第5章 程序控制结构

知识导图

本章知识导图如图 5-0 所示。

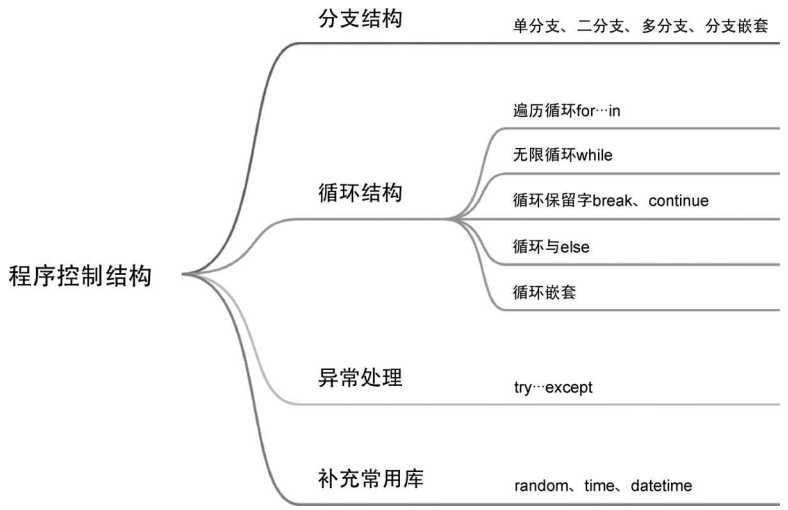


图 5-0　第 5 章知识导图

问题导向

- 鱼与熊掌不可兼得时，如何选择？
- 选择时，用于判断的条件如何设置？
- 多个条件限制时如何设置？
- 用什么来控制重复做某些事情？从什么时候开始？到什么时候结束？
- 如果不知道具体要重复多少次，不确定什么时候结束，如何处理？

重点与难点

- 分支结构的条件设置，逻辑关系。
- 循环结构的基本格式。
- 无限循环的终止。
- 循环嵌套的应用。

5.1 程序流程图

【案例 5-1】 掷骰子游戏。

输入一个整数 n。A、B 两人玩掷骰子游戏，每一盘游戏中每个人轮流掷 n 次，将每次掷出的骰子点数累加，n 盘之后，累计点数较大的获胜，点数相同则为平局。根据此规则实现掷骰子游戏，并算出 n 盘后的胜利者。


观看视频

程序分析：

（1）掷出来的骰子点数是随机的，这里需要用随机函数来辅助。

（2）A、B 轮流掷骰子，每个人都掷 5 次，需要将每一次掷出来的点数进行累加。

这个操作需要重复 5 次，可以用循环来进行控制，开始次数为 0，结束次数为 5。重复执行的代码是：掷骰子，加点数。

（3）对累加后得到的总点数进行比较，谁的点数大，谁获胜，相同为平局。

可用如图 5-1 所示流程图表示。

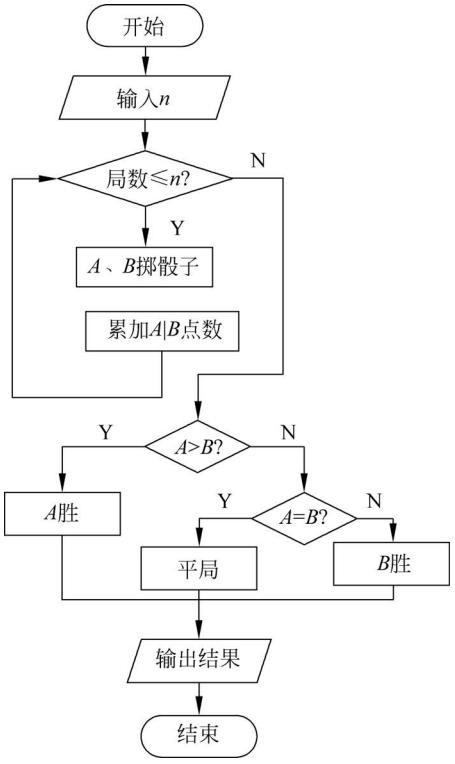


图 5-1 掷骰子流程图

代码实现：

```
import random
n = eval(input('请输入局数：'))
sumA = 0
sumB = 0
```

```
for i in range(n):
  a = random.randint(1,6)
  b = random.randint(1,6)
  sumA += a
  sumB += b
if sumA > sumB:
  print('A 获胜')
elif sumA == sumB:
  print('双方战成平局')
else:
  print('B 获胜')
```

在上面的案例中，为了把程序执行过程展现出来，用各种不同的几何图形组合起来得到了一个流程图。

在程序设计中，算法的描述方法有用自然语言表示算法，用流程图表示算法，用 N-S 流程图表示算法，用伪代码表示算法。

自然语言描述的算法通俗易懂，不用专门的训练，但存在以下问题：①由于自然语言的歧义性，容易导致算法执行的不确定性；②自然语言的语句一般较长，导致描述的算法太长；③当一个算法中循环和分支较多时就很难清晰地表示出来；④自然语言表示的算法不便翻译成计算机程序设计语言。

伪代码的优势：伪代码回避了程序设计语言的严格、烦琐的书写格式伪代码的书写方便，同时具备格式紧凑、易于理解、便于向计算机程序设计语言过渡的优点。伪代码的不足：由于伪代码的种类繁多，语句不容易规范，有时会产生误读。所以这里主要介绍常用的流程图和 N-S 图，下面简单学习一下算法的这两种表示方法。

1. 程序流程图

程序流程图是一种传统的算法表示方法。程序流程图利用图形化的符号框来代表各种不同性质的操作，并用流程线来连接这些操作。用流程图表示算法，直观形象，易于理解。如图 5-2 所示的流程图符号，给出了流程图的构图元素，以及对应的不同含义。

流程图是程序分析和过程描述的最基本方式，它的基本元素一共有 7 种，如图 5-3 所示。

起止框表示一个程序的开始和结束；判断框表示判断一个条件是否成立，并根据判断结果选择不同的执行路径；操作框表示对数据的处理过程；输入输出框表示数据输入或输出结果；注释框表示对程序的解释说明；流程线以带箭头直线或曲线形式指示程序的执行路径；连接点将多个流程图连接到一起，常用于将一个较大流程图分隔为若干部分。

在当前的程序设计中，程序有三种基本结构，分别为顺序结构、分支结构和循环结构。三种基本结构都是有一个入口、一个出口。

顺序结构是程序的基础，计算机程序可以看作一条一条顺序执行的代码，但是单一的顺序结构不可能解决所有问题，因此需要引入控制结构来更改程序的执行顺序以满足多样的功能需求。

分支结构是程序根据条件判断结果而选择不同的路径向前执行的方式，根据分支路径上的完备性，分支结构包括单分支结构和二分支结构。

循环结构是程序根据条件判断的结果决定是否向后反复执行的一种方式。根据循环体

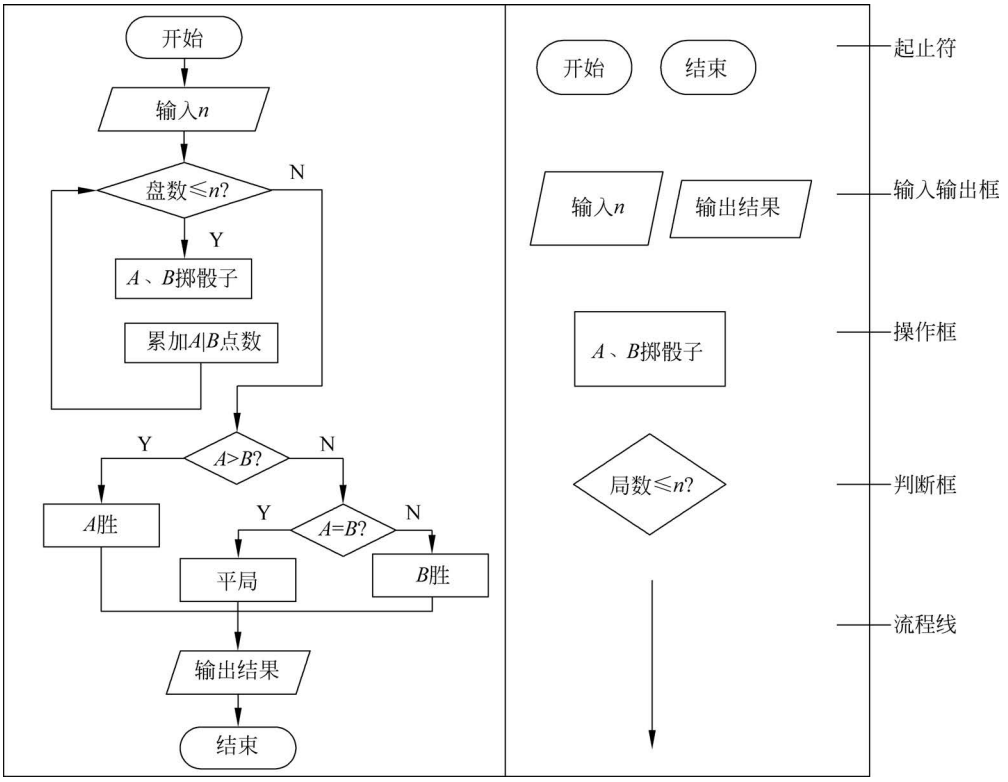


图 5-2　程序流程图符号及含义

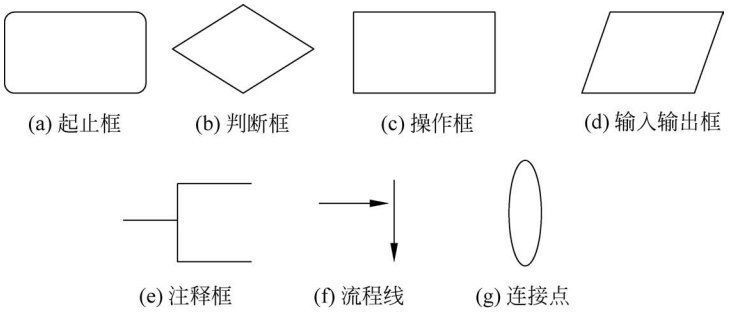


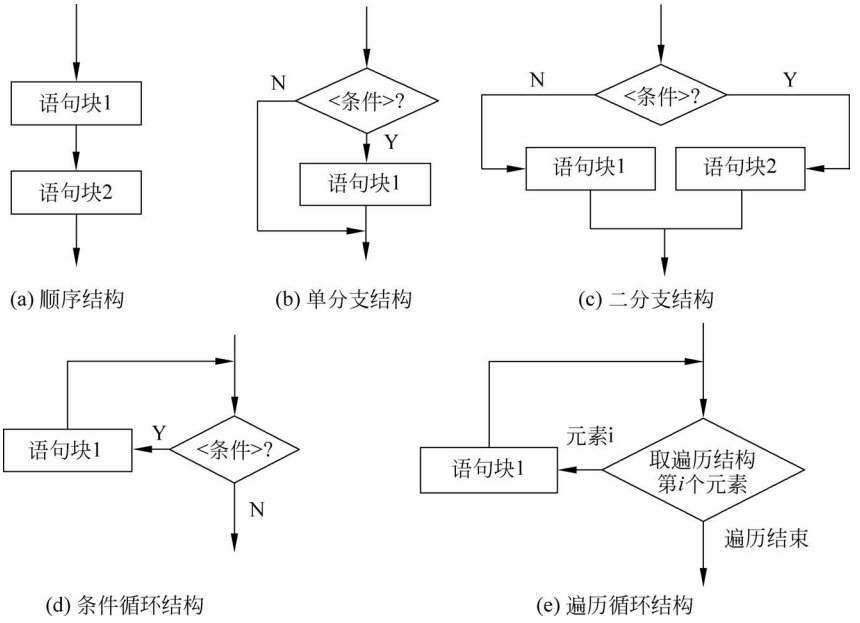
图 5-3　程序流程图基本元素

的触发条件不同，可分为条件循环结构和遍历循环结构。

图 5-4 用流程图来表示三种基本结构。

2. N-S 图

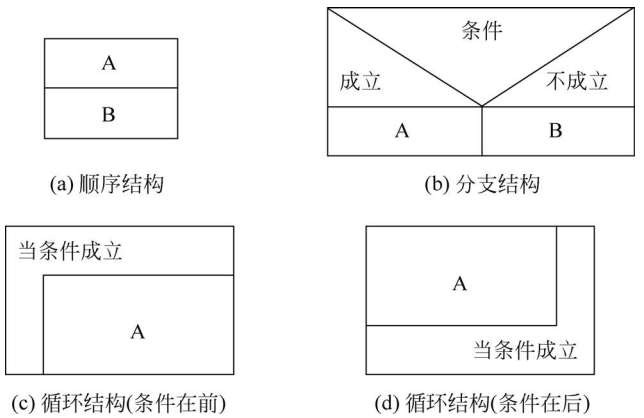
N-S 图也被称为盒图或 CHAPIN 图。1973 年，美国学者 I. Nassi 和 B. Shneiderman 提出了一种在流程图中完全去掉流程线，全部算法写在一个矩形阵内，在框内还可以包含其他框的流程图形式。即由一些基本的框组成一个大的框，这种流程图又称为 N-S 结构流程图（以两个人名字的第一个字母组成）。N-S 图包括顺序、选择和循环三种基本结构，如图 5-5 所示。



(a) 顺序结构　(b) 单分支结构　(c) 二分支结构

(d) 条件循环结构　(e) 遍历循环结构

图 5-4　程序基本结构流程图



(a) 顺序结构　(b) 分支结构

(c) 循环结构(条件在前)　(d) 循环结构(条件在后)

图 5-5　三种基本结构流程 N-S 图

5.2　程序的分支结构

5.2.1　单分支结构：if 语句

Python 中用 if 语句来表示分支结构，单分支结构的格式如下。

```
if <条件>:
  <语句块>
```

以上格式中的< if >、:和<语句块>前面的缩进都是语法的一部分。< if > 关键字与判断条件构成 if 语句，if 语句后使用:结尾，<语句块>与 if 语句之间通过缩进形成逻辑关联。

若 if 语句中的<条件>成立，则执行 if 语句后的<语句块>；若<条件>不成立，则跳过 if 语句后的<语句块>。单分支结构中的<语句块>只有“执行”和“跳过”两种情况。

【案例 5-2】 下象棋。

下象棋需要先背下各种棋的走法口诀：马走日，象走田，车走直路炮翻山，士走斜线护将边，小卒一去不回还。

```
chess = input('请输入您要走的棋子:')
if chess == '马':
  print('马走日,沿着"日"字的对角线走')
if chess == '象':
  print('象走田,沿着"田"字的对角线')
if chess == '车':
  print('车走直路,沿着直线走')
if chess == '炮':
  print('炮翻山,沿着直线走,隔山打牛法吃子')
if chess == '士':
  print('士走斜线护将边,一次一格')
if chess == '将':
  print('将在"宫"中,一次一格横斜走')
if chess == '':
  print('小卒一去不回还,一次一格只进不退')
```

案例代码中使用了 7 个 if 语句。只有当输入的内容是“马”时，才会执行 print('马走日，沿着“日”字的对角线走')，否则就会跳过这一个语句。也就是每个 if 语句都会先判断当前的输入是否与自己的条件相符，条件成立则执行该 if 里面包含的语句；条件不成立，则该 if 语句被跳过。

5.2.2 二分支结构：if…else 语句

Python 中用 if…else 语句来表示二分支结构，格式如下。

```
if <条件>:
  <语句块 1>
else:
  <语句块 2>
```

若 if 语句中的判断条件成立，则执行 if 语句后的<语句块 1>；若条件不成立，则跳过 if 语句后的<语句块 1>，执行 else 后的<语句块 2>。

如果将案例 5-2 中棋子名的判断只按两级判断：有这种棋、没有这种棋，其代码则可改成：

```
chess = input('请输入您要走的棋子:')
if chess in '马象车跑士将卒':
  print('马走日,象走田,车走直路炮翻山,士走斜线护将边,小卒一去不回还')
else:
  print('象棋中没有这种棋子')
```

运行程序时，只要输入的是“马象车跑士将卒”中的任何一个名称，“print('马走日，象走

田，车走直路炮翻山，士走斜线护将边，小卒一去不回还')"即会被执行。else 及"print('象棋中没有这种棋子')"将被跳过。

二分支结构还有如下更为简洁的表达方式，适合通过判断返回特定值。

```
<表达式 1>  if <条件>  else  <表达式 2>
```

例如，判断数值 n 是否为偶数，是则返回 True，否则返回 False。用简洁版紧凑格式可写成：

```
n = eval(input('请输入整数 n 的值:'))
True  if  n % 2 == 0  else  False
```

5.2.3　多分支结构：if…elif…else 语句

Python 中用 if…elif…else 语句来表示多分支结构，格式如下。

```
if <条件 1>:
  <语句块 1>
elif <条件 2>:
  <语句块 2>
…
else:
  <语句块 N>
```

多分支结构流程图如图 5-6 所示。

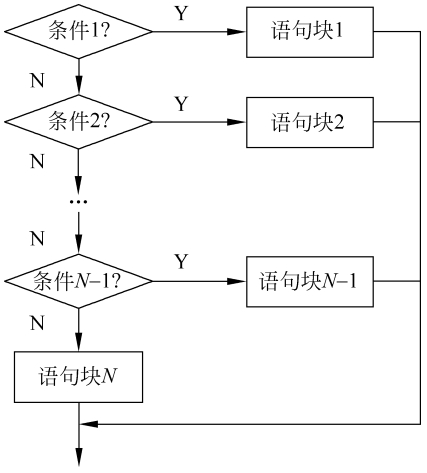


图 5-6　多分支结构流程图

对于前面的案例中是哪种棋子名的判断中，无论输入哪个棋子名，7 个 if 语句均会被执行，这样其实是一种冗余。如果改用多分支结构对案例 5-2 的棋子判断代码进行修改，使得 7 个分支只有一个分支会被执行，可写成：

```
chess = input('请输入您要走的棋:')
if chess == '马':
```

```
    print('马走日,沿着"日"字的对角线走')
elif chess == '象':
    print('象走田,沿着"田"字的对角线')
elif chess == '车':
    print('车走直路,沿着直线走')
elif chess == '炮':
    print('炮翻山,沿着直线走,隔山打牛法吃子')
elif chess == '士':
    print('士走斜线护将边,一次一格')
elif chess == '将':
    print('将在"宫"中,一次一格横斜走')
elif chess == '卒':
    print('小卒一去不回还,一次一格只进不退'  )
else:
    print('象棋中没有这种棋子'))
```

与案例 5-2 的代码相比较，将案例 5-2 中的第二个及后面的 if 改成 elif 和 else，在写条件时，例如，第一个 elif 中的条件“chess ==='象'”，其实隐含一个条件：chess!='马'，同样所有后面的 elif 的条件判断中都隐含去除它前面所有 if 和 elif 中设置的条件。

【案例 5-3】 身体质量指数 BMI。

随着生活水平的提高，人们越来越关注“身体质量”。身体质量指数(Body Mass Index，BMI)是国际上常用来衡量人体肥胖程度和是否健康的重要标准。BMI 通过人体体重和身高两个数值获得相对客观的参数，并用这个参数所处范围来衡量身体质量。参考标准如表 5-1 所示。

$$BMI=体重(kg)/身高(m^2)$$

例如，一个人的身高 1.8m，体重 80kg，则他的 BMI 值为 $80/1.8^2=24.69$，按国际标准为正常，按国内标准为偏胖。

表 5-1 BMI 指标分类

分　类	国际 BMI 值/kg·m^{-2}	国内 BMI 值/kg·m^{-2}
偏瘦	＜18.5	＜18.5
正常	18.5～25	18.5～24
偏胖	25～30	24～28
肥胖	≥30	≥28

编写根据输入的身高和体重计算 BMI 值的程序，同时输出国际和国内的 BMI 指标建议值。

案例分析：

输入数据：需要接收两个输入的数据，分别赋值给表示身高和体重的变量。

数据处理：将输入的数据按 BMI 计算公式进行计算，并将结果赋给变量。

计算结果判断：利用多分支结构对于计算结果按照表 5-1 的范围进行分类。

输出结果：将分类结果按格式进行输出。

程序编写时可以将两个判断标准分开进行，也可将两个标准融合在一起进行判断。

BMI 指标各处独立判断，代码如下。

```
h,w = eval(input('请输入身高(m)和体重(kg)[逗号隔开]:'))
BMI = w /pow(h,2)
INT,CHN = '',''
if BMI < 18.5:
  INT = '偏瘦'
elif BMI < 25:
  INT = '正常'
elif BMI < 30:
  INT = '偏胖'
else:
  INT = '肥胖'

if BMI < 18.5:
  CHN = '偏瘦'
elif BMI < 24:
  CHN = '正常'
elif BMI < 28:
  CHN = '偏胖'
else:
  CHN = '肥胖'
print('BMI 数值为{:.2f},BMI 国际指标{},国内指标{}'.format(BMI,INT,CHN))
```

在这里将 18.5≤BMI＜25 简写成 BMI＜25，原因是 elif 指的条件是去除了它之前的 if 中限定的 BMI＜18.5 这个范围了，剩下的范围则只能是 BMI≥18.5，所以 elif 中的条件 BMI＜25 中其实还有一个隐含的条件 BMI≥18.5，实质上条件范围是 18.5 ≤BMI＜25。

BMI 指标融合代码如下。

```
h, w = eval(input('请输入身高(m)和体重(kg)[逗号隔开]:'))
BMI = w / pow(h,2)
INT, CHN = '',''
if  BMI < 18.5:
  INT,CHN = '偏瘦','偏瘦'
elif  BMI < 24:
  INT,CHN = '正常','正常'
elif  BMI < 25:
  INT,CHN = '正常','偏胖'
elif  BMI < 28:
  INT,CHN = '偏胖','偏胖'
elif  BMI < 30:
  INT,CHN = '偏胖','肥胖'
else:
  INT,CHN = '肥胖','肥胖'
print('BMI 数值为{:.2f},BMI 国际指标{},国内指标{}'.format(BMI,INT,CHN))
```

两组代码相比较而言，第一种代码的可读性更好，代码清晰明了，容易调试和修改。第二种代码虽然行数更少，但是相对而言难以理解。对于程序设计来说，程序的简洁性和可读性更为重要，因而更推荐第一种写法。

需要注意的是，在使用 if…elif 这种结构时，一定要注意前后条件是否存在包含关系。例如，下列判断成绩等级的代码中，90 及以上代表优秀，60～89 代表合格，60 以下代表不合格。如果代码写成：

```
score = int(input())
if score >= 60:
  print('合格')
elif score >= 90:
  print('优秀')
else:
  print('不合格')
```

当输入成绩为 95 时，不会输出“优秀”，而是输出“合格”，原因是在第一个条件判断时，范围指的是 60 及以上的，而 95 属于这个范围，所以无论输入哪个数值，elif score＞＝90 这个判断都不会被执行。一般在设置这种有前后关联的条件时，按照从大到小的顺序则用＞或＞＝的方式来设置条件，而按从小到大的顺序时则用＜或＜＝来设置条件。例如，成绩等级判断的代码可以改成下列两种方式。

按从大到小的顺序设条件：

```
score = int(input())
if score >= 90:
  print('优秀')
elif score >= 60:
  print('合格')
else:
  print('不合格')
```

按从小到大的顺序设条件：

```
score = int(input())
if score < 60:
  print('不合格')
elif score < 90:
  print('合格')
else:
  print('优秀')
```

5.2.4 分支嵌套结构

Python 中分支嵌套结构格式如下。

```
if <条件 1>:
  <语句块 1>
    if <条件 2>:
    <语句块 2>
    else:
    <语句块 3>
else:
…
```

流程结构图如图 5-7 所示。

【案例 5-4】 用户登录。

提示用户输入用户名，然后再提示输入密码。如果用户名是"admin"并且密码是

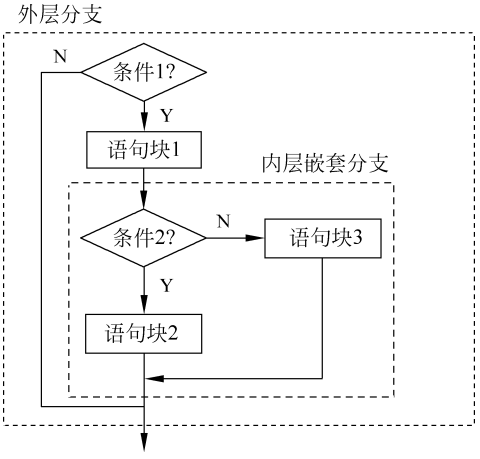


图 5-7　分支嵌套结构流程图

“888888”则提示正确，否则提示“密码错误”，如果用户名不是 admin 则提示“用户名不存在”。

```
name = input("请输入用户名:")
pwd = input('请输入密码:')
if name == 'admin':
  if pwd == '888888':
    print('欢迎登录')
  else:
    print('密码错误')
else:
  print('用户名不存在')
```

【案例 5-5】　闰年的判断。

如果年份能被 4 整除但是不能被 100 整除，或者这个年份能被 400 整除，那么这一年是闰年，否则是平年。编写程序，用分支嵌套的方式实现闰年的判断。

在第 3 章中通过 and、or 这样的逻辑运算符将两种情况的表达式进行连接实现闰年判断。在这一章中，用分支嵌套的方式来实现。在进行分支判断时，需要注意条件的包含关系，要么从大到小，要么从小到大层层递进进行判断，否则会出现逻辑错误，得不到预期的结果。

在这里用从小到大的方式，先从能否被 4 整除开始，如果满足，再判断能否被 100 整除，最后判断能否被 400 整除。实现代码如下。

```
year = int(input('请输入年份:'))
if year % 4 == 0:
  if year % 100 == 0:
    if year % 400 == 0:
      print('{}年是闰年'.format(year))
    else:
      print('{}年不是闰年'.format(year))
  else:
    print('{}年是闰年'.format(year))
else:
    print('{}年不是闰年'.format(year))
```

运行结果：

```
请输入年份:2020
2020 年是闰年
```

各种分支结构都可以嵌套使用，但是过多的嵌套会导致程序逻辑混乱，降低程序的可读性，增加程序维护的难度，因此，在进行程序开发时应仔细梳理程序逻辑，尽量避免多层嵌套。

5.3 循环结构

Python 的循环结构分为 for 循环和 while 循环两种。其中，for 循环确定循环次数，称为“遍历循环”，循环次数采用遍历结构中的元素个数来体现。while 循环不确定循环次数，称为“不定循环”，不能明确循环体可能的执行次数，而是通过条件判断是否继续执行循环体。

5.3.1 遍历循环：for 循环

遍历循环是逐一访问目标中的数据，例如，逐个访问字符串的字符、逐个访问列表中的元素等。Python 一般使用保留字 for 遍历循环，语法格式如下。

```
for <循环变量>  in  <遍历结构>:
  <语句块>
```

for 语句中的循环执行次数是根据遍历结构中元素个数确定的，遍历循环可以理解成从遍历结构中逐一提取元素，放在循环变量中，对于所提取的每个元素执行一次<语句块>。

<遍历结构>可以是字符串、文件、组合数据类型或 range()函数等。

<循环变量>用于保存本次循环中访问到的遍历结构中的元素。


观看视频

1. 遍历元素

使用 for 循环遍历字符串、列表等组合类型。

遍历字符串，代码如下。

```
#遍历字符串
s = 'ABCDE'
for c in s:
  print(c,end = ', ')
```

运行结果：

```
A,B,C,D,E
```


观看视频

遍历列表，代码如下。

```
ls = ['abc',123,'Python','X']
for t in ls:
  print(t)
```

运行结果：

```
abc
123
Python
X
```

从遍历字符和列表的运行结果可以发现，遍历是按字符串、列表中的元素有序地进行访问，先从索引号为 0 的开始，一直到最后一个。一次循环取一个元素，将元素的值赋给循环变量。

【案例 5-6】 日历的输出。

现有一个列表存储了 2021 年 1 月的日期，一个列表存储了星期的数据。请按从星期日到星期六的顺序方式将星期几和日期输出。已知 1 月 1 日是星期五，每个数字占 8 个字符的宽度，数字之间以竖线“|”分隔。输出结果如下。

```
  星期日   |星期一   |星期二   |星期三   |星期四   |星期五   |星期六   |
                                          |   1     |   2
    3      |   4     |   5     |   6     |   7     |   8     |   9     |
   10      |  11     |  12     |  13     |  14     |  15     |  16     |
   17      |  18     |  19     |  20     |  21     |  22     |  23     |
   24      |  25     |  26     |  27     |  28     |  29     |  30     |
   31      |
```

案例分析：

先对星期几的列表进行遍历，将星期输出在一行中。再确定 1 日的位置，在 1 日前面输出相应数量的空格。

再对日期列表进行遍历，将日期从 1 号开始遍历输出，每输出一个日期，就将位置值加 1。如果输出的位置是 7 的倍数时，则换行，否则不换行输出日期。

实现代码如下。

```
January = [i for i in range(1,32)]                    #用循环生成日期列表
week = '日一二三四五六'
for w in week:
  print('{:^5}'.format('星期' + w),end = '|')
print()
position = 5
print(' ' * 9 * 5,end = '')
for date in January:
  print('{:^8}'.format(date),end = '|')
    position += 1


  if position % 7 == 0:
    print()
```

2. range()函数

range()函数可以创建一个整数列表。range()函数的语法格式如下。


观看视频

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

函数说明：

start：表示列表的起始位置，该参数可以省略，省略则表示列表默认从 0 开始。

stop：表示列表的结束位置，开区间，即不包含 stop 的值，如 range(6)、range(0,6)表示结束的值为 5，产生的列表为[0,1,2,3,4,5]。

step：表示列表中元素的增幅，该参数可以省略，省略则表示元素默认步长为 1，如 range(0,6) 相当于 range(0,6,1)。

range()函数一般与 for 循环搭配使用，以控制 for 循环中代码段的执行次数。例如，对上面的字符串、列表的遍历也可以由 range()函数来控制，range()函数的结束值为字符串、列表的长度。相应的代码修改如下：

字符串遍历代码。

```
s = 'ABCD'
for i in range(len(s)):
  print(s[i])
```

列表遍历代码：

```
ls = ['abc',123,'Python','X']
for i in range(len(ls)):
  print(ls[i])
```

for 与 range()函数的搭配使用在 Python 中应用非常频繁，一般非元素遍历而又明确循环次数的均可使用这种搭配。

【案例 5-7】 计算 1～100 的累加之和。

案例分析：要计算 1～100 的累加之和，可以使用循环语句，将 1～100 的数字逐一取出来进行累加。可以使用 for-range()的搭配，循环次数为 100 次，结束值为 100。算法流程图如图 5-8 所示。

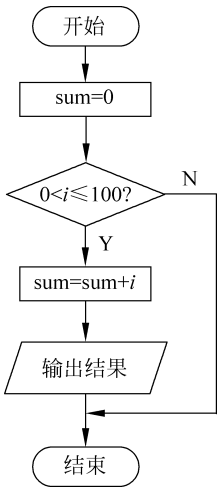


图 5-8　1～100 累加流程图

程序代码如下。

```
sum = 0
for i in range(101):
  sum += i
print('1 + 2 + 3 + ... + 100 = ',sum)
```

运行结果：

```
1 + 2 + 3 + ... + 100 = 5050
```

【案例 5-8】 打印出所有的水仙花数。所谓“水仙花数”是指一个三位数，其各位数字立方和等于该数本身。例如，153 是一个水仙花数，因为 $153=1^3+5^3+3^3$。

案例分析：最小的三位数是 100，最大的三位数是 999，需要在 100～999 的范围内逐一遍历，对每一个数值分别取出它的百

位、十位和个位，查看这三位数字的 3 次方之和是否与这个数值本身相等，如果相等，则输出。

如何取出一个三位数的百位、十位和个位呢？

方法一：个位直接用这个数对 10 取余即可获取，百位可用数值对 100 整除获取，十位则需要先用数值对 10 整除，再用整除的结果对 10 取余。

方法二：可以先将这个数值转换成字符串，用取子串的方式逐一取出对应位的数字，再将取出的数字转换为整型。

方法一的代码如下。

```
for i in range(100,1000):
  a = i //100                          #获得百位
  b = i //10 % 10                      #获得十位
  c = i %10                            #获得个位
  if a**3 + b**3 + c**3 == i:
    print(i)
```

方法二的代码如下。

```
for i in range(100,1000):
  num = str(i)
  a = int(num[0] )                     #获得百位
  b = int(num[1] )                     #获得十位
  c = int(num[2] )                     #获得个位
  if a**3 + b**3 + c**3 == i:
    print(i)
```

运行结果：

```
153
370
371
407
```

5.3.2　无限循环：while 循环


观看视频

很多应用无法在执行之初无法确定遍历结构，这就需要编程语言提供根据条件来进行循环的语法，这种循环称为无限循环，也称为条件循环。无限循环一直保持循环操作直到循环条件不满足才结束，不需要提前知道循环次数。

Python 通过保留字 while 实现无限循环，语法格式如下。

```
while <条件> :
  <语句块>
```

<条件>与 if 语句中的判断条件一样，结果为 True 或 False。

当程序执行到 while 语句时，若<条件>的结果为 True，则执行<语句块>中的内容，<语句块>执行完之后再次回到 while 语句进行判断，如此往复，直到循环<条件>的结果为

False，则终止循环，执行 while 循环结构之后的语句。

如案例 5-7 计算 1～100 的累加之和，用 while 循环结构来实现，代码如下。

```
sum = 0
i = 0
while i <= 100:
  sum += i
  i += 1
print('1 + 2 + 3 + ... + 100 = ',sum)
```

注意：<语句块>中一定要有控制<条件>变化的语句，否则会变成死循环。如上面代码的条件是“i <= 100”，循环控制变量为 i，在循环体中必然有变量 i 的值发生变化的语句，如“i＋＝1”。这个变量也称为程序维护计数器。在 for 循环结构中，循环变量是逐一取自遍历结构的，所以不需要程序维护计数器。


观看视频

【案例 5-9】 欲与天公试比高。一张厚度为 0.1mm 的纸想要与世界第一高峰珠穆朗玛峰(8848m)比高。纸张每次通过对折来增加高度，请问它对折多少次能超越珠穆朗玛峰呢？

```
high = 0.0001
n = 0
while high < 8848:
  high * = 2
  n += 1
print("纸张对折{}次高度为{},超越珠峰".format(n,high))
```

运行结果：

```
纸张对折 27 次高度为 13421.7728,超越珠峰
```

【案例 5-10】 用欧几里得距离(辗转相除)计算两个数字的最大公约数和最小公倍数。输入两个数字，求出两数的最大公约数和最小公倍数。

案例分析：

欧几里得距离法也称辗转相除法，指的是将两个数进行取余运算，取余的结果作为下一轮的除数，上一轮的除数则为下一轮的被除数。用来求最大公约数，则是当取余的结果为 0 时，当前的除数即最大公约数。最小公倍数则是两数的积除以两数的最大公约数。

要进行取余运算，一般是将大的数字作为被除数，小的数字作为除数。实现代码如下。

```
m , n = eval(input('请输入两数,用逗号分隔:'))
product = m * n
if m < n:
  m , n = n , m
while n != 0:
  m , n = n, m % n
print('最大公约数是:{}'.format(m))
print('最小公倍数是:{}'.format(product//m))
```

运行结果：

```
请输入两数,用逗号分隔:48, 32
最大公约数是:16
最小公倍数是:96
```

5.3.3　循环保留字：break 和 continue


观看视频

循环结构在条件满足时可一直执行,但是在一些特殊的情况下,程序需要终止循环,跳出循环结构。例如,玩游戏时,在游戏正在运行时,按 Esc 键,将终止程序主循环,结束游戏。

Python 中提供了两个保留字：break 和 continue,用它们来辅助控制循环执行。

1. break

break 跳出它所属的循环结构,脱离循环后程序从循环代码后继续执行。该语句通常与 if 结构结合使用。语法格式如下。

```
for <循环变量>  in  <遍历结构>:
  <语句块 1 >
  if  <判断条件> :
    break
  <语句块 2 >
```

```
while <循环条件> :
  <语句块 1 >
  if  <判断条件> :
    break
  <语句块 2 >
```

对应的流程图如图 5-9 所示。

在这种结构中,当满足循环条件时,执行<语句块 1 >的内容,若不满足<判断条件>时,继续执行<语句块 2 >的内容,如此往复。满足循环条件,同时也满足<判断条件>时,则执行完<判断条件>后就从循环中退出,即终止循环。

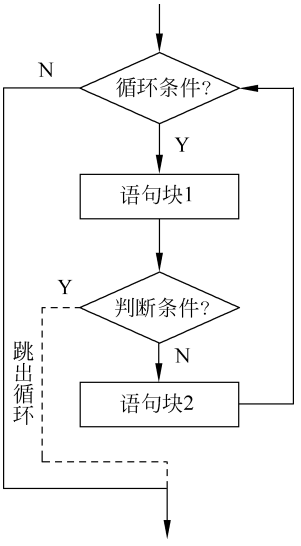


图 5-9　break 流程图

例如代码：

```
for c in 'Python':
  if c == 't':
    break
  print(c,end = ' ')
```

这段代码在字符串中遍历,从字符串“Python”中逐一取出字符进行输出,当取出的字符为“t”时,满足判断条件,则会从循环中跳出,故输出结果为

```
P y
```

【案例 5-11】 分解质因子。将一个数 $n(n>1)$的所有质因子输出来。例如 $36=2\times2\times3\times3$。

输入示例：36

输出：36＝2＊2＊3＊3

案例分析：

最小的质数为 2,所以从 2 开始循环,依次去找这个数能除尽的最小数字,如果直到找到的数字与自己相同,还未找到,则退出循环,输出所有找到的数字。

```
n = int(input())
m = n
i = 2
prime = []
while True:
  if m % i == 0:
    m = m/i
    prime.append(i)
  else:
    i += 1
  if i >= n + 1:
    break
print('{} = {}'.format(n,' * '.join(map(str,prime))))
```

2. continue

continue 与 break 的区别在于，continue 是结束本次循环，继续下一轮循环判断，而不是终止整个循环的执行；break 语句则会结束整个循环过程，不再执行循环的条件。

continue 同样与 if 语句结合使用，语法格式如下。

```
for <循环变量>  in  <遍历结构>:
  <语句块 1>
  if  <判断条件>:
    continue
  <语句块 2>

while <循环条件>:
  <语句块 1>
  if  <判断条件>:
    continue
  <语句块 2>
```

对应的流程图如图 5-10 所示。

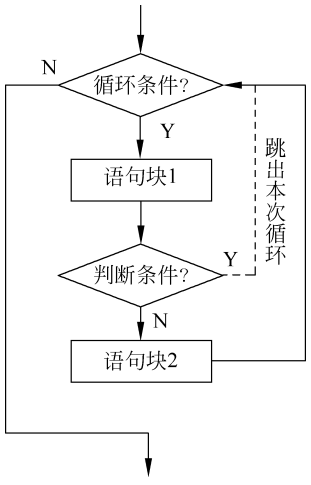


图 5-10　continue 流程图

在这种结构中，当满足循环条件时，执行<语句块 1>的内容，若不满足<判断条件>时，继续执行<语句块 2>的内容，如此往复。满足循环条件，同时也满足<判断条件>时，则执行完<判断条件>后回到循环<判断条件>，本次循环跳过<语句块 2>。

使用 continue 修改上面的代码：

```
for c in 'Python':
  if c == 't':
    continue
  print(c, end = ' ')
```

这段代码在字符串中遍历，从字符串“Python”中逐一取出字符进行输出，当取出的字符为“t”时，满足了判断条件，则会跳过判断条件后面的语句“print(c,end=' ')”，继续下一轮循环的判断，故字“t”不会被输出。运行结果为

```
Python
```

【案例 5-12】 统计个数。

编写程序统计每种不同的个位数字出现的次数。例如，给定 $N=100311$，则有 2 个 0、3 个 1 和 1 个 3。

输出格式：对 N 中每一种不同的个位数字，以 $D:M$ 的格式在一行中输出该位数字 D 及其在 N 中出现的次数 M。要求按 D 出现的顺序输出。

输入样例：100311

输出样例：

0:2

1:3

3:1

案例分析：

利用列表遍历规则，逐一遍历列表中的元素，如果元素在字符串中存在的，就跳过，否则将元素添加到字符串中，再按数字顺序进行遍历，如果遍历到的数字在字符串中，则按格式输出。

```
n = input()
s = ''
for c in n:
  if c in s:
    continue
  else:
    s += c
for i in range(10):
  if i in s:
    print('{}:{}'.format(i, n.count(i)))
```

【案例 5-13】 用穷举法计算两个数字的最大公约数和最小公倍数。

案例分析：

用穷举法来求最大公约数，则可以从两数中小的那个数字开始进行遍历，一直到 1 为止。如果遍历过程中发现某个数字能同时被两数整除，那么这个数字即为最大公约数。实现代码如下。

```
m, n = eval(input('请输入两数,用逗号分隔:'))
product = m * n
if m < n:
  m,n = n, m
for i in range(n, 0, -1):
  if m % i == 0 and n % i == 0:
    print('最大公约数是:{}'.format(i))
    break
print('最小公倍数是:{}'.format(product//i))
```

【案例 5-14】 数字黑洞。

任意一个 4 位数，只要它们各个位上的数字不是完全相同的，就有如下规律。

（1）将组成该 4 位数的 4 个数字由大到小排列，形成由这 4 个数字构成的最大的 4 位数。

（2）将组成该 4 位数的 4 个数字由小到大排列，形成由这 4 个数字构成的最小的 4 位数（如果 4 个数中含有 0，则得到的数不足 4 位）；

（3）求两个数的差，得到一个新的 4 位数（高位零保留）。

重复以上过程，最后一定会得到结果 6174。

例如，9998→0999→8991→8082→8532→6174，经过 5 次变换，得到 6174。

编写程序，判断输入的 4 位数需要经过几次变换得到 6174。

案例分析：

对输入的数字需要分离，再按大小排序。由于输入函数 input()的返回值是字符串类型，可以用 sorted() 方法对字符串的字符分离排序，得到一个列表。再将列表的元素用 join() 方法拼接起来得到一个排序后的字符串，再对两个字符串进行数据类型转换，用 int()函数将它变成整型，对转换类型后的两个数字进行减法运算，再将运算结果用 str()方法转换为字符串，以方便下一轮的分离、排序。如果减法运算得到的结果低于 4 位数，则添加 0 补齐 4 位。

实现代码如下。

```
n = input('请输入 4 位数:')
count = 0
while n!= '6174':
  max_num = sorted(n,reverse = True)
  max_num = ''.join(max_num)
  min_num = sorted(n)
  min_num = ''.join(min_num)
  n = str(int(max_num) - int(min_num))
  count += 1
  print('第 %d 次转换得到:%s' %(count,n))
print('一共进行了 %d 次转换' %count)
```

运行结果：

```
请输入 4 位数:9998
第 1 次转换得到:0999
第 2 次转换得到:8991
第 3 次转换得到:8082
第 4 次转换得到:8532
第 5 次转换得到:6174
一共进行了 5 次转换
```

5.3.4 循环与 else

1. for…else

for 循环还能与保留字 else 搭配使用，for…else 的语法结构如下。

```
for <循环变量>  in  <遍历结构>:
  <语句块 1>
```

```
else:
  <语句块 2>
```

else 后的<语句块 2>只在循环正常执行完成之后才执行。因此可以在<语句块 2>中放置判断循环执行情况的语句，如下列代码。

```
for c in  'ABC':
  print('循环进行中:' + c)
else:
  print('循环正常结束:')
```

运行结果：

```
循环进行中:A
循环进行中:B
循环进行中:C
循环正常结束
```

【案例 5-15】 质数的判断。输入一个正整数，检查该数是否为质数。例如，输入 34，输出结果为：34 不是质数。再如，输入 53，输出结果为：53 是质数。

案例分析：

质数是指某个数除了 1 和自身以外，没有其他的因子。1 不是质数，最小的质数是 2。要判断一个数 n 是否有因子，则可以在 2～n－1 的范围内逐个检验是否为 n 的因子，但是如果 n 是一个非常大的值时，会发现循环次数太多，造成时间复杂度过大，因而需要减少遍历的范围，如从 2 到 $n/2$，则可以减少一半的次数。如果数字是 16，则遍历范围可变成 2～8。但是我们发现可以再继续优化，16 的因子有 2、4、8。2 是 16 的因子，同时 16/2＝8，应该找到 2，8 就不用再检验了。再找下一个因子 4，16/4＝4，因而遍历的范围可以变成 2～16 的平方根，即可找到这个数的因子。同样是数字 n 时，对数字 n 开平方，如果得到小数，对它取整即可。

如果一个数能找到因子，那么它不是质数，如果循环结束了还没有找到因子，那么这个数就是质数。代码如下。

```
n = int(input())
if n == 1:
  print('{}不是质数'.format(n))
else:
  for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
      print('{}不是质数'.format(n))
      break
  else:
    print('{}是质数'.format(n))
```

2. while…else

无限循环也一样可以与保留字 else 搭配组成扩展模式，语法如下。

```
while <条件>：
  <语句块 1>
else:
  <语句块 2>
```

在这种模式中，当 while 循环正常执行后，程序会继续执行 else 语句中的内容。else 语句只在循环正常执行后才执行。因此，在<语句块 2>中可以放置判断循环情况的语句。例如：

```
s = 'ABC'
i = 0
while i < len(s)  :
  print('循环进行中:' + s[i])
  i += 1
else:
  print('循环正常结束:')
```

运行结果：

```
循环进行中:A
循环进行中:B
循环进行中:C
循环正常结束
```

【案例 5-16】 吹气球。已知一只气球最多能充 v 升气体，如果气球内的气体超过 v 升，气球就会炸掉。小明每天吹一次气，每次吹进去 m 升气体，由于气球慢漏气，到了第二天早上，发现少了 n 升气体。若小明从早上开始吹一只气球，请问 d 天之后气球会被吹爆吗？如果不能吹气球，则输出“气球吹不破”。要求输入的 v、m 和 d 大于 0，n 大于或等于 0。

样例输入：20,5,3,10

样例输出：第 9 天吹破气球

代码如下。

```
v,m,n,d = eval(input())
t = m
day = d - 1
while day > 0:
  if v <= t:
    print('第{}天吹破气球'.format(d - day))
    break
  t -= n
  t += m
  day -= 1
else:
  print('气球吹不破')
```


观看视频

5.3.5 循环嵌套

循环嵌套是指在一个循环体内完整地包含一个或者几个循环结构，也称为多重循环。

Python 中的两类循环 while 循环和 for 循环都可以相互嵌套，循环的嵌套也可以是多层的。

循环嵌套可以使复杂的问题结构化，把一个功能的实现拆分成多个更小的功能，然后再实现，在此实现的过程中必须要注意结构上的逻辑性和该逻辑的正确性，要保证每一个小的功能能够完全正确，最终实现一个完整的循环。循环嵌套常用于解决矩阵计算、报表打印等这类问题。

【案例 5-17】 使用嵌套语句输出如图 5-11 所示图形。

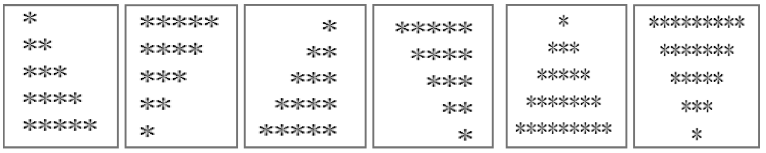


图 5-11　程序效果图

案例分析：

在控制输出过程中，嵌套循环的执行原理是：外层循环表示行数；内层循环表示列数；外层变量换到内层，达到递增递减效果。

图 5-11(a)中，符号左对齐输出，每一行符号都是递增的，一共 5 行，第一行是 1 个，第 i 行是 i 个，则可知符号个数等于行号。实现代码如下。

```
for i in range(1, 6):
  for j in range(i):
    print('*', end = '')
  print()
```

图 5-11(b)中，符号左对齐输出，每一行符号都是递减的，一共 5 行，第一行是 5 个，第 i 行是 $6-i$ 个，则可知符号个数由总行数加 1 减行号得到。实现代码如下。

```
for i in range(1,6):
  for j in range(6 - i):
      print('*', end = '')
  print()
```

图 5-11(c)中，符号右对齐输出，每一行的符号前面需要输出空格来控制右对齐。每一行的空格数是递减的，第 1 行 4 个空格，第 i 行 $5-i$ 个空格，即每一行的空格数是行数减去行号。每一行的符号都是递增的，第一行是 1 个，第 n 行是 n 个，即可知符号个数等于行号。实现代码如下。

```
for i in range(1,6):
  for k in range(5 - i):
    print(' ', end = '')
  for j in range(i):
    print('*', end = '')
  print()
```

图 5-11(d)中，符号右对齐输出，每一行的符号前面需要输出空格来控制右对齐。总行

数为5，每一行的空格数是递增的，第1行0个空格，第i行$i-1$个空格，即每一行的空格数是行号减1。每一行的符号都是递减的，第一行是5个，第i行是$6-i$个，即可知符号个数等于总行数加1减去行号。实现代码如下。

```
for i in range(1,6):
  for k in range(i-1):
    print(' ', end = '')
  for j in range(6-i):
    print('*', end = '')
  print()
```

图5-11(e)中，符号居中输出，每一行的符号前面需要输出空格来控制居中对齐。总行数为5，每一行的空格数量是递减的，第1行的空格数为4个，即$5-1$，第i行的空格数为$5-i$，即空格数等于总行数减去行号。每一行的符号都是递增的，第1行是1个，第2行是3个，第i行是$2i-1$个，则可知符号个数等于行号乘2减1。实现代码如下。

```
for i in range(1,6):
  for k in range(5-i):
    print(' ',end = '')
  for j in range(2*i-1):
    print('*',end = '')
  print()
```

图5-11(f)中，符号居中输出，每一行的符号前面需要输出空格来控制居中对齐。总行数为5，每一行的空格数量是递增的，第1行的空格数为0个，第2行的空格数为1个，第i行的空格数为$i-1$，即空格数等于行号减去1。每一行的符号都是递减的，第1行是9个，第2行是7个，第i行是$2\times5-i$个，则可知符号个数等于总行数乘2减行号。实现代码如下。

```
for i in range(5):
  for k in range(i):
    print(' ',end = '')
  for j in range(2*(5-i)-1):
    print('*',end = '')
  print()
```

【案例5-18】 使用嵌套语句输出九九乘法表。

案例分析：

由外层循环控制行，同时行号即为被乘数，用内层循环控制列，同时列号即为乘数。每一行输出内容为：行号×列号＝积。列号的最小值为1，最大值为行号。实现代码如下。

```
for i in range(1,10):
  for j in range(1,i+1):
    print('{}*{}={:<2}'.format(i,j,i*j), end = '|')
  print()
```

运行结果如图5-12所示。

```
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
```

图 5-12　九九乘法表

【案例 5-19】 蛇形矩阵是由 1 开始的自然数依次排列成的一个上三角矩阵。要求输入整数 n，生成并输出蛇形矩阵。例如，输入 6，输出下列矩阵。

```
1 3 6 10 15 21
2 5 9 14 20
4 8 13 19
7 12 18
11 17
16
```

案例分析：

一共有 n 行，有 n 个斜对角线，第 i 个斜对角线有 i 个元素。故外层循环控告行，变化范围是 0～n；内层循环控制斜对角线，范围为 0～i＋1。

元素值的变化规则：假设矩阵为 ls，元素值从数字 1 开始，每一行第一个为起始位置，即 ls[i][0]，变化方向是斜对角线递增 1，即按 ls[i][0]→ls[i－1][1]→ls[i－2][2]这个顺序进行变化，i 表示行，j 表示列，则可用 ls[i－j][j]表示。

实现代码如下。

```
n = int(input())
num = 1
ls = [[0 for i in range(n)]for i in range(n)]
for i in range(n):
  for j in range(i + 1):
    ls[i - j][j] = num
    num += 1
for i in range(n):
  for j in range(n - i):
    print('{:<6}'.format(ls[i][j]),end = '')
  print()
```


观看视频

5.4　异常处理

Python 通过 try…except 语句来进行异常处理。

异常指的是程序运行时，发生的不被期望的事件，它阻止了程序按照程序员的预期正常执行。异常发生时，是任程序自生自灭，立刻退出终止，还是做出一定的处理呢？Python 提供了一种异常处理机制。

例如，在运行下列代码时，需要进行数据输入，如果用户输入的是数字，程序会正常运

行；如果输入的是非数字，则会报错。

```
n = eval(input('请输入一个整数:'))
print(n)
```

如果运行时输入了非数字 ab，则会报以下错误。

```
NameError                          Traceback (most recent call last)
<ipython-input-6-6022e8109005> in <module>()
----> 1 n = eval(input('请输入一个整数:'))
      2 print(n)
<string> in <module>()
NameError: name 'ab' is not defined
```

可以看出，Python 解释器返回了异常信息，同时退出了程序。在这些信息中，“NameError”表示出现的异常类型；“Traceback”表示异常回溯标记；“----> 1”表示异常发生的代码行数；“NameError：name 'ab' is not defined”表示此类异常类型中异常的内容提示。

异常处理机制能让程序在异常发生时，按照代码预先设定的异常处理逻辑，针对性地处理异常，让程序尽最大可能恢复正常并继续执行，且保持代码的清晰。Python 的异常机制采用了 try、except、else 和 finally 等关键字。

try：用于监听。将要被监听的代码，即可能产生异常的代码，放在 try 语句块之内，当 try 语句块内发生异常时，异常就被触发。

except：处理 try 语句块中发生的异常。except 关键字后面可以给出异常类型，也可以省略不写。如果给出异常类型，则只处理这一类型的异常。若没有给出异常类型，则可以处理所有其他异常。

else：异常结构的 else 语句与 for…else、while…else 结构一样，当 try 语句正常执行结束，没有产生异常时，则执行 else 中的语句块，可以看成是对 try 语句块正常执行后的一种追加处理。

finally：不管是否发生异常，finally 中的语句块总是会被执行。它主要用于回收在 try 块里打开的物理资源（如数据库连接、网络连接和磁盘文件），相当于一些收尾工作放在这个语句块中。

Python 常用 try…except 语句来实现异常处理，基本格式如下。

```
try:
  <语句块 1>
except <异常类型>:
  <语句块 2>
```

语句块 1 是正常执行的程序内容，当发生异常时执行 except 保留字后面的语句块。对输入输出数据代码加上异常处理如下。

```
try:
  n = eval(input('请输入一个整数:'))
```

```
    print(n)
except NameError:
    print('输入错误,请输入一个整数!')
```

运行时同样输了非数字 ab,则会出现如下结果。

```
请输入一个整数:ab
输入错误,请输入一个整数!
```

如果要处理多种类型的异常,则可使用多个 except 语句,类似于 if…elif…else 的分支结构。基本格式如下。

```
try:
    <语句块 1>
except <异常类型 1>:
    <语句块 2>
except <异常类型 2>:
    <语句块 3>
…
except:
    <语句块 n + 2>
```

对输入输出数据代码实现多种异常处理如下。

```
try:
    n = eval(input('请输入一个整数:'))
    print(n)
except NameError:
    print('输入错误,请输入一个整数!')
except :
    print('其他错误类型!')
```

运行时同样输了非数字 ab,则会出现如下结果。

```
请输入一个整数:ab
输入错误,请输入一个整数!
```

5.5　random 库

在导入案例中,由于掷骰子的点数是不确定的,可能是 1～6 中的任意一个,这种随机数字的产生需要用到随机库 random。

随机数在计算机中的应用非常广泛,Python 内置的 random 库主要用于产生各种分布点的伪随机数序列。random 库采用了梅森旋转算法(Mersenne Twiser)生成伪随机数序列,可以用于除随机性要求更高的加密算法之外的大数工程应用。

5.5.1　random 库的常用函数

使用 random 库时,必须先导入,导入的方式常用以下两种。

方法一：import random

方法二：from random import *

使用方法一导入 random 库，每次用库中的方法时，必须加上前缀 random，如 random.random()。

使用方法二导入 random 库，不需要用 random 库名作前缀，可直接使用方法名，如 random()。

random 库中所有的函数都是基于最基本的 random.random()函数扩展实现的。表 5-2 列出了 random 库中的常用函数。

表 5-2　random 库中的常用函数

函　　数	说　　明
seed(*a* =None)	随机种子，默认值为当前系统时间
random()	生成一个[0.0,1.0)中的随机小数
randint(*a*,*b*)	生成一个[*a*,*b*]中的随机整数
getrandbits(*k*)	生成一个 *k*b 长的随机整数
randrange(start,stop[,step])	生成一个[start,stop)中以 step 为步长的随机整数
uniform(*a*,*b*)	生成一个[*a*,*b*]中的随机小数
choice(seq)	从序列类型，如列表、字符串中随机获取一个元素
shuffle(seq)	将序列类型的元素随机排列，返回打乱后的序列
sample(pop,*k*)	从 pop 类型中随机选取 *k* 个元素，以列表类型返回

使用 random 库中的方法时，每次执行的结果不一定相同，例如：

```
>>> from random  import *
>>> random()
0.8482752336931072
```

每次运行，用 random()函数得到的结果都不一样。

但是如果用了 seed()函数，那么每次运行会得到一组相同的随机数。例如：

```
seed(10)
for i in range(5):
  print(random())
```

第一次运行结果如下。后再次运行得到的还是这样的同一组随机数。

```
0.5714025946899135
0.4288890546751146
0.5780913011344704
0.20609823213950174
0.81332125135732
```

randint()函数随机生成一个指定闭区间中的整数。例如：

```
for i in range(5):
  print(randint(1,5) ,end = ',')
```

运行结果：

```
1,4,3,5,1,
```

randrange()函数随机生成一个指定范围(左闭右开)中的整数，例如，生成5个在[1,5)范围内的随机数：

```
for i in range(5):
  print(randrange(1,5),end = ',')
```

运行结果：

```
2,2,1,3,4,
```

getrandbits()函数随机生成指定长度的二进制数对应整数。例如，getrandbits(4)的范围是0000B～1111B，对应十进制就是0～15。

```
for i in range(5):
  print(getrandbits(4),end = ',')
```

运行结果：

```
9,0,1,15,12,
```

uniform()函数与randrange()函数相似，只是生成的是左闭右开区间的随机小数。

```
for i in range(5):
  print(uniform(1,2),end = ',')
```

运行结果：

```
1.7728493463067059 ,1.328063930829693 ,1.2236656695977493 ,1.5448505559010297 ,
1.0436729440961572 ,
```

choice()函数是从指定序列中随机选择一个元素，例如：

```
ls = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(5):
  print(choice(ls),end = ',')
```

运行结果：

```
7,3,7,8,4,
```

shuffle()函数是将指定序列的顺序随机打乱，是在原序列上操作的。

```
ls = [1, 2, 3, 4, 5, 6, 7, 8, 9]
shuffle(ls)
ls
```

运行结果：

```
[3,1,9,7,8,6,4,5,2]
```

sample()函数是从指定的组合类型中随机选择指定个数的元素作为返回值，返回值类型为列表。例如：

```
dic = {'a', 'b', 'c', 'd'}
sample(dic,3)
```

运行结果：

```
['a','c','d']
```

5.5.2 random 库的应用

【案例 5-20】 随机验证码。

请编写程序，生成 10 组的随机验证码。具体要求如下。

(1) 使用 random 库，采用 10 作为随机数种子。

(2) 验证码的字符由下列字符串中的字符组成。

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
```

(3) 每个验证码长度固定为 10 个字符。

(4) 程序运行每次产生 10 个验证码，每个验证码一行。

(5) 每次产生的 10 个验证码首字符不能一样，且不能以数字开头。

(6) 按行输出 10 个验证码。

案例分析：

将组成验证码的字符作为字符串赋给变量，通过字符切片的方式从中获取。虽然知道一组验证码有 10 个，但是每次生成的验证码并不一定符合要求，所以不能用遍历循环，而是使用不定循环来实现控制验证码生成的数量。

每个验证码由 10 个字符组成，且相互间的首字符不能相同，也不能是数字。解决方法是用一个序列变量来存储首字符，初始值为 10 个数字。

验证码由 10 个字符组成，循环 10 次实现。每次循环都是从字符串中随机选取一个字符。

每次生成一个验证码，先将首字符与变量中的值进行匹配，如果已存在，则不做处理；如果不存在，则生成的验证码是有效的，将该验证码的首字符加入到存首字符的变量中，并将这个验证码输出来。

```
import random
s = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
random.seed(10)
psw_list = []
first_char = '0123456789'
```

```
print('生成的一组验证码是:')
while len(psw_list)<10:
  psw = ''
  for i in range(10):
    psw += s[random.randint(0,len(s)-1)]
  if psw[0] in first_char:
    continue
  else:
    psw_list.append(psw)
    first_char += psw[0]
    print(psw)
```

运行结果：

```
生成的一组验证码是:
KcBEKanD1F
p99VxcA4iM
wyAs1RqDlR
tQxiDX4pCN
ycLapim97t
IxX6puQJCB
EePLu3Gk2o
ApccFt1MQe
Mjh0XXgCkZ
m9wBACpRrj
```

5.6　time 库与 datetime 库

time 库是 Python 的标准库之一，主要用于系统级别的精确计时，获取当前时间并进行时间的格式化输出。time 库的使用需要引用：

```
import time
```

time 库主要包括三类函数，分别用于时间获取、时间格式化以及程序计时。

5.6.1　time 库的时间获取

time 库的时间获取主要用到了以下 4 个函数，这 4 个函数及其具体作用如表 5-3 所示。

表 5-3　time 库获取时间函数

函　数	说　明
time()	用于获取当前时间戳，即计算机内部的系统时间值，该值是一个浮点数
ctime()	获取当前的时间，并且以一个易读的方式显示，返回的结果是一个字符串
gmtime()	返回 0 时区当前时间，返回值是结构体，是方便计算机处理的时间格式
localtime()	返回当前时区时间，返回值是结构体，是方便计算机处理的时间格式

gmtime()函数和 localtime()函数返回的结构体中有很多参数，分别是当前的年、月、日、小时、分钟、秒、星期(注意，0 表示周一，5 表示周六)、当天在一年中属于第多少天以及是否为夏令时。

```
import time
print(time.time())
print(time.ctime())
print(time.gmtime())
print(time.localtime())
```

运行结果：

```
1692425851.9165916
Sat Aug 19 14:17:31 2023
time.struct_time(tm_year = 2023, tm_mon = 8, tm_mday = 19, tm_hour = 6, tm_min = 17, tm_sec =
31, tm_wday = 5, tm_yday = 231, tm_isdst = 0)
time.struct_time(tm_year = 2023, tm_mon = 8, tm_mday = 19, tm_hour = 14, tm_min = 17, tm_sec =
31, tm_wday = 5, tm_yday = 231, tm_isdst = 0)
```

5.6.2 time 库的时间格式化

对时间的格式化是指将当前时间按照用户想要的格式来进行输出，time 库的时间格式化类似于字符串的格式化，需要有展示模板和特定的格式化控制字符。

strftime()函数可以将一个时间变量转换成我们想要的字符串格式，使用格式如下。

```
strftime([格式化模板字符串],[计算机内部时间类型变量])
```

格式化控制字符有很多，常见的字符如表 5-4 所示。

表 5-4　time 库时间格式化控制字符

格式化	作　　用	格式化	作　　用
%Y	年份，取值范围是 0000～9999	%a	星期缩写，如 Mon
%m	月份，取值范围是 01～12	%H	24h 制的小时，取值范围是 00～23
%B	月份英文名称，如 January	%I	12h 制的小时，取值范围是 00～12
%b	月份缩写，Jan	%p	表示上午(AM)或者下午(PM)
%d	日期，取值范围是 01～31	%M	分钟，取值范围是 00～59
%A	星期英文名称，如 Monday	%S	秒，取值范围是 00～59

strftime()函数的“逆函数”strptime()函数，可以将一个含有时间的字符串，按照用户想要的格式提取并转换成时间。strptime()函数使用如下格式。

```
strptime([带有时间的字符串],[格式化模板])
```

```
import time
t = time.gmtime()
print(time.strftime("%Y-%m-%d %H:%M:%S",t))
t2 = '2023 年 08 月 19 日 14 时 34 分 25 秒'
print(time.strptime(t2,'%Y 年%m 月%d 日 %H 时%M 分%S 秒'))
```

运行结果：

```
2023-08-19 06:42:00
time.struct_time(tm_year = 2023, tm_mon = 8, tm_mday = 19, tm_hour = 14, tm_min = 34, tm_sec =
25, tm_wday = 5, tm_yday = 231, tm_isdst = -1)
```

如果想用 strptime()函数输出中文的时间格式，需要用 locale 库中的 setlocale()函数，例如：

```
import time
import locale
t = time.gmtime()
locale.setlocale(locale.LC_CTYPE,'chinese')
print(time.strftime('%Y年%m月%d日,%H时%M分%S秒',t))
```

运行结果：

```
2023年08月19日,16时38分38秒
```

5.6.3　time 库的计时和休眠

除了获取时间和时间的格式化以外，time 库还可以用于程序计时和程序休眠。程序计时指的是测量一段程序运行所经历的时间，程序休眠是指让程序停止运行(休眠)一段时间。

time 库中使用 perf_counter()函数来进行程序计时，返回一个 CPU 级别的精确时间计数值，单位为 s。由于这个计数值起点不确定，连续调用差值才有意义。

time 库还支持 sleep()函数，该函数可以让程序休眠指定的时间，格式为

```
sleep(s)
```

参数 *s* 表示拟休眠的时间，单位是 s，可以是浮点数。

```
import time
start = time.perf_counter()
s = 0
for i in range(10000):
  s += i
end = time.perf_counter()
print('程序循环1万次用时:{}'.format(start - end))
time.sleep(0.5)
end = time.perf_counter()
print('程序总用时:{}'.format(start - end))
```

运行结果：

```
程序循环1万次用时:-0.000933500000002141
程序总用时:-0.5014548000000048
```

5.6.4　datetime 库的时间格式化

datetime 是 Python 内置的一个处理日期和时间的标准库，可以轻松处理日期和时间，也可以进行日期和时间的格式化操作。使用时需要引用：

```
import datetime
```

datetime 库中有三个主要的日期和时间类：datetime、date 和 time。每个类都包含许

多有用的函数和方法，以处理相关的操作。

1. datetime 类

datetime 类用于处理日期和时间，包括年份、月份、日期、小时、分钟、秒钟和微秒。

datetime()：返回日期和时间的对象。

格式：

```
datetime.datetime(year,month,day,hour = 0,minute = 0,second = 0,microsecond = 0)
```

参数说明：

year：年份，介于 1～9999。

month：月份，介于 1～12。

day：日期，介于 1～31(取决于月份)。

hour：小时，介于 0～23。

minute：分钟，介于 0～59。

second：秒钟，介于 0～59。

microsecond：微秒，介于 0～999 999。

示例代码：

```
>>> dt = datetime.datetime(2022, 5, 1, 12, 30, 0, 0)
>>> print(dt)
2022-05-01 12:30:00
```

若不指定格式，则可使用 now()方法，获取当前系统时间格式。

```
>>> now = datetime.datetime.now()
>>> print(now)
2023-08-19 22:15:09.591140
```

常用的 datetime 类函数和方法如下。

date()：返回一个 date 对象，表示该 datetime 对象所在的日期。

time()：返回一个 time 对象，表示该 datetime 对象所在的时间。

strftime()：将 datetime 对象转换为指定格式的字符串。

replace()：用指定的属性值替换 datetime 对象中的属性值，并返回一个新的 datetime 对象。

```
>>> now = datetime.datetime.now()
>>> print(now.date())
2023-08-19
>>> print(now.time())
22:18:16.509573
>>> print(now.strftime("%Y-%m-%d %H:%M:%S"))
2023-08-19 22:22:19
>>> new_dt = now.replace(hour=12)  #将小时属性替换为 12
>>> print(new_dt)
2023-08-19 12:18:16.509573
```

2. date 类

date 类用于处理日期，包括年份、月份和日期。

date 类的格式：

```
datetime.date(year,month,day)
```

参数说明：

year：年份，介于 1～9999。

month：月份，介于 1～12。

day：日期，介于 1～31(取决于月份)。

常用的 date 类属性有 year、month、day，分别返回该 date 对象的年份、月份和日期。

```
>>> import datetime
>>> today = datetime.date.today()
>>> print(today.year)
2023
>>> print(today.month)
8
>>> print(today.day)
19
```

3. time 类

time 类用于处理时间，包括小时、分钟、秒钟和微秒。

time 类的格式：

```
datetime.time(hour = 0,minute = 0,second = 0,microsecond = 0)
```

参数说明：

hour：小时，介于 0～23。

minute：分钟，介于 0～59。

second：秒钟，介于 0～59。

microsecond：微秒，介于 0～999 999。

常用的 time 类属性有 hour、minute、second、microsecond，分别返回该 time 对象的小时、分钟、秒钟和微秒。

```
>>> now = datetime.datetime.now()
>>> t = now.time()
>>> print(t.hour)
22
>>> print(t.minute)
27
>>> print(t.second)
30
>>> print(t.microsecond)
308041
```

4. datetime 类其他常用方法

datetime.timedelta：表示两个日期或时间之间的差异(例如，两个日期之间的天数)，精确到微秒。

datetime.strptime()：把格式化的字符串转换为日期对象。

datetime.strftime()：把日期对象格式化为字符串。

datetime.timetuple()：返回一个 time.struct_time 对象，具有包含 9 个元素的命名元组接口。

```
import datetime
now = datetime.datetime.now()
print("当前时间为:", now)
d = datetime.datetime(2023, 10, 12, 15, 0)
print("指定的日期和时间为:", d)
delta = datetime.timedelta(days=7) #获取两个日期之间的差异
next_week = now + delta
print("一周后的时间为:", next_week)
date_string = "2022-01-01" #把字符串转换为日期对象
date_object = datetime.datetime.strptime(date_string, "%Y-%m-%d")
print("转换后的日期为:", date_object)
date_str = date_object.strftime("%d/%m/%Y") #把日期对象转换为字符串
print("转换后的字符串为:", date_str)
```

运行结果：

```
当前时间为: 2023-08-19 22:34:20.228145
指定的日期和时间为: 2023-10-12 15:00:00
一周后的时间为: 2023-08-26 22:34:20.228145
转换后的日期为: 2023-11-01 00:00:00
转换后的字符串为: 01/11/2023
```

【案例 5-21】 今天是本学期的第几周的第几天？根据输入的开学日期和当前的日期，判断当前日期是本学期的第几周的第几天。

输入示例：

```
2023-9-1
2023-10-1
```

输出示例：今天是本学期的第 5 周的第 3 天

案例分析：

将输入的两个日期由字符串通过 datetime.strptime()方法转换为日期格式，两个日期相减就能得到 datetime.timedelta。使用 timedelta.days 属性可得到相差的天数，由天数对 7 整除可得到第几周，对 7 取余数可得到这周的第几天。由于没有第 0 周第 0 天这个说法，所以将周与天的值均加 1。

```
import datetime
start = input()
now = input()
st = datetime.datetime.strptime(start,"%Y-%m-%d")
now = datetime.datetime.strptime(now,"%Y-%m-%d")
delta = now - st
week,no = divmod(delta.days,7)
print('今天是本学期的第{}周的第{}天'.format(week+1,no+1))
```

运行结果：

```
今天是本学期的第 5 周的第 3 天
```

习题

1. 打印出“回文数”。所谓“回文数”是指一个数，无论从左向右读还是从右向左读，都是相同的。这样的数字叫作回文数字。例如，8118 是一个回文数，它的 4 个数字之和为 16。请编写程序找出所有 4 位数的数字之和为 16 的回文数。

2. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1、2、3，而 6＝1＋2＋3，因此 6 就是“完数”。又如，28 的因子为 1、2、4、7、14，而 28＝1＋2＋4＋7＋14，因此 28 也是“完数”。编写一个程序，判断用户输入的一个数是否为“完数”。

3. 分别用穷举法和欧几里得距离（辗转相除）法求两个指定数字的最大公约数和最小公倍数。

4. 请编写程序实现：输入一个正整数，检查该数是否为质数。

5. 请编写程序实现输出杨辉三角形，结果如图 5-13 所示。

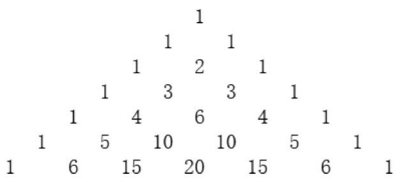


图 5-13　杨辉三角形


观看视频