

第3章

CHAPTER 3

程序控制

借助计算机的显示屏,计算机软件能以图形图像的方式输出计算结果,这些结果只是计算机存储器(严格意义上是显示存储器)中字节的映射效果。同样,借助计算机软件计算机可以播放视频图像,可以进行三维图形设计,可以制作电路板等。所有这些软件都是由计算机的基本运算实现的。

计算机运算器能够直接处理的基本运算只有二进制数码的加法(和乘法)、移位以及数码的读取和存储操作。计算机能够直接处理的数据类型为二进制数码(注:不是数值),计算机没有数的概念。而计算机控制器控制机器代码程序执行的方式只有两种,即顺序执行和跳转执行(当程序计数器指针 PC 不是指向下一个程序地址时)。对应于机器语言程序的两种执行方式,Python 语言程序具有三种程序结构,即顺序结构、分支结构和循环结构。当 Python 语言程序被“翻译”为机器语言程序时,其分支和循环程序结构将被“翻译”为跳转执行,而顺序程序结构被“翻译”为顺序执行。

高级语言程序只有顺序、分支和循环三种程序结构,这些结构可以实现所有的计算机算法(参考图灵机,1937 年)。作为一种高级语言,Python 的程序控制具有极致精简的特点,每行代码为一条执行语句,处于同一缩进位置的所有语句按位置构成顺序结构;由 if 语句或 match 语句及其下一级缩进的语句构成分支结构;由 while 语句或 for 语句及其下一级缩进的语句构成循环结构。本节将详细介绍 Python 语言程序的控制结构。

本章的学习重点:

- (1) 了解 Python 语言程序控制结构。
- (2) 掌握 match 分支结构。
- (3) 熟练掌握 if 结构、while 结构和 for 结构。
- (4) 熟练应用 while 结构实现排序算法。

3.1 顺序结构

顺序结构的程序的执行按照其语句的位置先后顺序执行。顺序结构是计算机程序的总体执行结构,只有在程序的部分功能模块中才可以采取分支结构或循环结构。

下面的程序段 3-1 为一段顺序结构程序的典型实例。要求输入平面上两个点的坐标,输出这两个点之间的距离。

程序段 3-1 文件 zym0301

```

1  import math
2  x1 = input('Please input x of Point A:')
3  y1 = input('Please input y of Point A:')
4  x2 = input('Please input x of Point B:')
5  y2 = input('Please input y of Point B:')
6  x1 = float(x1)
7  y1 = float(y1)
8  x2 = float(x2)
9  y2 = float(y2)
10 d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
11 print(f'Distance of A({x1},{y1}) and B({x2},{y2}) is: {d}.')
```

在程序段 3-1 中,第 1 行装载 math 模块,第 10 行使用了 math 模块中的 sqrt 函数(开平方函数)。

第 2 行“x1=input('Please input x of Point A: ')”调用 input 函数读取键盘的输入字符串,作为点 A 的 x 坐标,第 6 行“x1=float(x1)”将 x1 转化为浮点数,仍使用标签 x1。

同理,第 3 行读取键盘的输入字符串,作为点 A 的 y 坐标,第 7 行将 y1 转化为浮点数,仍使用标签 y1。第 4 行读取键盘的输入字符串,作为点 B 的 x 坐标,第 8 行将 x2 转化为浮点数,仍使用标签 x2。第 5 行读取键盘的输入字符串,作为点 B 的 y 坐标,第 9 行将 y2 转化为浮点数,仍使用标签 y2。

第 10 行“d=math.sqrt((x2-x1)**2+(y2-y1)**2)”调用 math 模块的 sqrt 函数计算两点间的距离。

第 11 行“print(f'Distance of A({x1},{y1}) and B({x2},{y2}) is: {d}. ')”输出两个点 A 和 B 以及它们的欧氏距离 d。

程序段 3-1 的执行结果如图 3-1 所示。

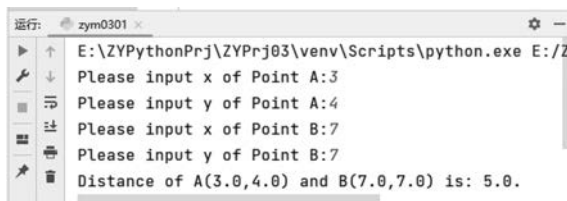


图 3-1 模块 zym0301 执行结果

在程序段 3-1 中使用两个 input 语句输入一个点的两个坐标,这是因为此处 input 函数从键盘获取一个字符串。可借助字符串的 split 函数将输入的单个字符串分隔为两个或多个字符串实现 input 的多输入处理。参考程序段 2-8 可知,字符串借助 split 函数分隔后将得到一个包含字符串子串的列表。

下面的程序段 3-2 实现了与程序段 3-1 相同的功能,但是使用 input 一次性输入一个点的两个坐标或两个点的全部坐标值。

程序段 3-2 文件 zym0302

```

1  import math
2  import re
3  [x1,y1] = input('Input "x1,y1" as A(x1,y1):').split(',')
4  [x2,y2] = input('Input "x2,y2" as A(x2,y2):').split(',')
```



视频讲解



视频讲解

```

5    x1 = float(x1)
6    y1 = float(y1)
7    x2 = float(x2)
8    y2 = float(y2)
9    d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
10   print(f'Distance of A({x1},{y1}) and B({x2},{y2}) is: {d}.')
11
12   [x1,y1,x2,y2] = re.split('[, ]+',input('Input "x1 y1,x2 y2" as A(x1,y1) and B(x2,y2):'))
13   x1 = float(x1)
14   y1 = float(y1)
15   x2 = float(x2)
16   y2 = float(y2)
17   d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
18   print(f'Distance of A({x1},{y1}) and B({x2},{y2}) is: {d}.')

```

在程序段 3-2 中,使用了两种方法实现多输入处理。第 3、4 行使用字符串的 `split` 方法,将输入字符串以逗号“,”作为分隔符,得到一个包含两个子串 of 列表,并将这两个子串分别赋给点的两个坐标。

第 12 行使用了正则表达式。第 2 行“`import re`”装载正则表达式模块。第 12 行“`[x1,y1,x2,y2]=re.split('[, ]+',input('Input "x1 y1,x2 y2" as A(x1,y1) and B(x2,y2):'))`”表示字符串的分隔符为逗号或空格,这里的“`[, ]+`”(方括号中有一个逗号和一个空格)表示方括号中的逗号或空格将作为分隔符,“`+`”号表示一个以上连续的逗号或空格也将作为分隔符。这样,将输入的字符串分成四个子串依次赋给 `x1`、`y1`、`x2` 和 `y2`。这里的赋值方式,本质上是一个列表“赋给”另一个列表。

程序段 3-2 中的第 13~16 行可以用列表赋值的形式写作一行,即

```
[x1,y1,x2,y2]=[float(x1),float(y1),float(x2),float(y2)]
```

程序段 3-2 的执行结果如图 3-2 所示。

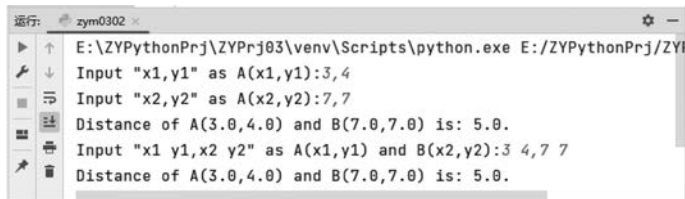


图 3-2 模块 `zym0302` 执行结果

在 Python 语言程序中,具有顺序执行结构的语句必须具有相同的缩进位置。一般地,使用 `Tab` 键控制缩进量,在 PyCharm 中,`Tab` 键默认为 4 个空格(这只是为了代码显示美观,其值可调整)。在程序段 3-2 中,各个语句的缩进量均为 0,Python 要求第一级程序代码的缩进量必须为 0,但是第二级或后续级别的代码的缩进量可以为任意数值,只需要保证同级别的(顺序执行的)语句具有相同的缩进量即可。

3.2 分支结构

程序的分支结构用于实现有条件地选择特定语句执行。Python 语言中使用缩进表示同一分支结构中的语句。例如:

```
if 条件表达式:
    语句 1
    语句 2
    语句 3
else:
    语句 4
    语句 5
    语句 6
```

上面的“if”和“else”必须位于相同的缩进位置,这两者地位相同,表示“条件表达式”为真时执行“语句 1”至“语句 3”,“条件表达式”为假时执行“语句 4”至“语句 6”。“语句 1”至“语句 3”的缩进位置必须相同,表示这三条语句属于“if”部分,即“条件表达式”为真时执行的部分。而“语句 4”至“语句 6”的缩进位置必须相同,表示这三条语句属于“else”部分,即“条件表达式”为假时执行的部分。如果“if”和“else”属于第一级缩进,那么“语句 1”至“语句 3”属于第二级缩进;“语句 4”至“语句 6”也属于第二级缩进。但是,“语句 1”至“语句 3”不需要与“语句 4”至“语句 6”处于相同的缩进位置。

同样,Python 语言中缩进也用于表示循环结构中的归属关系。

对缩进格式而言,一条语句(一般为控制语句)下的具有相同缩进量的所有语句均属于该语句的控制部分。只是为了程序代码美观,每级缩进均使用相同的缩进量。

分支结构也称选择结构,Python 语言中使用 if 或 match 关键字实现分支控制,下面通过分支结构的介绍进一步体会缩进格式的用法以及 Python 语言的简洁之美。

3.2.1 if 语句

if 语句具有四种常用结构。

(1) 单个 if 结构。

```
if 条件表达式:
    语句 1
    .....
    语句 n
```

上述结构为单个 if 结构,即当“条件表达式”为真时,执行“语句 1”至“语句 n”。

(2) 标准 if-else 结构。

```
if 条件表达式:
    语句 1
    .....
    语句 m
else:
    语句 m+1
    .....
    语句 n
```

上述结构为标准的 if-else 结构,即当“条件表达式”为真时,执行“语句 1”至“语句 m”;否则,执行“语句 m+1”至“语句 n”。如果 if 部分没有语句,用“pass”语句表示。

(3) if-elif-else 结构。

```
if 条件表达式 1:
    语句 1
```

```

.....
    语句 k
elif 条件表达式 2:
    语句 k+1
.....
    语句 m
elif 条件表达式 3:
    语句 m+1
.....
    语句 p
..... # 其他的 elif 语句
else:
    语句 t
.....
    语句 n

```

上述结构是典型的多分支结构,即如果“条件表达式 1”为真时,执行“语句 1”至“语句 k”;否则,如果“条件表达式 2”为真时,执行“语句 k+1”至“语句 m”;否则,如果“条件表达式 3”为真时,执行“语句 m+1”至“语句 p”;否则,当上述所有“条件表达式”均为假时,执行“else”部分的“语句 t”至“语句 n”。

(4) if 语句的嵌套结构。

if 结构中的 if 部分、else 部分和 elif 部分中均可再嵌入 if 结构,此时要注意各个部分的缩进关系,以保证 if 部分与其相应的 elif 或 else 部分相一致,不至于出现“张冠李戴”的问题。

下面借助以下几个问题展示一下 if 结构的几种结构。

问题 1: 求两个数 a 和 b 的较大者。

问题 2: 求一个年份 year 是否为闰年。如果 year 可以整除以 4 但不能整除以 100(称为普通闰年),或者 year 可以整除以 400(称为世纪闰年),则 year 为闰年。

问题 3: 输入一数值 x, 计算分段函数 $f(x)$ 的值。

$$f(x) = \begin{cases} x+1, & x > 1 \\ 3x-1, & 0 < x \leq 1 \\ -x^2-1, & -1 < x \leq 0 \\ 2x^3, & x \leq -1 \end{cases} \quad (3-1)$$

程序段 3-3 解决了上述三个问题。

程序段 3-3 文件 zym0303

```

1  if __name__ == '__main__':
2      [a,b] = input('Please input two numbers a,b:').split(',')
3      [a,b] = [float(a),float(b)]
4      m1 = a
5      if m1 < b:
6          m1 = b
7      print(f'Bigger of {a} and {b} is: {m1}')
8
9      if a > b:
10         m2 = a
11     else:
12         m2 = b

```



视频讲解

```

13     print(f'Bigger of {a} and {b} is: {m2}')
14
15     m3 = a
16     if m3 > b:
17         pass
18     else:
19         m3 = b
20     print(f'Bigger of {a} and {b} is: {m3}')
21
22     year = input('Please input a year:')
23     year = int(year)
24     leap = False
25     if year % 4 == 0:
26         if year % 100 != 0:
27             leap = True
28         elif year % 400 == 0:
29             leap = True
30         else:
31             leap = False
32     else:
33         leap = False
34     if leap:
35         print(f'{year} is a leap year.')
36     else:
37         print(f'{year} is a nonleap year.')
38
39     x = input('Please input a number:')
40     x = float(x)
41     y = 0
42     if x > 1:
43         y = x + 1
44     elif 0 < x <= 1:
45         y = 3 * x - 1
46     elif -1 < x <= 0:
47         y = -x ** 2 - 1
48     else:
49         y = 2 * x ** 3
50     print(f'{x} produces {y:.2f}')

```

程序段 3-3 的执行结果如图 3-3 所示。

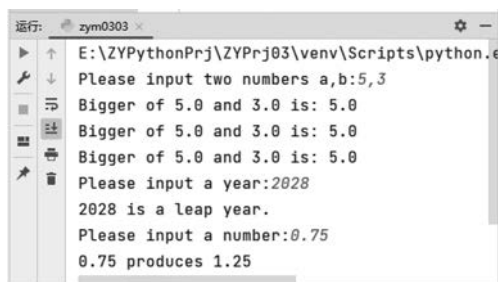


图 3-3 模块 zym0303 执行结果

在程序段 3-3 中,第 1 行“if __name__ == '__main__':”在执行模块 zym0303 时该条件表达式为真,则将执行第 2~50 行的语句。

第2行“`[a,b]=input('Please input two numbers a,b: ').split(',')`”输入两个数值形式的字符串,用逗号分隔。第3行“`[a,b]=[float(a),float(b)]`”将输入的 `a` 和 `b` 转化为浮点数。

第4~7行为一个功能模块,求解上述的“问题1”。第4行“`m1=a`”将 `a` 赋给 `m1`。第5行“`if m1<b:`”判断如果 `m1` 小于 `b`,则执行第6行“`m1=b`”将 `b` 赋给 `m1`。第7行“`print(f'Bigger of {a} and {b} is: {m1})`”输出 `a`、`b` 和它们的较大者 `m1`。这里的第5、6行为一个 `if` 结构。

第9~13行为一个功能模块,实现的功能也是求得 `a` 和 `b` 的较大者。这里的第9~12行为一个 `if-else` 结构。第9行“`if a>b:`”判断如果 `a` 大于 `b`,则执行第10行“`m2=a`”将 `a` 赋给 `m2`; 否则(第11行“`else:`”),执行第12行“`m2=b`”将 `b` 赋给 `m2`。此时,`m2` 为 `a` 和 `b` 的较大者。第13行输出 `a`、`b` 和它们的较大者 `m2`。

第15~20行为一个功能模块,仍然是求 `a` 和 `b` 的较大者。注意,第17行“`pass`”,表示不执行任何功能。

第22~37行为一个功能模块,求解上述的“问题2”。第22行“`year=input('Please input a year: ')`”输入一个表示年份的字符串赋给 `year`。第23行“`year=int(year)`”将 `year` 转化为整型,仍然用 `year` 作为标签。第24行“`leap=False`”将 `False` 赋给 `leap`(第24行可省略)。

第25~33行为一个嵌套的 `if-else` 结构。第25行“`if year % 4 == 0:`”判断如果 `year` 能被4整除,则执行第26~31行。如果第26行“`if year % 100 != 0:`”中 `year` 不能被100整除,则第27行“`leap=True`”将 `True` 赋给 `leap`; 否则如果第28行“`elif year % 400 == 0:`”中 `year` 被400整除,则第29行“`leap=True`”将 `True` 赋给 `leap`; 否则(第30行“`else:`”)执行第31行“`leap=False`”将 `False` 赋给 `leap`。

按照缩进位置关系可知,第32行的“`else:`”语句与第25行的“`if`”对应,表示如果 `year` 不能被4整除时,执行第33行“`leap=False`”将 `False` 赋给 `leap`。

第34~37行为一个 `if-else` 结构,如果第34行“`if leap:`”中的 `leap` 为 `True`,则执行第35行“`print(f'{year} is a leap year.')`”在屏幕打印 `year` 是一个闰年; 否则,执行第37行“`print(f'{year} is a nonleap year.')`”在屏幕打印 `year` 是一个平年。

第39~50行为一个功能模块,求解上述的“问题3”。第39行“`x=input('Please input a number: ')`”从键盘输入一个表达数值的字符串给 `x`。第40行“`x=float(x)`”将 `x` 转化为浮点数,仍用 `x` 作为标签。第41行“`y=0`”将0赋给 `y`。

第42~49行为一个多分支结构,对应于式(3-1)的四种情况,注意在 Python 语言中,支持这种“ $0 < x \leq 1$ ”级联的关系不等式。第42行“`if x > 1:`”如果 `x` 大于1,则执行第43行“`y=x+1`”将 `x` 与1的和赋给 `y`; 否则,如果第44行“`elif 0 < x <= 1:`”中 `x` 大于0且小于或等于1,则执行第45行“`y=3 * x - 1`”,将 $3x-1$ 的值赋给 `y`; 否则,若第46行“`elif -1 < x <= 0:`”中 `x` 大于-1且小于或等于0,则执行第47行“`y=-x**2-1`”将 $-x^2-1$ 赋给 `y`; 在上述情况都不成立时(第48行“`else:`”),执行第49行“`y=2 * x ** 3`”将 $2x^3$ 赋给 `y`。

第50行“`print(f'{x} produces {y: .2f})`”输出 `x` 和 `y` 的值。

现在,反复阅读程序段3-3,思考一下:还有没有可能进一步简化 `if` 结构的表达形式? 好像除了条件表达式后面的“`:`”,已经简化到极致了。目前在 Python 3 中,这个冒号“`:`”仍

不可少。

if 结构可以写成一条语句,形如:

```
y = 表达式 1 if 条件 else 表达式 2
```

表示当“条件”为真时,将“表达式 1”的值赋给 y,否则将“表达式 2”的值赋给 y。这里的“表达式 2”还可以嵌入新的 if 结构。例如:程序段 3-3 中第 41~49 行可以用如下的一条语句表示:

```
y = x + 1 if x > 1 else 3 * x - 1 if 0 < x <= 1 else -x ** 2 - 1 if -1 < x <= 0 else 2 * x ** 3
```

上述语句实现了式(3-1)所示的分段函数。

3.2.2 match 语句

match 多分支控制语句是 Python 3.10 新添加的控制语句,其基本语法形式如下。

```
match 表达式:
    case 表达式 1:
        语句 1
        .....
        语句 k
    case 表达式 2:
        语句 k + 1
        .....
        语句 p
    ..... # 其他的 case 情况
case _:
    语句 m
    .....
    语句 n
```

上述 match 分支的功能非常强大,表现在这里的“表达式”可以为字符串、数值、逻辑值、列表、元组和字典(见第 4 章)等,每个 case 部分可以添加条件限制,用“if 表达式”表示。当 match 后面的“表达式”与某个 case 后面的表达式匹配后,则执行该 case 部分的语句。例如,当 match 后面的“表达式”为“表达式 1”时,执行“语句 1”至“语句 k”;当 match 后面的“表达式”为“表达式 2”时,执行“语句 k+1”至“语句 p”;而当 match 后面的“表达式”与所有 case 后的表达式均不匹配时,则执行“case _:”部分中的“语句 m”至“语句 n”。

程序段 3-4 为 match 结构的简单用法实例。当输入 0 或 1 或 2 时,输出“低分”;当输入 3 或 4 时,输出“中分”;当输入 5 时,输出“高分”;当输入小于 0 或大于 5 的数时,提示输入有误。

程序 3-4 文件 zym0304

```
1  if __name__ == '__main__':
2      x = int(input('Input a number:'))
3      match x:
4          case 0 | 1 | 2:
5              print('Low mark. ')
6          case 3 | 4:
7              print('Medium mark. ')
8          case 5:
9              print('High mark. ')

```



视频讲解

```

10         case _ if x > 5:
11             print('Input is too big. ')
12         case _ if x < 0:
13             print('Input is too small. ')

```

程序段 3-4 中,第 2 行“`x=int(input('Input a number: '))`”从键盘读取一个整数,赋给 `x`。第 3~13 行为 `match` 结构,第 3 行“`match x:`”为 `match` 结构的头部,`x` 为 `match` 结构的表达式;第 4 行“`case 0 | 1 | 2:`”表示当 `x` 的值为 0 或 1 或 2 时,执行第 5 行语句“`print('Low mark. ')`”,输出“Low mark.”。这里的“`|`”表示“或”(注意,不能使用 `or`)。

第 6 行“`case 3 | 4:`”表示 `x` 的值为 3 或 4 时,执行第 7 行“`print('Medium mark. ')`”,输出“Medium mark.”。

第 8 行“`case 5:`”表示 `x` 的值为 5 时,执行第 9 行“`print('High mark. ')`”,输出“High mark.”。

第 10 行“`case _ if x > 5:`”表示当 `x` 的值不满足前面的各种 `case` 情况(也可理解为当 `x` 值为任意值时)且满足限定条件 `x` 大于 5(这里使用 `if` 设定限定条件,限定条件可以应用于任何 `case` 语句中)时,执行第 11 行“`print('Input is too big. ')`”,输出“Input is too big.”。

第 12 行“`case _ if x < 0:`”表示当 `x` 的值不满足前面的各种 `case` 情况且满足限定条件 `x` 小于 0 时,执行第 13 行“`print('Input is too small. ')`”,输出“Input is too small.”。

程序段 3-4 的典型执行结果如图 3-4 所示。

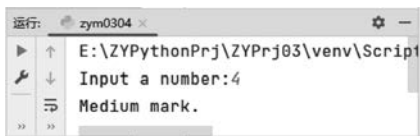


图 3-4 模块 `zym0304` 执行结果

程序段 3-5 展示了 `match` 结构使用字符串作为表达式进行分支程序控制的情况。在程序段 3-5 中,输入一个字符串 `str`,若 `str` 为“ok”,则输出“Finished”;如果 `str` 长度为 1(只包含一个字符)且为小写字母时,将其转化为大写字母输出;如果 `str`

中包含了数字 0~9、小写字母 `a~f` 或大写字母 `A~F`,则提取其中的这部分内容,添加上“0x”前缀输出。

程序段 3-5 文件 `zym0305`

```

1     import re
2     if __name__ == '__main__':
3         str = input('Input a string:')
4         match str:
5             case 'ok':
6                 print('Finished. ')
7             case _ if len(str) == 1 and 'a' < str < 'z':
8                 print(str.upper())
9             case _ if re.search('[0-9a-fA-F]', str) != None:
10                 m = re.search('[0-9a-fA-F]', str).group()
11                 print('0x' + m)
12             case _:
13                 print(str)

```

在程序段 3-5 中,第 1 行“`import re`”装载 `re` 模块,为第 9、10 行的正则表达式服务。第 3 行“`str=input('Input a string: ')`”从键盘输入一个字符串,赋给 `str`。

第 4~13 行为一个 `match` 结构,第 4 行“`match str:`”根据字符串 `str` 的值进行选择处理,当 `str` 为“ok”时(第 5 行“`case 'ok':`”成立),则执行第 6 行“`print('Finished. ')`”,输出



视频讲解

“Finished.”；如果 str 长度为 1 且为小写字母(第 7 行“case _ if len(str) == 1 and 'a' < str < 'z':”成立)时,则执行第 8 行“print(str.upper())”,输出 str 对应的大写字母；当 str 中包含数字 0~9、字母 a~f 或 A~F(第 9 行“case _ if re.search('[0-9a-fA-F]+' ,str) != None:”成立)时,则执行第 10、11 行,第 10 行“m=re.search('[0-9a-fA-F]+' ,str).group()”读取 str 中匹配了数字 0~9、字母 a~f 或 A~F 的部分(第一次匹配成功的部分),赋给 m,第 11 行“print('0x'+m)”为字符串 m 添加前缀“0x”后输出。这里的正则表达式“'[0-9a-fA-F]+'”表示由数字 0~9、字母 a~f 或 A~F 组成的字符串(方括号表示其中的字符为匹配用的字符,而“+”号表示至少包含一个上述字符)。

第 12 行“case _:”表示上述所有 case 均不成立时,执行第 13 行“print(str)”,输出 str 字符串。注意:“case _:”必须放在所有 case 的最后。

程序段 3-5 的执行结果如图 3-5 所示。

在 match 结构中,表达式为列表或元组等时,其匹配的情况分为两种:完全匹配和部分匹配。完全匹配是指 match 的表达式与 case 的情况完全相同;部分匹配是指 match 的表达式可以仅匹配 case 的部分值,而 case 中的其余“标签”视为匹配任意值。程序段 3-6 为 match 结构中的表达式为列表的实例。



图 3-5 模块 zym0305 执行结果

程序段 3-6 match 结构中的表达式为列表的情况

```
1  if __name__ == '__main__':
2      match [3,5,7]:
3          case [3,x,y]:
4              print(x)
5              print(y)
6              print(3 + x + y)
7          case _:
8              print('Any')
```

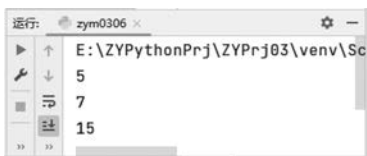


图 3-6 模块 zym0306 执行结果

在程序段 3-6 中,第 2 行“match [3,5,7]:”表明 match 的表达式为列表“[3,5,7]”,此时将与第 3 行的“case [3,x,y]:”相匹配,这里的 x 和 y 表示可匹配任意值的标签,匹配成功后,x 将为 5,y 将为 7,因此,第 4~6 行被执行,得到如图 3-6 所示结果。

在 match 结构中,列表、元组、字典等的匹配,有些类似于形式上的匹配,这使得 match 结构的功能异常强大,事实上,match 结构可以实现任何 if 结构实现的功能。学过 Mathematica(Wolfram 语言)的读者,可以感受到 Python 语言的 match 结构受到 Wolfram 语言的模式匹配的影响。Python 语言正集成众多语言的优点不断进化。

3.3 循环结构

Python 语言中,只有两个循环控制方式,即 while 结构和 for 结构。while 结构有时称为“当型”循环,即当 while 后的表达式为真时,执行 while 结构中的语句;如果 while 后的



视频讲解

表达式为假,则跳过 while 结构执行其后的语句。

Python 语言中的 while 结构与 C 语言中的 while 结构类似。但是 Python 语言中的 for 结构却与 C 语言中的 for 结构完全不同,而是类似于 C# 语言中的 foreach 结构。

Python 语言中的 for 结构仅适用于可数型的对象(即可统计对象中成员的个数),在面向对象技术的计算机语言中,将这类对象称为“可迭代”对象。例如,列表就是一个可迭代对象,因为列表的元素可数;字符串也是一个可迭代对象,因为字符串中的字符是可数的。在 Python 语言中,大部分数据类型定义的对象为“可迭代”对象。

下面依次介绍 while 结构和 for 结构。

3.3.1 while 结构

while 结构有两种基本形式,即标准的 while 循环结构和 while-else 结构。

(1) while 循环结构。

while 循环结构的语法形式如下。

```
while 条件表达式:
    语句 1
    .....
    语句 n
```

在 while 循环结构中,每次循环均需判断“条件表达式”的值,如果为真,则执行“语句 1”至“语句 n”。如果“条件表达式”的值为假,则跳出循环。

例如:计算 $1+2+\cdots+100$ 的值,其 Python 程序如程序段 3-7 所示。

程序段 3-7 文件 zym0307

```
1  if __name__ == '__main__':
2      sum = 0
3      i = 0
4      while i <= 100:
5          sum += i
6          i += 1
7      print(f'1 + 2 + ... + 100 = {sum}')
```

在程序段 3-7 中,第 2 行“sum=0”将 0 赋给 sum。第 3 行“i=0”将 0 赋给 i。

第 4~6 行为 while 循环结构,当第 4 行“while i<=100:”的条件表达式“i<=100”为真时,循环执行第 5、6 行。第 5 行“sum+=i”将 i 累加到 sum 中;第 6 行“i+=1”表示循环变量 i 累加 1。

程序段 3-7 将输出结果“ $1+2+\cdots+100=5050$ ”。

(2) while-else 结构。

while-else 结构的语法形式如下。

```
while 条件表达式:
    语句 1
    .....
    语句 m
else:
    语句 m+1
    .....
    语句 n
```



视频讲解

上述结构表示如果 while 后面的“条件表达式”为真,则循环执行“语句 1”至“语句 m”;如果“条件表达式”为假,则执行“else:”后的“语句 m+1”至“语句 n”。

表面上看“else:”部分是多余的,因为其总会被执行。通常在 while 结构中,当其后的“条件表达式”为假时,会跳出 while 结构,执行其后的语句,所以即使没有“else:”,“语句 m+1”至“语句 n”也会被执行。但是有一个例外,这个例外和 break 语句有关。

break 语句可以用于 while 结构(和 for 结构)中,用于跳出 while 结构(和 for 结构)。对于 while-else 结构而言,break 语句将跳出整个 while-else 结构,因此,如果在 while 部分执行时遇到了 break 语句,将跳出整个 while-else 结构,即 else 部分的语句将不被执行。如程序段 3-8 所示。

程序段 3-8 文件 zym0308

```
1  if __name__ == '__main__':
2      sum = 0
3      i = 0
4      while i <= 100:
5          sum += i
6          i += 1
7          if i == 51:
8              break
9      else:
10         print('Not executed.')
11     print(f'1 + 2 + ... + {i - 1} = {sum}')
```

程序段 3-8 在程序段 3-7 的基础上,添加了第 7~10 行。第 7 行“if i == 51:”如果 i 等于 51,则执行第 8 行“break”跳出 while-else 结构,去执行第 11 行“print(f'1 + 2 + ... + {i - 1} = {sum}')

由程序段 3-9 可知,break 语句用于跳出其所在的 while 结构或 while-else 结构(或 for 结构或 for-else 结构),执行这些结构后面的语句。另一个循环控制语句称为 continue,continue 语句用于循环体中时,当执行到 continue 时,将跳过其后的(循环体中的)语句,回到循环体的头部(即 while 条件表达式)继续下一次循环。如程序段 3-9 所示。程序段 3-9 实现了 100 以内的奇数的相加。

程序段 3-9 文件 zym0309

```
1  if __name__ == '__main__':
2      sum = 0
3      i = 0
4      while i < 100:
5          i += 1
6          if i % 2 == 0:
7              continue
8          sum += i
9      else:
10         print('Be executed.')
11     print(f'1 + 3 + ... + {i - 1} = {sum}')
```

在程序段 3-9 中,第 4~10 行为一个 while-else 结构。第 4 行“while i < 100:”判断当 i 小于 100 时,执行第 5~8 行:第 5 行“i += 1”循环变量 i 累加 1;第 6、7 行为一个 if 结构,第 6 行“if i % 2 == 0:”判断 i 为偶数时,执行第 7 行“continue”,跳过第 8 行返回到第 4 行



视频讲解



视频讲解

执行。当 i 累加到 100 时,第 4 行的条件表达式为假,将执行第 9、10 行,输出“Be executed.”。然后,跳出 while-else 结构,执行第 11 行,输出“ $1+3+\dots+99=2500$ ”。

break 语句和 continue 语句也可以应用于 for 结构或 for-else 结构中(见 3.3.2 节)。break 语句还常用在无限循环体中,用于跳出无限循环体,如程序段 3-10 所示。

程序段 3-10 文件 zym0310

```

1  if __name__ == '__main__':
2      while True:
3          x = input('Please input the first number:')
4          y = input('Please input the second number:')
5          x = float(x)
6          y = float(y)
7          z = x + y
8          print(f'{x} + {y} = {z}.')
9          s = input('Do you want to calculate again?')
10         if s == 'y':
11             pass
12         else:
13             print('Over')
14             break

```

程序段 3-10 的执行结果如图 3-7 所示。

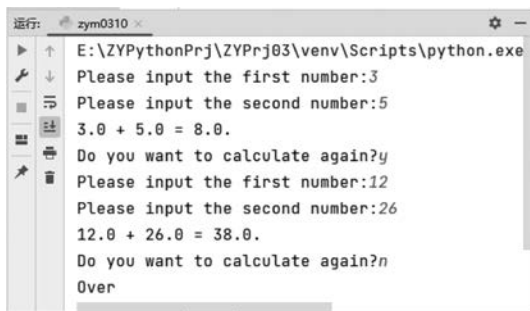


图 3-7 模块 zym0310 执行结果

在程序段 3-7 中,第 2~14 行为一个 while 结构,由于第 2 行“while True:”条件表达式始终为真,故该 while 结构为一个无限循环。第 3 行“ $x = \text{input}(\text{'Please input the first number: '})$ ”从键盘输入一个字符串 x ; 第 4 行“ $y = \text{input}(\text{'Please input the second number: '})$ ”从键盘输入一个字符串 y ; 第 5 行“ $x = \text{float}(x)$ ”将字符串 x 转化为浮点数 x ; 第 6 行“ $y = \text{float}(y)$ ”将字符串 y 转化为浮点数 y 。

第 7 行“ $z = x + y$ ”将 x 与 y 取和赋给 z 。第 8 行“ $\text{print}(f'\{x\} + \{y\} = \{z\}.')$ ”输出 x 加上 y 等于 z 的加法等式。

第 9 行“ $s = \text{input}(\text{'Do you want to calculate again?'})$ ”输入一个字符串 s 。第 10~14 行为一个 if 结构,如果第 10 行“ $\text{if } s == \text{'y'}:$ ”为真,即输入的字符串 s 为单个字符“y”,则执行第 11 行“pass”无操作,回到第 2 行循环执行; 否则(第 12 行“else:”),执行第 13、14 行: 第 13 行“ $\text{print}(\text{'Over'})$ ”输出“Over”,第 14 行“break”跳出 while 无限循环结构,程序结束。

3.3.2 for 结构

while 结构几乎可以处理所有的循环操作,既然如此,那还有没有必要再编写一种循环



视频讲解

结构？Python 语言中除了 while 结构外，还有一种循环结构，称为 for 结构。在某些情况下 for 结构是否比 while 结构更具优势？答案是肯定的。

for 结构的语法有两种形式。

(1) 基本 for 结构。

```
for 元素 in 可迭代对象:
    语句 1
    .....
    语句 n
```

上述 for 结构表示：在“元素”遍历“可迭代对象”中的全部元素的过程中，对于每一个“元素”，执行“语句 1”至“语句 n”。

(2) for-else 结构。

```
for 元素 in 可迭代对象:
    语句 1
    .....
    语句 m
else:
    语句 m+1
    .....
    语句 n
```

上述 for-else 结构表示：在“元素”遍历“可迭代对象”中的全部元素的过程中，对于每一个“元素”，执行“语句 1”至“语句 m”。当“可迭代对象”中的全部元素遍历完成后，执行“else:”部分的“语句 m+1”至“语句 n”。

表面上，有无“else:”，“语句 m+1”至“语句 n”都将在 for 部分的循环语句执行完后被执行。但是，有一个例外，即当 for 部分中含有 break 语句时，如果执行到 break 语句，将跳出整个 for-else 结构，执行 else 部分后面的语句。这种情况和 while-else 结构中遇到 break 语句的情况类似。

回到本节的开头问题：在某些情况下 for 结构是否比 while 结构更具优势？现在有一个列表，求列表中全部元素的和，在程序段 3-11 中，既可以使用 while 结构实现，也可以使用 for 结构实现。

程序段 3-11 文件 zym0311

```
1  if __name__ == '__main__':
2      a = [1,3,5,7,9]
3      s1 = 0
4      for e in a:
5          s1 += e
6      print(f'Sum of list "a" is: {s1}.')
7      s2 = 0
8      i = 0
9      while i < len(a):
10         s2 += a[i]
11         i += 1
12     print(f'Sum of list "a" is: {s2}.')
```

程序段 3-11 的执行结果如图 3-8 所示。

在程序段 3-11 中，第 2 行“a=[1,3,5,7,9]”定义列表 a。



视频讲解



图 3-8 模块 zym0311 执行结果

第 3 行“s1=0”将 0 赋给 s1。第 4、5 行为一个 for 结构,对于列表 a 中的每个元素 e(第 4 行“for e in a:”),将元素 e 加到 s1 中(第 5 行“s1+=e”)。第 6 行“print(f'Sum of list "a" is: {s1}. ')”输出图 3-8 所示的“Sum of list "a" is: 25.”。

第 7~12 行用 while 结构实现与上述 for 结构相同的功能。这里第 7 行“s2=0”将 s2 赋为 0;第 8 行“i=0”将 i 赋为 0,这里的 i 作为列表 a 的索引。第 9 行“while i<len(a):”判断当 i 小于列表的长度时,执行第 10、11 行:第 10 行“s2+=a[i]”将 a[i]累加到 s2 中;第 11 行“i+=1”列表 a 的索引号 i 自增 1。第 12 行“print(f'Sum of list "a" is: {s2}. ')”输出图 3-8 所示的“Sum of list "a" is: 25.”。

在程序段 3-11 中,比较 for 结构和 while 结构可见,在这种遍历列表等可迭代对象的情况下,for 结构具有明显的优势,即无须事先统计可迭代对象的总数。虽然在 Python 语言中,大部分的 for 循环结构均可以使用 while 结构替换,但 for 结构用于处理可迭代对象的循环操作时,效率明显比 while 结构高;while 结构则被视为一种更通用的循环结构。因此,遇到遍历可迭代对象的循环操作使用 for 结构,其他循环操作使用 while 结构。

下面的程序段 3-12 展示了 for-else 结构与 break、continue 语句的用法,该程序段用于计算 $1+3+\dots+99$ 的值。

程序段 3-12 文件 zym0312

```

1  if __name__ == '__main__':
2      a = list(range(1,100+1))
3      s = 0
4      b = 0
5      for e in a:
6          if e == a[-1]:
7              b = e - 1
8              break
9          if e % 2 == 0:
10             continue
11             s += e
12     else:
13         print('Not executed.')
14     print(f'1+3+...+{b} = {s}.')
```

在程序段 3-12 中,第 2 行“a=list(range(1,100+1))”创建一个列表 a 为[1,2,3,...,100]。这里的 range 函数生成一个对象,再使用 list 函数将 range 生成的对象转化为列表。range(n)生成一个 0 至 n-1 步长为 1 的数列,range(m,n+1)生成一个 m 至 n 步长为 1 的数列,所以,range(1,100+1)生成数列 1,2,...,100,这个数列本身也可数,故属于可迭代对象,第 5 行的语句“for e in a:”也可以替换为“for e in range(1,100+1):”。

第 3 行“s=0”将 s 赋为 0;第 4 行“b=0”将 b 赋为 0。

第 5~13 行为 for-else 结构。第 5 行“for e in a:”对列表 a 中的每个元素 e(从左向右遍



视频讲解

历), 执行第 6~11 行: 第 6~8 行为一个 if 结构, 第 6 行“if e==a[-1]:”若 e 为列表 a 的最后一个元素, 则执行第 7、8 行: 第 7 行“b=e-1”将 e-1 赋给 b; 第 8 行“break”跳出 for-else 结构, 执行第 14 行。如果第 9 行“if e % 2 == 0:”为真, 即 e 为偶数, 则执行第 10 行“continue”, 直接跳回到第 5 行执行。在第 6~10 行的两个 if 结构均不执行的情况下, 才执行第 11 行“s+=e”, 将 e 累加到 s 中。

这里的第 12、13 行的 else 部分不会被执行。由于第 6~8 行的 if 部分中的 break 语句执行时跳出了整个 for-else 结构。

第 14 行“print(f'1+3+...+{b} = {s}.')”输出结果“1+3+...+99 = 2500。”。

3.4 排序实例

数据序列的排序是循环控制的典型用法实例, 这里列举两种常用的排序方法: 冒泡排序法和选择排序法。下面首先介绍冒泡排序法。

对于一个列表 a, 设其具有 n 个元素(a[0]~a[n-1]), 使用冒泡排序法将其中元素从小至大排序的基本原理为:

(1) 把最大的数排至末尾。

从列表 a 的第 0 个元素开始, 从左向右依次比较相邻的两个元素, 将小的元素放在前面, 大的元素放在后面。请注意: 这是有重叠的比较, 相邻两次比较有一个位置重叠, 例如, 第一次比较是比较 a[0]与 a[1], 将两者中小的元素放在 a[0]、大的元素放在 a[1]; 第二次比较是比较 a[1]与 a[2], 将两者中小的元素放在 a[1]、大的元素放在 a[2]。这样处理后, 列表 a 的最后一个元素 a[-1]将为列表 a 中最大的元素。

(2) 只考虑列表中未排序的元素, 将这些元素中最大的数排至这些元素的末尾。重复这一过程, 直到列表中未排序的元素只剩下 a[0]。

程序段 3-13 为典型的冒泡排序算法实现程序。

程序段 3-13 冒泡排序法实例

```
1 import random
2 if __name__ == '__main__':
3     random.seed(299792458)
4     a = list(range(1, 10 + 1))
5     random.shuffle(a)
6     print(f'The shuffled sequence: {a}')
7     i = len(a)
8     while i > 1:
9         j = 0
10        while j < i - 1:
11            if a[j] > a[j + 1]:
12                t = a[j]
13                a[j] = a[j + 1]
14                a[j + 1] = t
15            j += 1
16        i -= 1
17    print(f'The sorted sequence: {a}')
```

在程序段 3-13 中, 第 1 行“import random”装载模块 random, random 为与伪随机数发



视频讲解

生器相关的模块。第 3 行“`random.seed(299792458)`”设置伪随机数发生器的种子为 299792458。一般地,无须设置伪随机数发生器的种子,伪随机数发生器自动使用当前计算机的时钟值作为种子,并从种子值开始迭代生成伪随机数。这里设定伪随机数发生器的种子的原因在于,可保证后续生成的伪随机数序列相同(由于使用相同的种子,读者生成的伪随机数序列与这里的完全相同)。

第 4 行“`a=list(range(1,10+1))`”生成列表 `a` 为 `[1,2,3,4,5,6,7,8,9,10]`。第 5 行“`random.shuffle(a)`”调用模块 `random` 的函数 `shuffle` 随机打乱列表 `a` 的元素。第 6 行“`print(f'The shuffled sequence: {a}')`”输出被打乱次序的列表 `a`。

第 7 行“`i=len(a)`”将列表 `a` 的长度(即元素个数)赋给 `i`。

第 8~16 行为一个 `while` 结构,第 8 行“`while i>1:`”判断如果 `i` 大于 1,则执行第 9~16 行,进行冒泡法排序。第 9 行“`j=0`”将 0 赋给 `j`,第 10~15 行为一个 `while` 结构,第 10 行“`while j<i-1:`”判断当 `j` 小于 `i-1` 时,执行第 11~15 行。由于 `i` 的初始值为序列 `a` 的长度,当第一次执行第 10~15 行时,`j` 从 0 开始按步长 1 累加到 `i-2`(第 15 行“`j+=1`”),对于 `a[j]` 和 `a[j+1]` 两个相邻元素进行排序,将其中的小数存入 `a[j]`,其中的大数存入 `a[j+1]`,这个操作由第 11~14 行的 `if` 结构实现。当 `j` 为 0 时,排序 `a[0]` 和 `a[1]`;当 `j` 为 1 时,排序 `a[1]` 和 `a[2]`;以此类推,当 `j` 为 `i-2` 时,排序 `a[i-2]` 和 `a[i-1]`(`a[i-1]` 为列表 `a` 的最后一个元素)。因此,第 10~15 行的第一次执行将序列 `a` 中最大的数保存在 `a[i-1]` 中(此时 `i=len(a)`,表示保存在序列 `a` 的最后一个元素中)。

对于每两个相邻元素,第 11 行“`if a[j]>a[j+1]:`”判断如果 `a[j]` 大于 `a[j+1]`,则执行第 12~14 行,第 12 行“`t=a[j]`”将 `a[j]` 赋给临时的 `t`;第 13 行“`a[j]=a[j+1]`”将 `a[j+1]` 赋给 `a[j]`;第 14 行“`a[j+1]=t`”将 `t` 赋给 `a[j+1]`。第 11~14 行的 `if` 结构的含义为,如果 `a[j]` 大于 `a[j+1]`,则交换这两个元素的值。

第 10~15 行的 `while` 结构中,第 15 行“`j+=1`”用于更新循环变量 `j` 的值。这里的 `j` 用作列表 `a` 的索引号。由第 9 行和第 10 行可知,`j` 在每次循环中,都是从 0 按步长 1(第 15 行)递增至 `i-2`。这个第 10~15 行的 `while` 结构可视为内循环,而第 8~16 行的 `while` 结构可视为外循环。外循环中,`i` 从 `len(a)`(即列表 `a` 的长度)按步长 1 递减(第 16 行)至 2,当 `i` 为 `len(a)` 时,内循环的操作将列表 `a` 中的最大值存入列表 `a` 的最后一个元素 `a[i-1]` 中;当 `i` 为 `len(a)-1` 时,内循环的操作将列表 `a` 中除最后一个元素外的其余全部元素的最大值保存在 `a[i-1]`(即列表 `a` 的倒数第 2 个元素)中;以此类推,当 `i` 为 2 时,内循环的操作将 `a` 中的 `a[0]` 和 `a[1]` 中的最大值保存在 `a[1]` 中。外循环的次数共 `len(a)-1` 次,内循环的次数依次为 `len(a)-1, len(a)-2, ..., 1`,共需要的循环次数为 `len(a) * (len(a)-1)/2`。这里 `len(a)` 为 10,所以冒泡法排序的循环次数为 45 次。

程序段 3-13 的流程图如图 3-9 所示。

程序段 3-13 的执行与图 3-9 中的流程图相对应。需要指出的是,标准流程图中的输入和输出部分(由于 Python 语言的文件被称为模块,这里流程图中的输入和输出模块称为输入和输出部分)应使用平行四边形,这里由于流程图的中间部分也有输出部分,故输出部分使用了矩形框。

程序段 3-13 的执行结果如图 3-10 所示。

除了上述的冒泡排序法,另一种常用的排序方法称为选择排序法。仍然设列表名为 `a`,

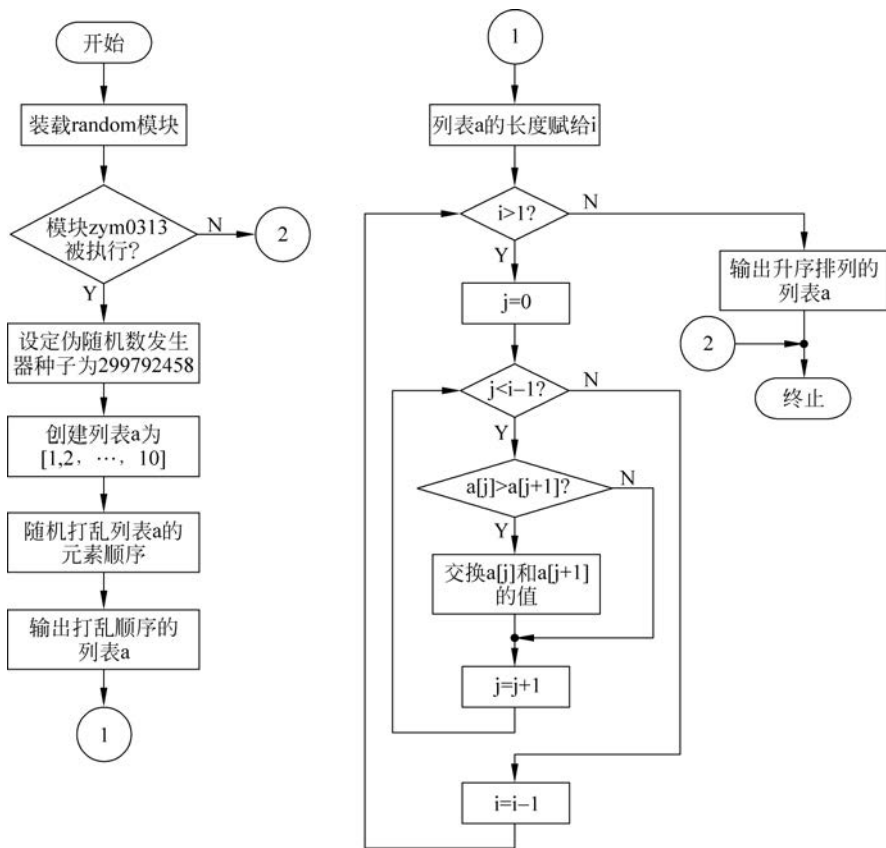


图 3-9 程序段 3-13 的流程图

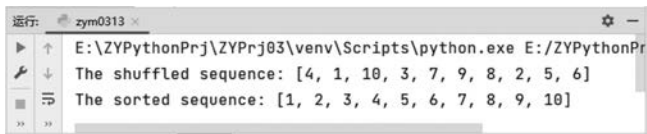


图 3-10 模块 zym0313 执行结果

具有 n 个元素($a[0]$ 至 $a[n-1]$),使用选择排序法将其中元素从小至大排序的基本原理为:

(1) 将列表 a 中最小的元素放在列表的首位置。

遍历列表 a 的元素,找出其中最小的元素,记录其索引号,将该元素与列表 a 的首元素交换位置。

(2) 将列表 a 中剩余的元素中的最小者放在剩余的元素组成的新列表的首位置。

遍历列表 a 剩余的元素,找出其中最小的元素,记录其索引号,将该元素与剩余的元素组成的新列表的首元素互换位置。

程序段 3-14 为典型的选择排序法实现程序。

程序段 3-14 选择排序法实例

```
1 import random
2 if __name__ == '__main__':
3     random.seed(299792458)
4     a = list(range(1,10 + 1))
```



视频讲解

```

5         random.shuffle(a)
6         print(f'The shuffled sequence: {a}')
7         i = 0
8         while i < len(a) - 1:
9             j = i + 1
10            k = i
11            while j < len(a):
12                if a[k] > a[j]:
13                    k = j
14                j += 1
15            if k != i:
16                t = a[k]
17                a[k] = a[i]
18                a[i] = t
19            i += 1
20        print(f'The sorted sequence: {a}')

```

在程序段 3-14 中,第 1~6 行与程序段 3-13 相同,第 5 行“random.shuffle(a)”产生一个被打乱顺序的列表 a。

第 7~19 行为选择排序法的实现代码。第 7 行“i=0”将 0 赋给 i。第 8~19 行为一个 while 结构。由第 7 行、第 8 行“while i<len(a)-1;”和第 19 行“i+=1”知,i 从 0 按步长 1 递增到 len(a)-2(即列表 a 的倒数第 2 个元素的索引号),共循环 len(a)-1 次。每次循环中,依次执行:第 9 行“j=i+1”将 i 加 1 赋给 j;第 10 行“k=i”将 i 赋给 k,k 记录遍历的数据中最小的值的索引号,这里初始化为 i;第 11~14 行为一个 while 结构,由第 9 行、第 11 行“while j<len(a):”和第 14 行“j+=1”可知,j 从 i+1 按步长 1 递增到 len(a)-1(即列表 a 的最后一个元素的索引号),对于每个 j,第 12 行“if a[k]>a[j]:”判断如果 a[k] 大于 a[j],则第 13 行“k=j”将 j 赋给 k,即 k 保存了本轮循环中列表 a 的最小值的索引号。然后,第 15 行“if k!=i:”判断如果 k 不等于 i,表明最小值的索引号不是 i(i 为没有完成排序的剩余列表的首索引号),则执行第 16~18 行,交换 a[k] 和 a[i] 的值。

当 i 等于 0 时,第 8~19 行的循环将使列表 a 的最小值保存在 a[0](即 a[i])中;循环执行一次后,i 累加为 1,再次执行第 8~19 行的循环体,使列表 a 的索引号为 1 至 len(a)-2 的元素中的最小值保存在 a[1](即此时的 a[i])中;a 继续累加 1,循环再次执行,每次执行循环,总是从列表 a 的索引号为 i 至 len(a)-2 的元素中找出最小值,然后,将这个最小值存放在 a[i]中;直到 i 为 len(a)-2,此时将 a[len(a)-2]和 a[len(a)-1]两个元素的最小值存放在 a[len(a)-2](即此时的 a[i])中,从而完成排列。

程序段 3-14 的执行结果与程序段 3-13 的执行结果完全相同,参考图 3-10。程序段 3-14 的流程图如图 3-11 所示。

选择排序法与冒泡排序法的循环次数相同,但是选择排序法中的循环体内的执行部分比冒泡法简单,所以,选择排序法的效率比冒泡排序法要高一些。

一般在排序的同时,还将记录排序后的序列的原始索引号,也就是将列表排序的同时,将由列表元素索引号组成的索引号列表同步排列,因此在对列表进行排序时,需要对列表的索引号同时进行处理。在交换列表的某两个元素时,同步交换它们的索引号,如程序段 3-15 所示。

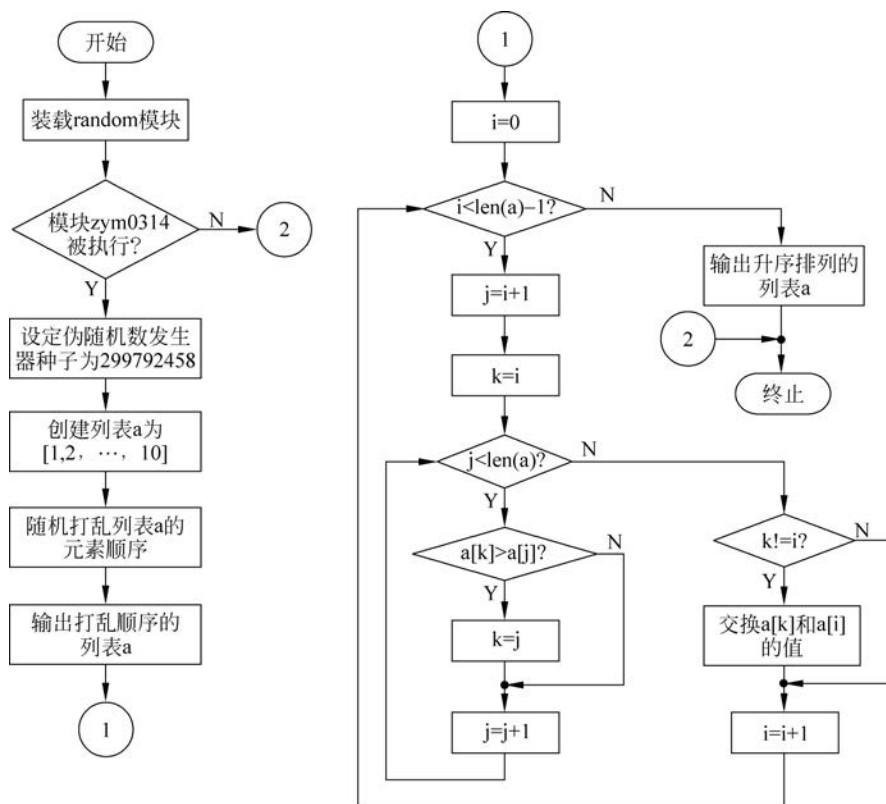


图 3-11 程序段 3-14 的流程图

程序段 3-15 列表元素排序且其索引号同步交换实例

```

1  import random
2  if __name__ == '__main__':
3      random.seed(299792458)
4      a = []
5      for e in range(10):
6          a.append(random.randint(100,200))
7      b = list(range(len(a)))
8      print(f'The list a: {a}')
9      print(f'The original index:{b}')
10     i = 0
11     while i < len(a) - 1:
12         j = i + 1
13         k = i
14         while j < len(a):
15             if a[k] > a[j]:
16                 k = j
17             j += 1
18         if k != i:
19             t = a[k]
20             a[k] = a[i]
21             a[i] = t
22             u = b[k]
23             b[k] = b[i]

```



视频讲解

```

24         b[i] = u
25         i += 1
26     print(f'The sorted list: {a}')
27     print(f'The resultant index:{b}')

```

程序段 3-15 的执行结果如图 3-12 所示。

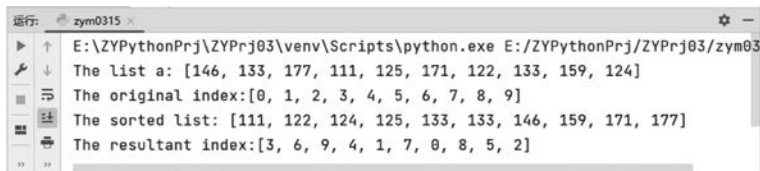


图 3-12 模块 zym0315 执行结果

结合程序段 3-15 和图 3-12 可知,第 4~6 行生成列表 a,a 中的元素为 100~200(含)内的伪随机数;第 7 行生成列表 a 的索引号列表 b,从 0~9(len(a)为 10,range(10)生成 0,1,...,9);第 8 行和第 9 行输出列表 a 和它的索引号列表 b。

第 10~25 行为选择排序法的实现代码。程序段 3-15 与程序段 3-14 相比,添加了第 22~24 行,即交换列表 a 的元素的同时,同步更新其索引号。第 26、27 行输出按升序排列的列表 a 及其相应的索引号值。

上述介绍了常用的冒泡法排序和选择法排序,但是对于列表元素的排序,在 2.4.3 节中(见表 2-8)曾指出列表本身具有 sort 方法可以快速实现排序。事实上,列表内置的 sort 排序法比冒泡法和选择法的效率都高,我们在第 5 章中将介绍一种基于递归调用的快速排序法,与内置的 sort 方法效率相当。

下面使用 for 循环替换 while 循环实现程序段 3-15 的功能,如下面程序段 3-16 所示。

程序段 3-16 用 for 循环替换 while 循环实现与程序段 3-15 相同的功能

```

1  import random
2  if __name__ == '__main__':
3      random.seed(299792458)
4      a = []
5      for e in range(10):
6          a.append(random.randint(100,200))
7      b = list(range(len(a)))
8      print(f'The list a: {a}')
9      print(f'The original index:{b}')
10     for i in range(len(a)):
11         k = i
12         for j in range(i + 1, len(a)):
13             if a[k] > a[j]:
14                 k = j
15         if k != i:
16             t = a[k]
17             a[k] = a[i]
18             a[i] = t
19             u = b[k]
20             b[k] = b[i]
21             b[i] = u
22     print(f'The sorted list: {a}')
23     print(f'The resultant index:{b}')

```



视频讲解

对比程序段 3-15 可知,程序段 3-16 中的第 10 行替换掉了程序段 3-15 中的第 10、11、25 行;第 12 行替换掉了程序段 3-15 中的第 12、14、17 行,实现了 for 循环替换 while 循环,其他内容相同。这里的第 10 行“for i in range(len(a)):”表示 i 从 0 按步长 1 递增到 len(a)-1,对于每个 i 执行第 11~21 行循环体。第 12 行“for j in range(i+1,len(a)):”表示 j 从 i+1 按步长 1 递增到 len(a)-1,对于每个 j 执行第 13、14 行的循环体。

3.5 本章小结

本章介绍了 Python 语言程序的结构与控制方式,详细介绍了分支控制结构和选择控制结构。对于分支结构,有两种实现方法,即 if-else 结构和 match-case 结构;对于循环结构,有两种实现方法,即 while 结构和 for 结构。一定意义上,基于列表数据结构和本章介绍的程序控制方法,理论上可以编写出实现任意算法的 Python 程序。

本书后续内容在这个基础上进一步介绍 Python 语言丰富的数据结构(第 4 章)、将实现特定功能的代码“包装”起来的函数和模块(第 5 章)以及面向现实世界“对象”的程序设计方法(第 6 章),这些内容将简化 Python 语言表达数据的方法并增强 Python 语言实现算法的能力。最后,需要培养的是“包”或“模块”的使用能力以及界面设计的“想象力”。例如,程序段 3-15 中第 4~6 行生成一个长度为 10 元素值为 100~200(含)内的伪随机数列表,可以借助 numpy 包中的 random 模块(import numpy as np),通过一条语句“a = np.random.randint(100, 200, size=10)”生成(同样可以使用自定义种子,例如:“np.random.seed(299792458)”)。

习题

1. 从键盘输入两个数,求这两个数的和、差、积、商(考虑除数为 0 的特殊情况)。
2. 给定一个一元二次方程式 $ax^2 + bx + c = 0$ 的系数 a、b 和 c,求该方程的根($a \neq 0$)。
3. 生成长度为 300 的列表 a,其每个元素均为 1~10 的伪随机数(由 randint(1,10)生成),求列表 a 中 1~10 各个元素的个数。
4. 设列表 a 为[8,12,5,20,23,21,14,2,11,17],使用冒泡法将 a 按降序排列。
5. 设列表 a 为[8,12,5,20,23,21,14,2,11,17],使用选择法将 a 按降序排列。
6. 设列表 a 为[8,12,5,20,23,21,14,2,11,17],使用选择法将 a 按降序排列,并同时给出排序后的元素的原索引号列表。
7. 输入一个正整数,判断其是否为素数。
8. 求 100~999 范围内的全部素数。
9. 如果一个素数的逆序仍是素数,称这两个素数为互逆序的素数对,例如,1031 和 1301 为互逆序的素数对,编程求得 1000~9999 范围内的所有互逆序的素数对。
10. 用 for 循环替换 while 循环,实现程序段 3-14 的选择法排序。