，程序运行结果如下：

```

请输入 1-5 中的整数:3
答案是 1,您猜的数字是 3
不对哦
接着玩按 y ---> y
请输入 1-5 中的整数:5
答案是 5,您猜的数字是 5
猜对啦
接着玩按 y ---> n

```

上机练习 3

说明：创建程序文件完成下列练习。

【题目 1】 输入圆柱体的高(H)和底面半径(R),编程求解并输出圆柱体的体积(V)。

【题目 2】 编程求解一元二次方程 $ax^2+bx+c=0$ 的两个根,要求输入 a、b、c 的值分别为 3、4、1。

【题目 3】 用 turtle 绘制如下填充了 3 种颜色的圆形,圆的半径为 200,填充颜色分别为红色、黄色、蓝色,如图 3-13 所示。

提示：参照例 3-4 编写代码,并尝试完成如下问题:

- (1) 更换本题中的颜色,并填充。
- (2) 尝试在圆中填充 6 种或更多种的颜色。

【题目 4】 输入一个数,如果该数大于 0,则输出其平方根。

【题目 5】 输入一个字符,如果该字符是字母,则输出“用户输入的是字母”。

【题目 6】 输入一个数,如果该数能被 3 和 7 整除,则输出“该数能同时被 3 和 7 整除”。

【题目 7】 某运输公司的收费按照用户运送货物的路程进行计算,其运费折扣标准如表 3-4 所示。请编写程序:在输入路程后,计算运输公司的运费并输出。

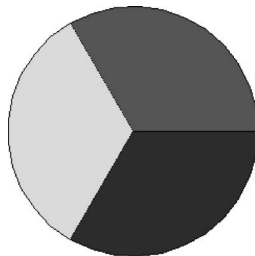


图 3-13 用 turtle 绘制三原色圆

表 3-4 运费折扣标准

路程/km	运费的折扣	路程/km	运费的折扣
$s < 250$	没有折扣	$1000 \leq s < 2000$	8%
$250 \leq s < 500$	2%	$2000 \leq s < 3000$	10%
$500 \leq s < 1000$	5%	$s \geq 3000$	15%

【题目 8】 输入三个数,找出其中的最大数。

提示：首先,输入三个数并分别保存到变量 x、y、z 中。先假定第一个数是最大数,即将 x 赋值给 max 变量,然后将 max 分别和 y、z 比较,两次比较后,max 的值即为 x、y、z 中的最大数。

要求：用嵌套的 if 结构来实现。

【题目 9】 计算 1~100 所有偶数的和。

【题目 10】 计算 $n!$ 。

【题目 11】 利用 for 循环逆时针绘制一个正方形,填充颜色为黄色,如图 3-14 所示。

【题目 12】从键盘输入若干个数,求所有正数的平均值。当输入 0 或负数时,程序结束。

提示:参照例 3-25。

【题目 13】输入一个数 n ,绘制 n 个如图 3-15 所示的圆,半径分别为 1×50 、 2×50 、 $\dots$ 、 $n \times 50$ 。要求:分别用 for 和 while 两种循环结构来完成。

【题目 14】利用 for 循环去掉字符串 "ABCABCABC" 中所有字母 "C",并输出。

【题目 15】用 while 改写题目 14。

【题目 16】利用循环输出如图 3-16 所示的图形。

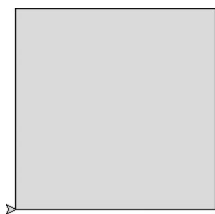


图 3-14 用 turtle 绘制正方形

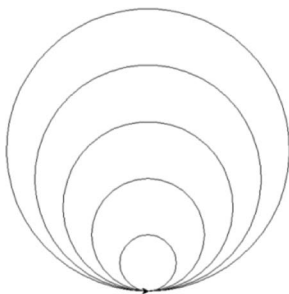


图 3-15 用 turtle 绘制多个圆



图 3-16 由 * 号组成的图形

【题目 17】请用异常处理改造如下程序,使其能够接收并处理用户的任何输入。

```
1 a = input("输入一个字符数据:")
2 n = int(input("输入一个整数:"))
3 for i in range(n):
4     print("序号为:{}".format(a, i))
```

【题目 18】随机产生一个 1~10 内的整数。玩家竞猜,对玩家猜的数字进行判断,根据不同情况分别输出“猜对了”“小了”“大了”。

【题目 19】用随机函数模拟扑克牌洗牌。

习 题 3

【选择题】

- 关于 Python 程序文件的描述,错误的是()。
 - Python 程序文件的扩展名为 .py
 - Python 文件一般包括模块导入、函数定义和程序主体等几个部分
 - Python 文件中必须包含注释部分
 - 双击 .py 文件不能打开 IDLE 的编辑窗口
- 关于缩进,以下说法中错误的是()。
 - Python 程序对缩进有着严格的要求
 - Python 可以使用 Tab 键缩进一级
 - Python 可以使用 4 个空格缩进一级
 - Python 可以使用 Insert 键缩进一级

3. 关于下面程序,说法正确的是()。

```
1 import turtle
2 x = eval(input("输入一个整数:"))
3 if x % 2 == 0:
4     turtle. color("blue")
5 else:
6     turtle. color("red")
7 turtle.pensize(5)
8 turtle.begin_fill()
9 turtle.circle(100)
10 turtle.end_fill()
```

- A. 这段代码的结果是绘制一个圆环
 - B. 绘制图形的线条颜色是黑色的
 - C. 绘制图形的填充颜色是红色或蓝色
 - D. 绘制圆形的半径是 110
4. 关于 Python 的分支结构,以下说法中正确的是()。
- A. 分支结构分为单路分支、双路分支和多路分支
 - B. 不能用双路分支语句改写多路分支语句
 - C. 在多路分支结构中,只要条件满足,所有分支下的语句都会被执行
 - D. 分支结构中不能嵌套其他分支结构
5. 对于语句 `y="Yes" if guess==5 else "No"`,描述错误的是()。
- A. 如果 `guess` 的值为 5,则 `y` 的值是 "Yes"
 - B. 如果 `guess` 的值为 15,则 `y` 的值是 "No"
 - C. 如果 `guess` 的值为 5,则 `y` 的值是 "No"
 - D. 如果 `guess` 的值为 -5,则 `y` 的值是 "No"
6. 下面程序的运行结果是()。

```
1 sum = 0
2 for i in range(2,101):
3     if i % 2 == 0:
4         sum += i
5     else:
6         sum -= i
7 print(sum)
```

- A. 49
 - B. 50
 - C. 51
 - D. 52
7. 下面程序的运行结果是()。

```
1 s = 'LNDX'
2 for i in s:
3     if i == 'N':
4         break
5     print(i)
```

- A. LNDX

- B. L
- N
- D
- X
- C. LN
- D. L

8. 关于下面程序,说法正确的是()。

```

1 num = 6
2
3 while True:
4     guess_num = int(input("guess_num:"))
5     if guess_num == num:
6         print("Right")
7         break
8     else:
9         print("Wrong")

```

- A. 该循环是一个死循环
 - B. 输入 6 时循环结束
 - C. 输入“Right”时循环结束
 - D. 输入“Wrong”时循环结束
9. 以下说法中错误的是()。
- A. break 语句的作用是跳出循环
 - B. continue 语句的作用是结束本次循环
 - C. pass 语句的作用是忽略本次循环
 - D. 循环语句可以和 break、continue 或 pass 语句一起使用
10. 关于 random 库,以下说法错误的是()。
- A. random 库中的大部分函数都基于 random() 函数
 - B. random 库是 Python 的一个第三方库
 - C. shuffle() 用于将序列的顺序重新随机排列
 - D. sample() 用于从序列中随机选取指定的多个元素组成新的序列

【填空题】

1. _____ 指每一行代码开始前的空白区域,用来表示代码之间的包含和层次关系。
2. 分支结构分为: _____ 结构、_____ 结构和 _____ 结构。
3. 表达式 'A' if 3>6 else ('B' if 5>2 else 'C') 的值为 _____。
4. 在循环语句中,_____ 语句的作用是跳出循环。
5. 执行循环语句 for i in range(2,8,2):print(i),循环体执行的次数是_____。
6. _____ 语句是空语句,不做任何事情,一般只用作占位。
7. _____ 就是分支内还有分支,循环内还有循环或者分支内有循环,循环内有分支等。
8. 程序执行时,如果除数为 0,会导致的异常类型为_____。
9. random 库包含两类函数,分别是 _____ 函数和 _____ 函数。
10. random 库中,_____ 函数的作用是初始化随机种子。

【判断题】

1. Python 代码的注释只能使用#号。 ()
2. 程序的 3 种基本结构为：顺序结构、分支结构和循环结构。 ()
3. 多路分支可以用双路分支改写。 ()
4. 如果仅仅是用于控制循环次数,那么使用 `for i in range(20)`和 `for i in range(20,40)`的作用是等价的。 ()
5. 在循环中 `continue` 语句的作用是跳出循环。 ()
6. 对于带有 `else` 子句的 `while` 循环语句,如果是因为循环条件表达式不成立而自然结束循环,则执行 `else` 子句中的代码。 ()
7. 程序中异常处理结构在大多数情况下是没必要的。 ()
8. 在异常处理结构中,不论是否发生异常,finally 子句中的代码总是会执行的。 ()
9. 在 `try...except...else` 结构中,如果 `try` 中语句引发了异常则会执行 `else` 中的代码。 ()
10. `randint(m,n)`用来生成一个 `[m,n]`区间上的随机整数。 ()

【简答题】

1. 简述 Python 程序的基本构成。
2. <条件表达式>的值在什么情况下为 False?
3. 简述 `range()`函数三个参数的作用。
4. 简述 `break`、`continue` 和 `pass` 语句的作用。
5. 简述异常的概念。