第3章 流程控制

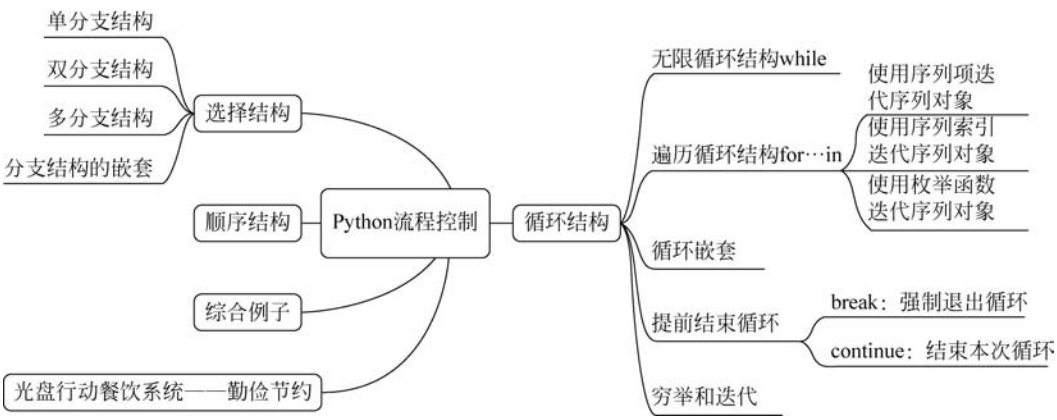
能力目标

【应知】 理解选择和循环的意义及其基本实现语句。

【应会】 掌握单分支、双分支及多分支选择结构语句的使用方法，掌握实现无限循环操作的 while 语句，掌握实现遍历操作的 for…in 语句，掌握用来提前结束循环的 break 和 continue 语句。

【难点】 嵌套语句的使用，穷举法和迭代法的使用。

知识导图



流程控制也称为控制流程，是计算机运算领域内的专用语，指在程序运行时，指令（或者程序、子程序、代码段）运行或者求值的顺序。流程控制对于任何一门编程语言来说都是至关重要的，它提供了控制程序如何执行的方法。Python 语言提供了顺序结构、选择结构和循环结构 3 种流程控制。

3.1 顺序结构


视频讲解

顺序结构是程序中最简单的流程控制结构，按照代码出现的先后顺序依次执行。程序中的代码大多是顺序执行的，其结构流程图如图 3.1 所示。

本章之前编写的代码大多数采用顺序结构。

【实例 3.1】 输出指定格式的日期。

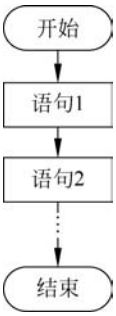


图 3.1　顺序结构流程图

```
#处理日期和时间的模块库 datetime.date 是表示日期的类,datetime.datetime 是表示日期
#时间的类
import datetime
today = datetime.date.today()    #datetime.day.today()用于获取当前的日期,返回格式
                                 #为 YYYY-mm-dd
oneday = datetime.timedelta()    #datetime.timedelta()表示两个时间之间的差
yesterday = today - oneday
tomorrow = today + oneday
print("今天是:" + str(today))
#strftime()函数接收时间元组,并返回可读字符串表示的时间,格式由参数 format 决定
print("昨天是:" + yesterday.strftime("%y/%m/%d"))
print("明天是:" + tomorrow.strftime("%m-%d-%Y"))
```

其中,%Y:4 位数的年份表示(0000～9999);

%y:2 位数的年份表示(00～99);

%m:2 位数的月份表示(01～12);

%d:月份内的某一天(1～31)。

运行结果:

```
今天是:2020-10-27
昨天是:20/10/27
明天是:10-27-2020
```


视频讲解

3.2　选 择 结 构

选择结构也称为分支结构,用于判断给定条件,然后根据判定结果来控制程序流程。例如日常生活中常见的登录即为选择结构,用户先输入用户名和密码,系统在数据库中查找并匹配。如果两者都与数据库中的记录保持一致,则“登录成功”,可以继续下面的操作;否则要重新输入或者退出系统。正如孟子曰:“鱼,我所欲也,熊掌亦我所欲也;二者不可兼得也,舍鱼而取熊掌者也。”Python 中常用的分支结构有单分支、双分支和多分支 3 种类型。

3.2.1 单分支结构

单分支结构指只有一个分支，满足判断条件执行相应语句。现实生活中的“如果天下雨，地就会湿”对应的就是单分支结构。

其结构流程图如图 3.2 所示。

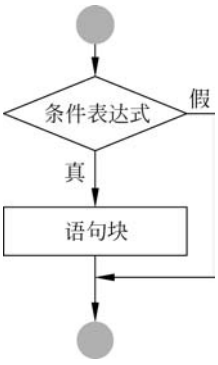


图 3.2 单分支选择结构流程图

对应的语法结构为：

```
if 条件表达式:
  语句块
```

执行过程为：先判断条件，如果执行结果为真则执行后续的语句块，否则什么也不做。

说明：

(1) “条件表达式”可以是逻辑表达式、条件表达式、算术表达式等任何类型的表达式，只要能判断非零或者非空即可。“语句块”可以是一条语句，也可以是多条语句。多条语句时，需要保证缩进对齐一致。

(2) “条件表达式”后面一定要加冒号“:”，这是初学者易犯错误的地方。

【实例 3.2】 根据出生年份判断是否为成年人。

```
1  import datetime
2  year = int(input("请输入出生年份: "))
3  if datetime.date.today( ).year - year >= 18:    #datetime.date.today( ).year 表示获
                                                   #取当前日期的年份
4      print("是成年人!")
```

运行结果：

请输入出生年份: 2015	请输入出生年份: 1990 是成年人!

【实例 3.3】 两个整数升序排列并输出。

```
1  a, b = input("input a,b: ").split(" ")   #一行输入多个数,用空格分开
2  print("排序前: " + a + "," + b)
3  if int(a) > int(b):
4      a, b = b, a   #交换 a,b 两个数的值
5  print("排序后: " + a + "," + b)
```

运行结果：

```
input a,b: 2 3
排序前: 2,3
排序后: 2,3
```

说明：在 Python 中，可以直接使用语句“a，b＝b，a”交换两个变量的值，而其他高级语言中必须引入中间变量完成交换工作，即“t＝a，a＝b，b＝t”，这正是 Python 语言的精妙之处。

拓展：可以尝试实现 3 个整数升序排列，更多数的排序需要采用其他高级数据类型实现。

3.2.2 双分支结构

若条件成立时需要执行某些操作，不成立时需要执行另外一些操作，则需要编写双分支结构。例如身份验证时，密码正确可以登录系统，密码错误则要重新输入。其结构流程图如图 3.3 所示。

对应的语法结构为：

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

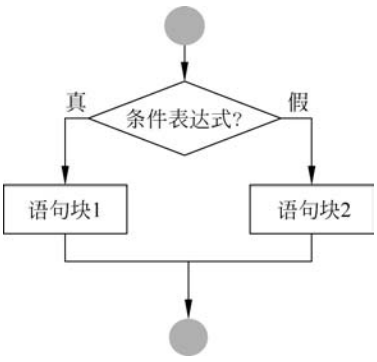


图 3.3 双分支选择结构流程图

其执行过程为：先判断条件表达式，如果结果为真或者非零，则执行语句块 1；否则执行语句块 2。

注意：

(1) 双分支结构中的 else 语句不能独立存在，即有 else 一定有相对应的 if，但有 if 不一定有 else。

(2) else 后面不需要也不能加条件表达式。

【实例 3.4】 求两个数的较大者(此例题可以使用 4 种方法实现)。

方法一：使用单分支结构

```
1  a, b = input("input a,b: ").split(" ")
2  max = int(a)
3  if (int(max) < int(b)):
4      max = int(b)
5  print("较大的数为: " + str(max))
```

方法二：使用双分支结构

```
1  a, b = input("input a,b:").split(" ")
2  if (int(a) > int(b)):
3      print("较大的数为:" + a)
4  else:
5      print("较大的数为:" + b)
```

方法三：使用三目运算符

```
1  a, b = input("input a,b: ").split(" ")
2  max = int(a) if int(a) > int(b) else
   int(b)
3  print("较大的数为: " + str(max))
```

方法四：使用内置函数 max

```
1  a, b = input("input a,b:").split
   (" ")
2  print("较大的数为:" + str(max(a,
   b)))
```

三目运算符也称为三元运算符，其语法格式为：

```
(True_statements) if (expression) else (False_statements)
```

运算规则为：先对逻辑表达式 expression 求值，如果逻辑表达式返回 True，则执行并返回 True_statements 的值；如果逻辑表达式返回 False，则执行并返回 False_statements 的值。

很明显，三目运算符是双分支结构的一种紧凑表现形式。“条件为真的语句”和“条件为

假的语句”可以放置多条语句，放置方式有两种：

（1）多条语句以英文逗号隔开，每条语句都会执行，程序返回多条语句的返回值组成的元组。如：

```
>>> a, b = input("input a,b: ").split(" ")
>>> s = "No cross, no crown.", "a 大于 b" if a > b else "a 小于或等于 b"
>>> print(s)
```

当输入“10,20”时，执行结果为：

```
('No cross, no crown.', 'a 小于或等于 b')
```

（2）多条语句以英文分号隔开，每条语句都会执行，程序只返回第一条语句的值。如：

```
>>> a, b = input("input a,b: ").split(" ")
>>> s = "No cross, no crown."; "a 大于 b" if a > b else "a 小于或等于 b"
>>> print(s)
```

当输入“10,20”时，执行结果为：

```
No cross, no crown.
```

另外，三目运算符支持嵌套，通过嵌套三目运算符，可以执行更复杂的判断。

3.2.3 多分支结构

在很多情况下，供用户选择的操作有多种，例如根据空气质量指数判断天气状况并提供生活建议，或者根据新冠肺炎疫情防控要求区分地区风险等级等。使用程序语句实现时，就可以使用多分支结构进行处理。其结构流程图如图 3.4 所示。

对应的语法格式为：

```
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 3:
    语句块 3
…
else:
    语句块 n
```

执行过程为：先判断条件表达式 1，如果结果为真，则执行语句块 1；否则判断条件表达式 2，如果结果为真，则执行语句块 2……只有在所有表达式都为假的情况下，才会执行 else 后面的语句块 n。

【实例 3.5】 计算阶梯电价（阶梯电价是按照用户消费的电量分段定价，用电价格随用电量增加呈阶梯状逐级递增的一种电价定价机制。具体规则为：当每月用电量在 0～260 度* 时为第一档，电价是 0.68 元/度；当每月用电量在 261～600 度时为第二档，260 度以内

* 即千瓦·时，此处保留习惯说法。

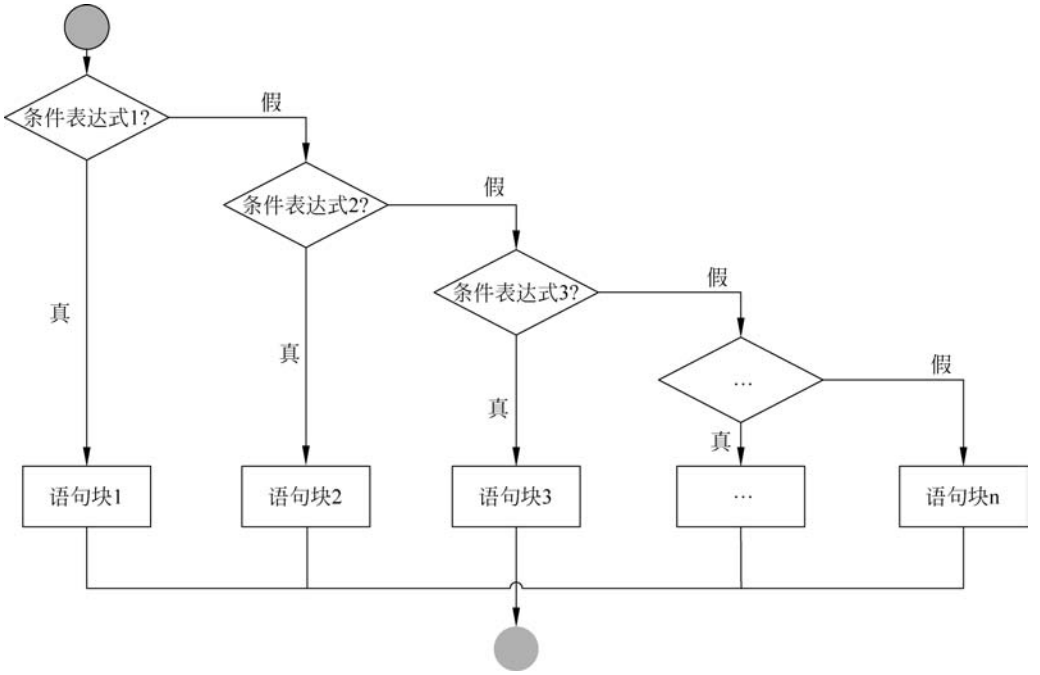


图 3.4　多分支选择结构流程图

的按照第一档收费,剩余的电按照 0.73 元/度收取;当每月用电量大于 601 度时,先分别按照第一档和第二档收费,剩余电按照 0.98 元/度收取)。

```
x = float(input("请输入每月用电量: "))
if x < 0:
    print("输入错误!")
else:
    if x <= 260:
        y = 0.68 * x
    elif x <= 600:
        y = 0.68 * 260 + (x - 260) * 0.73
    else:
        y = 0.68 * 260 + (600 - 260) * 0.73 + (x - 600) * 0.98
    print("本月电费为" + str(y) + "元")
```

运行结果:

请输入每月用电量: -1 输入错误!	请输入每月用电量: 200 本月电费为 136.0 元	请输入每月用电量: 400 本月电费为 279.0 元	请输入每月用电量: 601 本月电费为 425.98 元

【实例 3.6】 根据空气质量指数进行生活建议。

空气质量指数(Air Quality Index,AQI)是根据空气中的各种成分占比,将监测的空气浓度简化为单一的概念性数值形式,它将空气污染程度和空气质量状况分级表示,适合于表

示城市的短期空气质量状况和变化趋势。具体数值及等级如表 3.1 所示。

表 3.1　空气质量指数、对应等级及相关建议

AQI 数值	等　　级	生 活 建 议
0～50	1 级，优	空气清新，参加户外活动
51～100	2 级，良	可以正常进行户外活动
101～150	3 级，轻度污染	敏感人群减少体力消耗大的户外活动
151～200	4 级，中度污染	对敏感人群影响较大，减少户外活动
201～300	5 级，重度污染	所有人适当减少户外活动
＞300	6 级，严重污染	尽量不要留在户外

源代码：

```
x = int(input("请输入 AQI 数值: "))
if x < 0:
    print("输入错误!")
else:
    if x <= 50:
        s = "1 级,优,空气清新,参加户外活动."
    elif x <= 100:
        s = "2 级,良,可以正常进行户外活动."
    elif x <= 150:
        s = "3 级,轻度污染,敏感人群减少体力消耗大的户外活动."
    elif x <= 200:
        s = "4 级,中度污染,对敏感人群影响较大,减少户外活动."
    elif x <= 300:
        s = "5 级,重度污染,所有人适当减少户外活动."
    else:
        s = "6 级,严重污染,尽量不要留在户外."
    print("空气质量为" + s)
```

运行结果：

```
请输入 AQI 数值: 200
空气质量为 4 级,中度污染,对敏感人群影响较大,减少户外活动.
```

【实例 3.7】 新冠肺炎疫情风险等级划分。

根据新冠肺炎疫情实际情况和发展态势，综合考虑新增和累计确诊病例数等因素，以县市区为单位，划分为低风险区、中风险区、高风险区。

低风险区：无确诊病例或连续 14 天无新增确诊病例。

中风险区：14 天内有新增确诊病例，累计确诊病例不超过 50 例，或累计确诊病例超过 50 例，14 天内未发生聚集性疫情。

高风险区：累计病例超过 50 例，14 天内有聚集性疫情发生。

源代码：

```
x = int(input("请输入新增确认病例数: "))
y = int(input("请输入累计确诊病例数: "))
z = int(input("请输入聚集性疫情发生天数: "))
if x >= 14 or y == 0:
    s = "低"
elif (x > 0 and z <= 14 and y <= 50) or (y > 50 and z < 14):
    s = "中"
else:
    s = "高"
print("此地区为" + s + "风险区")
```

运行结果：

请输入新增确认病例数: 0 请输入累计确诊病例数: 0 请输入聚集性疫情发生天数: 0 此地区为低风险区	请输入新增确认病例数: 5 请输入累计确诊病例数: 10 请输入聚集性疫情发生天数: 2 此地区为中风险区	请输入新增确认病例数: 10 请输入累计确诊病例数: 100 请输入聚集性疫情发生天数: 20 此地区为高风险区

3.2.4 分支结构的嵌套

分支结构的嵌套是指实际开发过程中,在一个分支结构中嵌套另一个分支结构。基本语法格式如下：

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

从语法角度讲,选择结构可以有多种嵌套形式。程序员可以根据需要选择合适的嵌套结构,但一定要注意控制不同级别代码块的缩进量,因为缩进量决定代码块的从属关系。

【实例 3.8】 分段函数求值。

$$f(x)=\begin{cases}x, & x\leqslant 1\\ 2x-1, & 1<x<10\\ 3x-11, & x\geqslant 10\end{cases}$$

源代码：

```
x = float(input("input x: "))
if x <= 1:
    y = x
else:
    if x < 10:
```

```
        y = 2 * x - 1
    else:
        y = 3 * x - 11
print("x = " + str(x) + ",f(x) = " + str(y))
```

运行结果：

input x: 0.5 x = 0.5,f(x) = 0.5	input x: 5 x = 5.0,f(x) = 9.0	input x: 20 x = 20.0,f(x) = 49.0

当然，实例 3.8 也可以不使用嵌套语句实现，而使用多分支结构实现。

```
x = float(input("input x: "))
if x <= 1:
    y = x
elif x < 10:
    y = 2 * x - 1
else:
    y = 3 * x - 11
print("x = " + str(x) + ",f(x) = " + str(y))
```

很明显，多分支结构比分支嵌套可读性更强，Python 之禅中有一句话："Flat is better than nested."，扁平化总比嵌套好，所以能扁平化时尽量不要用嵌套。

3.3 循环结构

如果需要重复执行某条或者某些指令，例如"中国诗词大赛"中的"飞花令"，选手要根据给定的关键字，在给定的时间内轮流背诵含有关键字的诗句，直至时间结束。重复执行类似动作就是循环结构。Python 提供两种循环结构语句：while 循环和 for…in 循环。前者根据条件返回值的情况决定是否执行循环体，后者采用遍历的形式指定循环范围。要更加灵活地操纵循环语句的流向，还需要使用 break、continue 和 pass 等语句。

3.3.1 while 循环


视频讲解

while 循环也称为无限循环，是由条件控制的循环运行方式，一般用于循环次数难以提前确定的情况。while 循环的语法格式为：

```
while 条件表达式:
    循环体
[else:
    语句块]
```

其中，"条件表达式"可以是任何非空或者非零的表达式，"循环体"可以是单条语句或语句块，方括号内的 else 子句可以省略。

流程图如图 3.5 所示。

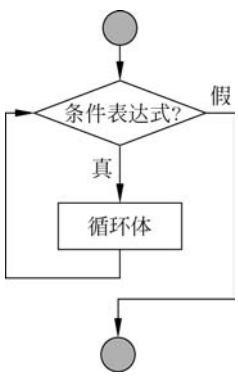


图 3.5 while 循环结构流程图

执行过程为先判断条件表达式,如果结果为真,则执行循环体,继续进行条件判断;否则循环结束。

【实例 3.9】 求 $\sum_{i=1}^{100} i$。

算法分析:设计循环算法需要考虑循环三要素:循环初值、结束条件以及增量(步长)。本例中,循环变量为 i,初值为 1,结束条件或者终值为 100,步长为 1。另外还需要一个变量存储累加和,其初值为 0。对应的结构流程图如图 3.6 所示。

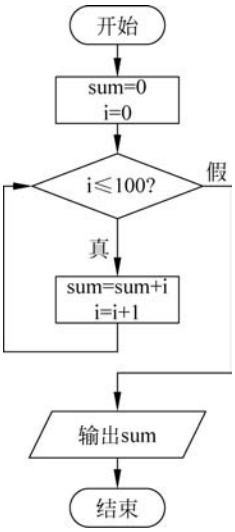


图 3.6 求累加和结构流程图

源代码:

```
sum, i = 0, 0
while i <= 100:
    sum += i
    i = i + 1
print("sum = " + str(sum))
```

运行结果:

```
sum = 5050
```

拓展:$1+3+\cdots+99$、$\prod_{i=1}^{100} i$、$\sum_{i=1}^{n} i$、$\sum_{i=m}^{n} i$ 等类似累加和或者累乘积的计算。

【实例 3.10】 求若干个学生某门课程的平均成绩。

算法分析:循环变量为学生人数,初值为 0,终值为学生个数 n,步长为 1。循环体累加每个学生的成绩,循环结束后求成绩的平均值。其结构流程图如图 3.7 所示。

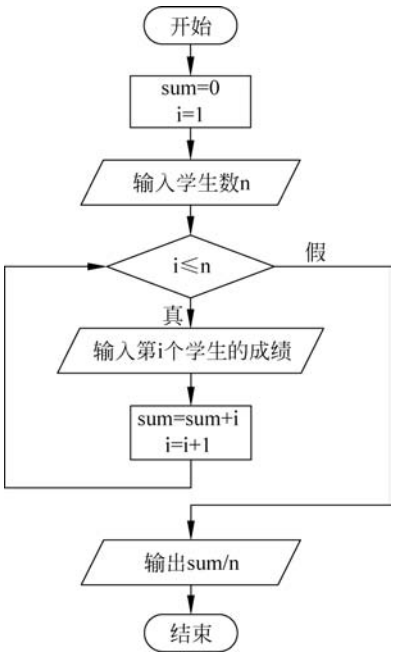


图 3.7　求平均成绩结构流程图

源代码：

```
sum,i = 0,1
n = int(input("请输入学生人数: "))
while i <= n:
    score = float(input("NO " + str(i) + ": "))
    sum += score
    i = i + 1
print("平均成绩为 " + str(sum / n))
```

运行结果：

```
请输入学生人数: 5
NO 1: 100
NO 2: 85.5
NO 3: 98.7
NO 4: 95
NO 5: 65
平均成绩为 88.84
```

循环结构中也可以使用 else 子句，用于表示不满足循环条件时程序的执行流程。

【实例 3.11】 循环结构使用 else 子句。

源代码：

```
count = 0
while count < 5:
    print(str(count) + " is less than 5.")
```

```
    count = count + 1
else:
    print(str(count) + " is not less than 5.")
```


运行结果：

```
0 is less than 5.
1 is less than 5.
2 is less than 5.
3 is less than 5.
4 is less than 5.
5 is not less than 5.
```

可见，当满足循环条件“count＜5”时，执行循环体，当不满足循环条件时，执行“print(str(count)＋" is not less than 5.")”。


视频讲解

3.3.2 for…in 循环

Python 提供的另一种循环结构是 for…in 循环。Python 提供的 for 循环语句与 Java、C++等编程语言提供的 for 语句不同，更像是 shell 或是脚本语言中的 for 循环，可以遍历如列表、元组、字符串等序列成员，也可以用在列表解析和生成器表达式中。

1. 使用序列项迭代序列对象

通过 for…in 循环可以迭代序列对象的所有成员，并在迭代结束后，自动结束循环，其语法如下：

```
for iterating_var in list:
    循环体
```

其中，iterating_var 为迭代变量，list 为序列(字符串、列表、元组、字典、集合)。执行时，迭代变量依次取序列中元素的值，直至取完，循环退出。

对应的结构流程图如图 3.8 所示。

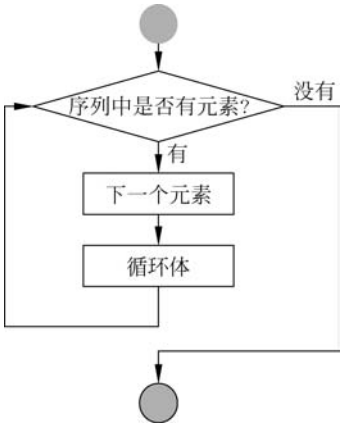


图 3.8　迭代序列 for…in 循环结构流程图

【实例 3.12】 统计字符串中各类字符的个数。

算法分析：使用迭代变量遍历序列(字符串)中的每一个元素，分别判断所属类型，并将对应个数加 1，直至遍历结束。结构流程图如图 3.9 所示。

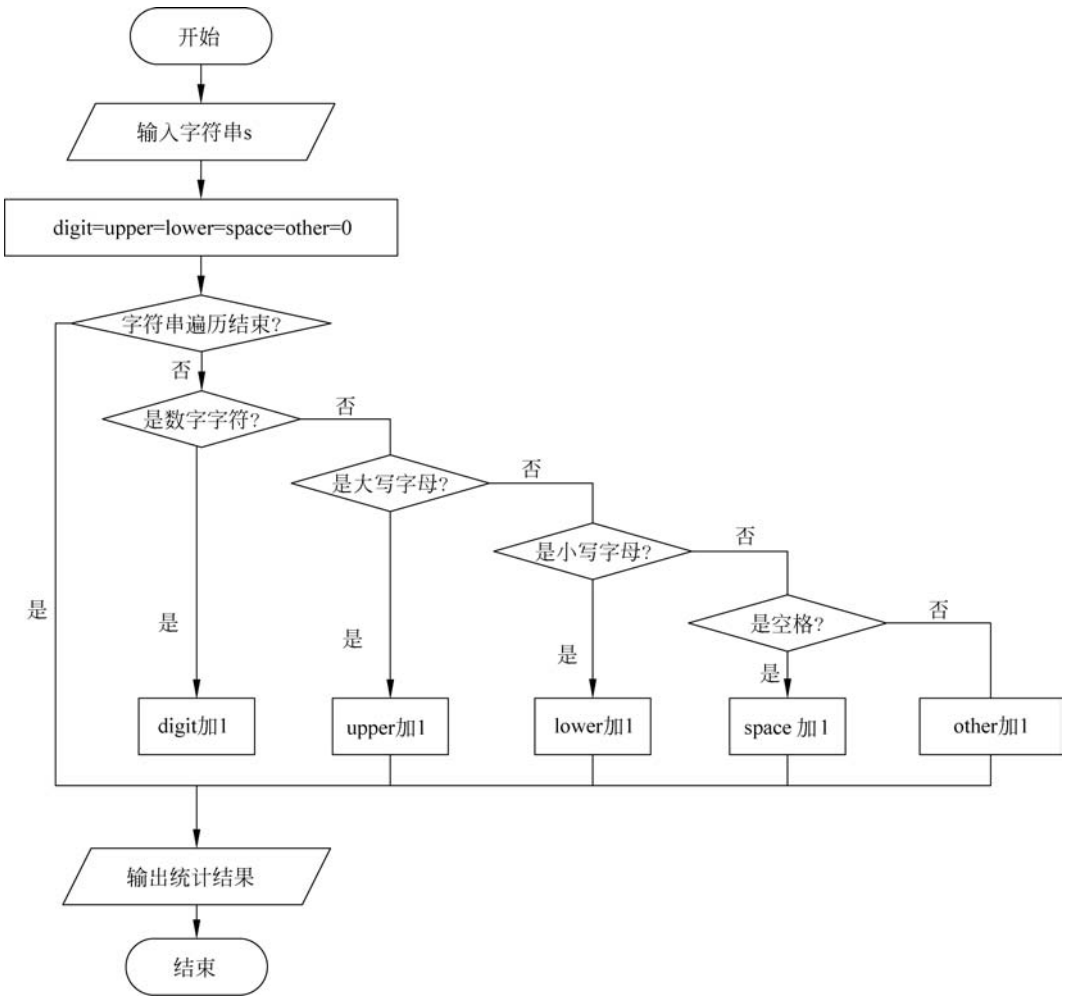


图 3.9 统计字符串中各类字符个数的结构流程图

源代码：

```
1  s = input("请输入一个字符串：")
2  digit,upper,lower,space,other = 0,0,0,0,0   #数字字符,大写字符,小写字符,空格字符,
                                               #其他字符的个数
3  for i in s:
4      if i >= '0' and i <= '9':               #判断 i 是否为数字字符
5          digit = digit + 1
6      elif i >= 'A' and i <= 'Z':             #判断 i 是否为大写字母
7          upper = upper + 1
8      elif i >= 'a' and i < 'z':              #判断 i 是否为小写字母
```

```
        lower = lower + 1
    elif i == ' ':   #判断 i 是否为空格字符
        space = space + 1
    else:   #其他字符
        other = other + 1
print("数字字符: %d\n 大写字母: %d\n 小写字母: %d\n 空格字符: %d\n 其他字符: %d
\n" %(digit,upper,lower,space,other))
```

运行结果:

```
请输入一个字符串: Life is short, we need Python!
数字字符: 0
大写字母: 2
小写字母: 21
空格字符: 5
其他字符: 2
```

其中,实现字符分类的循环体也可用内置函数代替,如:

```
if i.isdigit( ):          #判断 i 是否为数字字符
    digit = digit + 1
elif i.isupper( ):        #判断 i 是否为大写字母
    upper = upper + 1
elif i.islower( ):        #判断 i 是否为小写字母
    lower = lower + 1
elif i.isspace( ):        #判断 i 是否为空格字符
    space = space + 1
else:
    other = other + 1
```

内置函数 i.isdigit()、i.isupper()、i.islower()、i.isspace()分别用来判断 i 是否为数字字符、大写字母、小写字母和空格字符。

2. 使用序列索引迭代序列对象

在 for…in 循环结构中,也可以使用序列索引来遍历列表,语法如下:

```
for index in range(len(list)):
    循环体
```

其中,index 为序列的索引项,内置函数 range()为计数函数,len()获取序列长度。

【实例 3.13】 统计字符串中各类字符的个数(range 版)。

```
s = input("请输入一个字符串: ")
digit = upper = lower = space = other = 0   #数字字符、大写字母、小写字母、空格字符
                                            #和其他字符的个数
for i in range(len(s)):
    if s[i].isdigit( ):                     #判断 i 是否为数字字符
        digit = digit + 1
```

```
    elif s[i].isupper( ):   #判断 i 是否为大写字母
        upper = upper + 1
    elif s[i].islower( ):   #判断 i 是否为小写字母
        lower = lower + 1
    elif s[i].isspace( ):   #判断 i 是否为空格字符
        space = space + 1
    else:
        other = other + 1
print("数字字符: %d\n 大写字母: %d\n 小写字母: %d\n 空格字符: %d\n 其他字符: %d
\n" % (digit, upper, lower, space, other))
```

运行结果：

```
请输入一个字符串: I am a student, I am 20 years old!
数字字符: 2
大写字母: 2
小写字母: 20
空格字符: 8
其他字符: 2
```

使用 range()函数可以得到用来迭代的索引列表，使用索引下标“[]”可以方便快捷地访问序列对象。另外，还可以使用 range()函数实现类似 Java、C++等传统编程语言的 for 循环结构，即从循环三要素角度出发设计循环结构，语法格式为：

```
range([start,] end[, step = 1])
```

其中，range()函数会返回一个整数序列，可选项 start 为序列初值（循环变量初值），end 为序列终止值（循环变量终值，且不含 end 本身），可选项 step 为步长或增量，默认为 1。

【实例 3.14】 求 $\sum_{i=1}^{100} i$（range 版）。

```
sum = 0
for i in range(1, 101, 1):
    sum = sum + i
print("sum = " + str(sum))
```

运行结果：

```
sum = 5050
```

显然，此时的 for…in 循环与 while 循环完全等价。

3. 使用枚举函数迭代序列对象

Python 内置函数 enumerate()用于将一个可遍历的数据对象（列表、元组或者字符串）组合成一个索引序列，同时列出数据和下标，一般用于 for…in 循环中。语法格式为：

```
for index, iterating_var in enumerate(list, start_index = 0):
    循环体
```

其中,index 返回索引计数,iterating_var 为与索引计数相对应的索引对象成员,list 为待遍历的序列对象,start_index 为返回的起始索引计数,默认值为 0。

【实例 3.15】 输出学生花名册。

```
1  name_list = ["李白", "孟浩然", "王维", "李绅"]  #name_list 的数据类型为列表 list
2  for index, name in enumerate(name_list):
3      print(index, name)
```

运行结果:

```
0 李白
1 孟浩然
2 王维
3 李绅
```

视频讲解

3.3.3 循环嵌套

允许在一个循环结构中嵌入另一个循环结构,称为循环嵌套。在 Python 中,for…in 循环结构和 while 循环结构都可以进行循环嵌套。如:

```
while condition_expression 1:
    for index in range(len(list)):
        循环体
```

不仅 for 循环结构可以嵌入到 while 循环结构中,while 循环结构也可以嵌入到 for…in 循环结构中,同样也可以根据自身需要任意嵌套。

【实例 3.16】 输出九九乘法表(下三角形)。

算法分析:从结构看,九九乘法表是二维形式,单重循环无法实现。从内容看,第一个乘数每行一致,第二个乘数同行每列依次加 1,故使用两重循环。外循环控制第一个乘数(迭代变量为 1～9),内循环控制第二个乘数。因为要求按照下三角形输出,所以内循环迭代变量只能为 1～i。结构流程图如图 3.10 所示。

源代码:

```
1  for i in range(1, 10):
2      for j in range(1, i + 1):
3          print(str(i) + "*" + str(j) + "=" + str(i * j), end = " ")
4      print()      #每行末尾换行
```

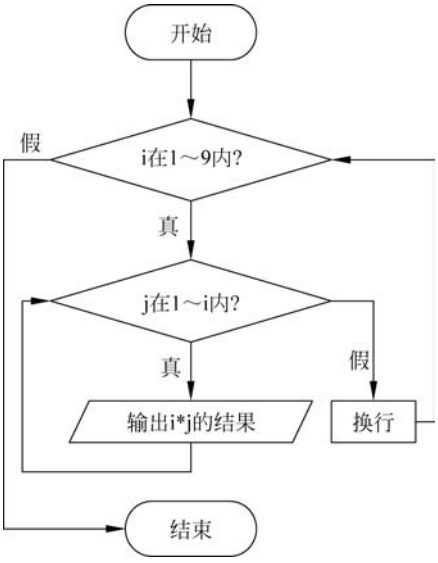


图 3.10　九九乘法表结构流程图

运行结果：

```
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
```

拓展：输出上三角九九乘法表、钻石形等图形。

【实例 3.17】 求 1!＋2!＋…＋20!。

算法分析：求 n! 需要用循环结构实现，累加和也需要用循环结构实现，故采用双重循环。外循环计算累加和，内循环求 n!。结构流程图如图 3.11 所示。

源代码：

```
sum = 0                          # 累加和
for i in range(1, 21):           # 外循环,用于累加和
    fact = 1                     # 存放 n!
    for j in range(1, i + 1):    # 内循环,用于计算 i!
        fact = fact * j
    sum = sum + fact
print("sum = " + str(sum))
```

运行结果：

```
sum = 2561327494111820313
```

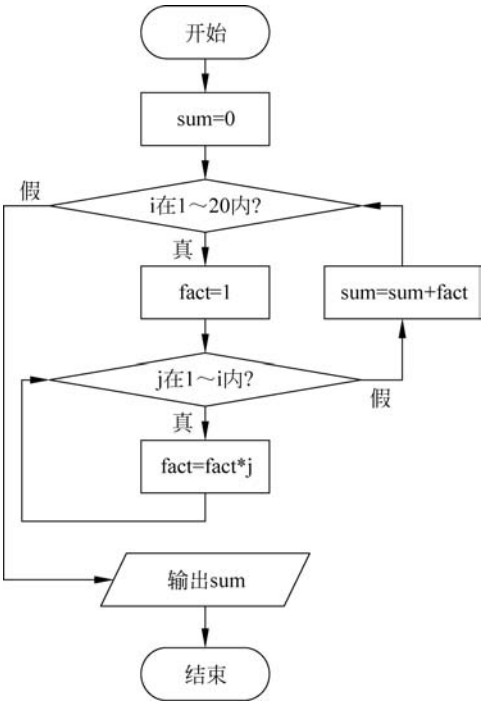


图 3.11　阶乘和结构流程图

观察运行过程可以发现,双重循环计算累加和时,每次都是从 1 开始计算 n!。事实上 n!=(n−1)! * n,故可以使用单重循环实现。优化后的代码:

```
sum = 0                 #累加和
fact = 1                #存放 n!
for i in range(1, 21):
    fact = fact * i     #直接使用 n!= (n - 1)! * n 来计算 n!
    sum = sum + fact
print("sum = " + str(sum))
```

循环结构中可以嵌套另一个循环结构,也可以嵌套选择结构,反之亦然。

【实例 3.18】 列出 1～200 的所有素数,要求每行输出 10 个数(标志变量版)。

算法分析:素数是一个只能被 1 和本身整除的自然数。判断 n 是否为素数的方法为依次除以 2～$\sqrt{n}$,如果能整除则不是素数。这个过程需要使用单重循环嵌套选择结构实现。而列出 1～200 的所有素数也需要用循环实现,故使用双重循环完成。流程图如图 3.12 所示。

源代码:

```
import math                 #math 模型库,sqrt()函数需要使用
count = 0                   #累计素数个数
for n in range(2, 201):     #外循环遍历 2～200
    i,flag = 2,True
```

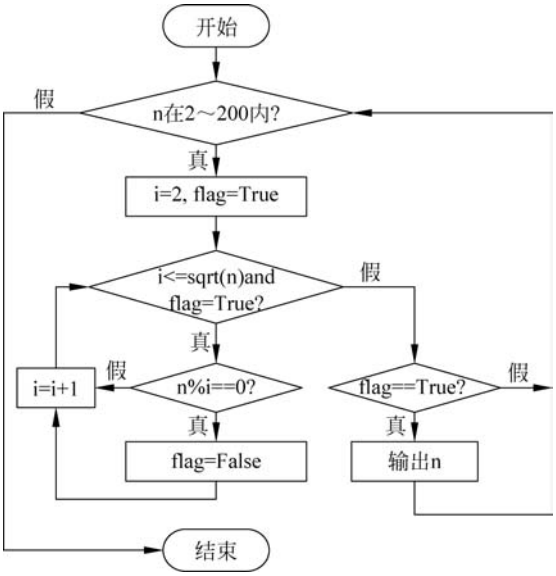


图 3.12　实例 3.18 结构流程图

```
5       while i <= math.sqrt(n) and flag:    # 内循环对每一个数进行是否为素数的判断
6           if n % i == 0:                   # 如果能够整除,则不是素数,flag 置为 False
7               flag = False
8           i = i + 1
9       if flag:                             # 如果是素数,则输出
10          count = count + 1
11          print(n, end = "\t")
12          if count % 10 == 0:              # 控制每行 10 个
13              print()
```

运行结果：

```
2   3   5   7   11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97  101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199
```

说明：本实例中外循环使用 for…in 结构，内循环使用 while 结构，并在内循环中嵌入分支结构进行整除判断。引入标志变量 flag 来标志 n 是否为素数(默认值为 True)，一旦判断能够整除(即不是素数)，则修改 flag 值为 False，内循环条件执行为否，内循环退出。引入计数变量 count 来记录每行输出个数。

3.3.4　break 和 continue

在循环结构中，大多数情况下当循环条件满足时，循环体会一直被执行，直到循环条件不满足。但有时需要在某种条件下使循环提前结束，实现方法有两种：一是使用标志变量，如实例 3.18 中的 flag，通过 flag 值的变化，在循环未正常结束时提前退出循环；二是使用

break 语句来实现。

break 语句可以终止当前的循环。使用时一般与 if 语句搭配使用,表示在某种条件下提前结束循环。

【实例 3.19】 break 示例。

```
n = int(input("n: "))
for i in range(1, 11):
    if i == n:
        break
    print(i, end = ",")
```

运行结果:

n: 5 1,2,3,4,	n: 20 1,2,3,4,5,6,7,8,9,10,

从运行结果可以看出,当 i 的迭代次数小于 10 时,循环会提前结束。

注意:使用嵌套循环时,break 语句只跳出最内层的循环。

【实例 3.20】 列出 1～200 的所有素数,要求每行输出 10 个数(break 版)。

```
import math                          #math 模型库,sqrt()函数需要使用
count = 0
for n in range(2, 201):              #外循环遍历数据
    i = 2
    while i <= math.sqrt(n):         #内循环判断是否为素数
        if n % i == 0:               #能整除则不是素数,判断结束
            break
        i = i + 1
    if i > math.sqrt(n):             #是素数
        count = count + 1
        print(n, end = "\t")
        if count % 10 == 0:          #控制每行 10 个
            print()
```

说明:内循环结束有两种途径,其中一种是 n 是素数正常结束,即所有的 n%i! =0,此时 i>sqrt(n);另一种是 n%==0 即 n 不是素数,提前结束循环。所以 break 版与标志变量版在输出素数时条件正好相反。

有时只是在一定条件下不想执行本次循环体,而继续执行下一轮循环,此时需要使用另一种语句——continue。

continue 是另一种提前结束循环语句,与 break 不同,continue 只结束本次循环,继续后续操作。

【实例 3.21】 continue 示例。

```
n = int(input("n: "))
for i in range(1, 11):
```

```
3       if i == n:
4             continue
5       print(i, end = ",")
```

运行结果：

n: 5 1,2,3,4,6,7,8,9,10,	n: 20 1,2,3,4,5,6,7,8,9,10,

从运行结果可以看出，当 n＜10 时，只是不执行本次循环而继续执行后续循环。

【实例 3.22】 break 与 continue 的区别示例。

```
1   import random
2   n = random.randint(0, 10)
3   print("您选择的是", n)
4   for i in range(1,11):
5       if i == n:
6             print(i, "结束了.")
7             break
8       if i % 3 != 0:
9             print(i, "继续!")
10            continue
11      print('I love Python!')
```

运行结果：

您选择的是 4 1 继续! 2 继续! I love Python! 4 结束了.	 # 满足 i % 3! = 0，执行 continue，进行下一次循环 # 两个判断条件都不满足 # 满足 i = = n，执行 break，退出循环

此程序段的执行过程也可以用图 3.13 表示。

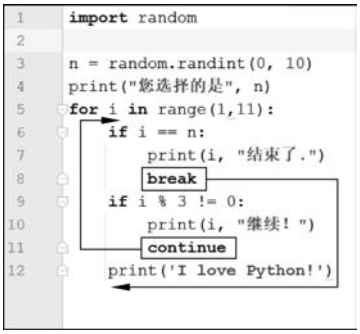


图 3.13　break 与 continue 的区别示意图

3.3.5 穷举与迭代

1. 穷举

穷举法也称为枚举法或者列举法,是计算机求解问题时常用的算法,用来解决那些通过公式推导、规则演绎等方法不能解决的问题。其基本思想是:不重复、不遗漏地列举出所有可能的情况,以便从中寻找满足条件的结果。采用穷举法解决实际问题时,主要使用循环结构嵌套选择结构实现——循环结构用于列举所有可能的情况,而选择结构用于判断当前条件是否为所求解,其一般框架为:

视频讲解

```
for 循环变量 x 的所有可能的值:
    if  x 满足指定条件:
        x 即为所求解
```

【实例 3.23】 (百钱买百鸡)我国古代数学家张丘建在《算经》一书中提出的数学问题:鸡翁一值钱五,鸡母一值钱三,鸡雏三值钱一。百钱买百鸡,问鸡翁、鸡母、鸡雏各几何?

算法分析:假设有 x 只鸡翁(即公鸡),y 只鸡母(即母鸡),则鸡雏(小鸡)有(100－x－y)只。根据题意,一只公鸡需要五钱,一只母鸡需要三钱,一只小鸡需要一钱,故可以列方程 $5*x+3*y+\frac{1}{3}*(100-x-y)=100$。这是一个不定式方程,不能利用普通的公式推导得出结论,最常用的解法就是枚举,把所有可能的情况一一列举出来。流程图如图 3.14 所示。

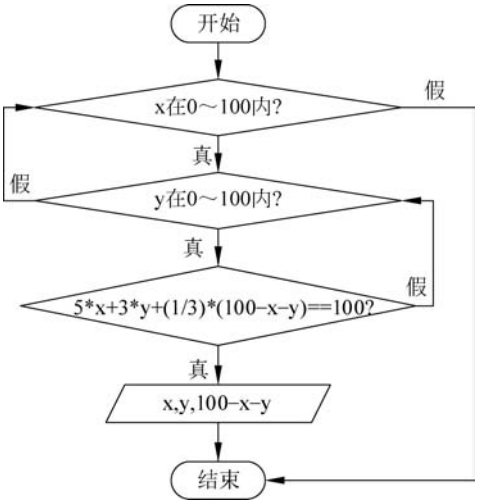


图 3.14　百钱买百鸡结构流程图

源代码:

```
1  for x in range(101):
2      for y in range(101):
3          if 5 * x + 3 * y + (1/3) * (100 - x - y) == 100:
4              print(x, y, (100 - x - y))
```

运行结果：

```
0 25 75
4 18 78
8 11 81
12 4 84
```

当然，此程序可以优化。由题意可以值，公鸡最多 20 只，母鸡最多 33 只，所以程序可以优化为：

```
for x in range(21):
    for y in range(34):
        if 5 * x + 3 * y + (1/3) * (100 - x - y) == 100:
            print(x, y, (100 - x - y))
```

2. 迭代

迭代是另一种常用的循环算法，它利用计算机运行速度快，适合做重复动作的特点，让计算机对一组语句进行重复操作，并且后一次的操作数直接依赖于前一次的执行结果。用迭代法求解实际问题时，需要考虑两方面的问题。

（1）确定迭代变量：由旧值直接或者间接递推而来的变量就是迭代变量。

（2）建立迭代关系式：迭代关系式即“循环不变式”，是一个直接或者间接地不断由旧值递推出新值的表达式。

【实例 3.24】（斐波那契数列）意大利著名的数学家斐波那契在《计算之书》中提出了一个有趣的兔子问题：一对成年兔子每个月恰好生下一对小兔子（一雌一雄）。在年初时，只有一对小兔子。在第一个月结束时，它们成长为成年兔子，并且第二个月结束时，这对成年兔子将生下一对小兔子。这种成长与繁殖的过程会一直持续下去，并假设生下的小兔子都不会死，那么一年之后共可有多少对小兔子？（为清楚地描述数列，输出前 20 项）

算法分析：斐波那契数列（Fibonacci sequence）又称黄金分割数列、兔子数列。从问题的描述可以发现，年初时只有一对小兔子，第二个月这一对小兔子长成中兔子（兔子总数为 1 对）；第三个月中兔子长成大兔子并生下一对小兔子（兔子总数为 2 对）；第四个月小兔子长成中兔子，大兔子再生下一对小兔子（兔子总数为 1＋1＋1＝3 对）……可以用表 3.2 描述这个过程。

表 3.2　斐波那契数列的变化过程

月数	小兔子对数	中兔子对数	大兔子对数	兔子总数
1	1	0	0	1
2	0	1	1	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
…	…	…	…	…

可以看出,斐波那契数列为 1,1,2,3,5,8,…,从第三个数开始,后一个数是前两个数的和。可以使用迭代法求解,迭代表达式为:

$$F(n)=\begin{cases}1, & n=1,2\\ F(n-1)+F(n-2), & n>2\end{cases}$$

其中 n 为 1 或者 2 时为迭代出口。

流程图如图 3.15 所示。

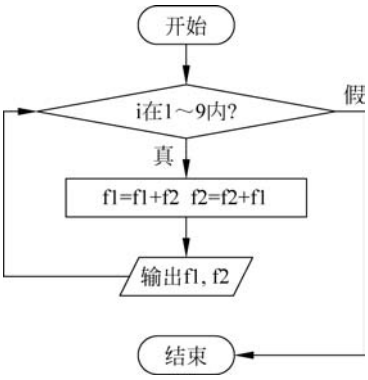


图 3.15 实例 3.24 结构流程图

源代码:

```
f1 = f2 = 1                    #迭代变量的初值
print(f1, f2, end = " ")       #先输出前两个数
for f_index in range(1, 10):#每次输出两个数,一共输出 10 个数
    f1 = f1 + f2               #迭代表达式,后一个数为前两个数的和
    f2 = f2 + f1
    print(f1, f2, end = " ")
```

运行结果:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

视频讲解

3.4 流程控制综合例子

【实例 3.25】 设计小型的加减乘除测试小程序(由系统随机出 10 个加减乘除运算题目,运算数和运算符都由系统随机给出,系统自动给出答题结果和运算时间)。

算法分析:此实例需要循环与多分支结构嵌套,循环负责控制题目个数,分支结构检测加减乘除并进行相应计算。为简单起见,减法和除法都要求第一个操作数大于第二个操作数,并且除数不能为 0。流程图如图 3.16 所示。

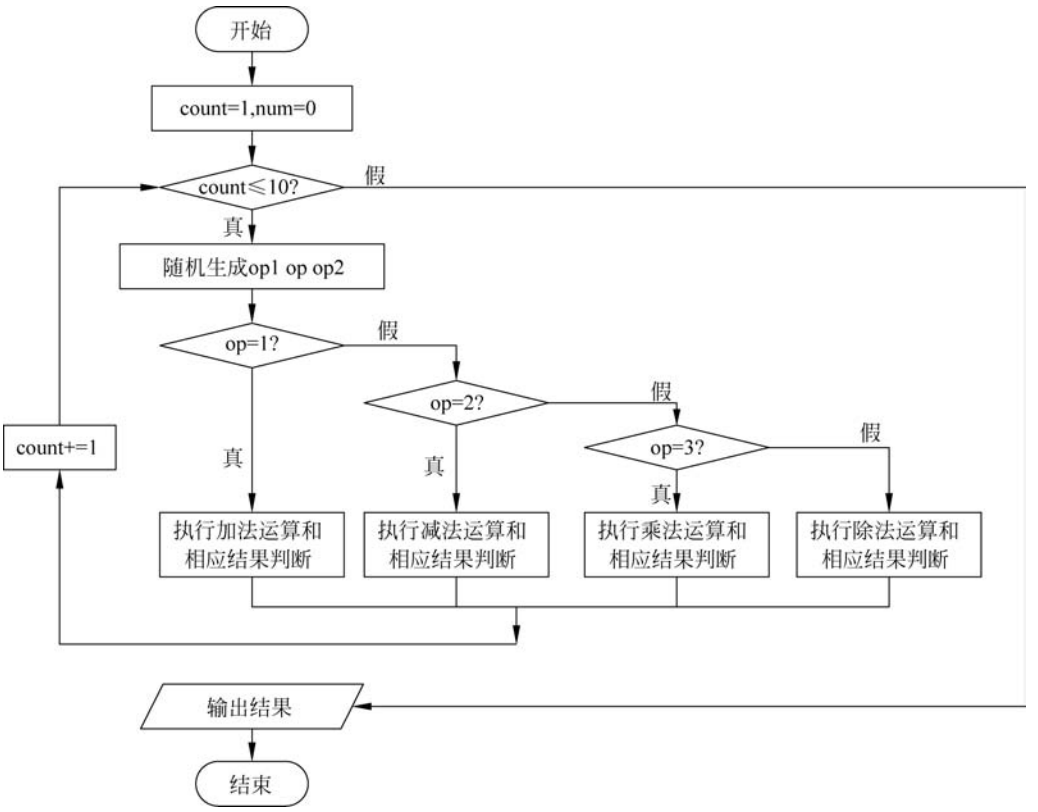


图 3.16　实例 3.25 结构流程图

源程序如下(为简单起见,将操作数规定为不大于 20 的数):

```
import random                          #随机函数库
import time                            #时间库
count,num = 1,0                        #分别表示题目个数,答对的题目个数
begin_time = time.time()               #取当前系统时间
while count < 11:
    op1 = random.randint(1, 20)   #随机生成一个不大于 20 的数作为第一个操作数
    op2 = random.randint(1, 20)   #随机生成一个不大于 20 的数作为第二个操作数
    op = random.randint(1, 4)     #随机生成一个不大于 4 的数作为操作符
    if op == 1:                        #加法运算
        print(str(op1) + "+" + str(op2) + "=", end="")
        result = int(input( ))
        if result == op1 + op2:
            print("正确!")
            num = num + 1              #答对题目个数加 1
        else:
            print("错误!")
    elif op == 2:                      #减法运算
        if op1 < op2:                  #保证被减数>减数
            op1,op2 = op2,op1
        print(str(op1) + "-" + str(op2) + "=", end="")
        result = int(input( ))
```

```
        if result == op1 - op2:
            print("正确!")
            num = num + 1          #答对题目个数加 1
        else:
            print("错误!")
    elif op == 3:                  #乘法运算
        print(str(op1) + " * " + str(op2) + " = ", end = "")
        result = int(input( ))
        if result == op1 * op2:
            print("正确!")
            num = num + 1          #答对题目个数加 1
        else:
            print("错误!")
    else:
        while op2 == 0:            #除法中除数不能为 0
            op2 = random.randint(1, 100)
        if op1 < op2:              #保证被除数>除数
            op1,op2 = op2,  op1
        print(str(op1) + "/" + str(op2) + " = ", end = "")
        result = int(input( ))
        if result == op1 // op2:  #为简单起见,采用整除
            print("正确!")
            num = num + 1          #答对题目个数加 1
        else:
            print("错误!")
    count = count + 1
end_time = time.time()  #获取系统当前时间
print("答对" + str(num) + "道题目, 得分" + str(10 * num))
print("用时为" + str(end_time - begin_time))    #两次时间差为运行时间
```

运行结果:

```
15/8 = 1
正确!
20 - 11 = 9
正确!
14 - 3 = 11
正确!
13/3 = 4
正确!
9 * 19 = 171
正确!
8/1 = 8
正确!
15/13 = 1
正确!
18 * 1 = 18
正确!
```

```
19 + 6 = 25
正确!
11/9 = 1
正确!
答对 10 道题目, 得分 100
用时为 28.110098600387573
```

【实例 3.26】 模拟“剪刀石头布”五局三胜猜拳游戏：选手和计算机轮流猜拳 5 次，3 次胜利才算赢。

算法分析：选手输入选项(“剪刀”“石头”“布”)，计算机随机给出选项，按照游戏规则——“布”>“石头”，“石头”>“剪刀”，“剪刀”>“布”进行评判和计数，一旦一方满足五局三胜则游戏结束。流程图如图 3.17 所示。

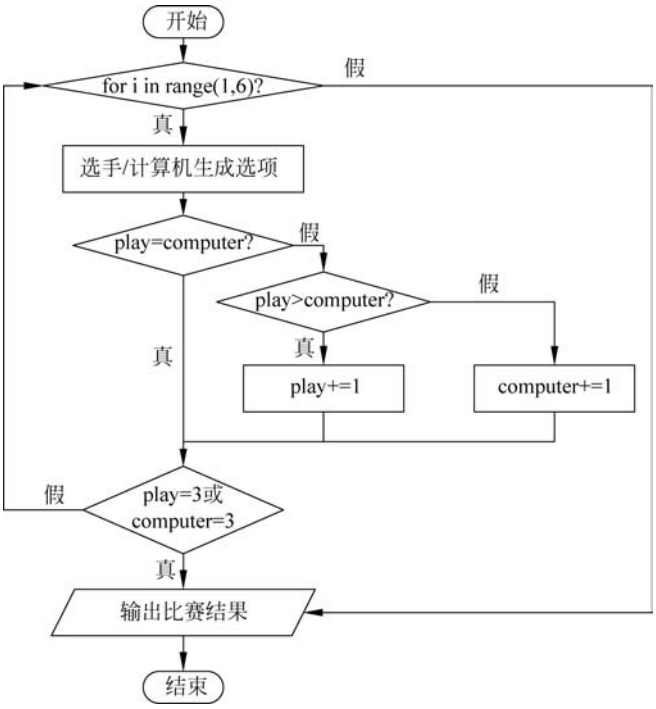


图 3.17　五局三胜猜拳游戏结构流程图

源程序如下：

```
1  import random                  #随机函数库
2  count1 = count2 = 0            #count1 和 count2 分别表示选手和计算机获胜的次数
3  for i in range(1, 6):          #最多进行五局
4      print("第 %d 局: " % i)
5      play = input("选手: ") #选手输入选项并进行分类"剪刀"-->1,"石头"-->2,"布"
                              #-->3
6      play = 1 if play == "剪刀" else (2 if play == "石头" else 3)  #三目运算符嵌套
```

```
    computer = random.randint(1, 3)  #计算机随机生成选项:1.-剪刀,2.-石头,3.-布
    #三目运算符嵌套输出计算机选项,可读性较差
    print("计算机:剪刀") if computer == 1 else (print("计算机:石头") if computer ==
2 else print("计算机:布"))
    if play == computer:              #进行判断并计数
        print("选项一样")
    elif (play == 1 and computer == 2) or (play == 2 and computer == 3) or ( play ==
3 and computer == 1):
                                      #"剪刀"<"石头","石头"<"布","布"<"剪刀"
        print("计算机赢")
        count2 += 1
    else:
        print("选手赢")
        count1 += 1
    if count1 == 3 or count2 == 3:    #三胜退出
        break
#输出最终结果
if count1 > count2:
    print("最终选手胜出")
elif count1 < count2:
    print("最终计算机胜出")
else:
    print("平局")
```

运行结果:

```
第 1 局:
选手:剪刀
计算机:布
选手赢
第 2 局:
选手:石头
计算机:布
计算机赢
第 3 局:
选手:剪刀
计算机:石头
计算机赢
第 4 局:
选手:石头
计算机:石头
选项一样
第 5 局:
选手:布
计算机:剪刀
计算机赢
最终计算机胜出
```

【实例 3.27】 用 1、3、5、8 这几个数字,能组成的互不相同且无重复数字的 3 位数各

是多少(每行输出 10 个数字)？总共有多少个？(蓝桥杯全国软件大赛青少年创意编程 Python 组)

算法分析：使用穷举法解决问题，循环结构列出所有可能，选择结构进行判断。

源程序：

```
data = [1, 3, 5, 8]     #列表存储数字,列表的内容在后续章节中详细介绍
count = 0               #满足条件的数的个数
for i in data:          #穷举法进行判断,循环结构穷举所有可能
    for j in data:
        for k in data:
            if i != j and j != k and k != i:   #选择结构进行判断是否满足给定条件
                count += 1                              #个数加 1
                print(100 * i + 10 * j + k, end = " ")  #输出数字
                if count % 10 == 0:                     #每行 10 个数字
                    print( )
print("\n一共" + str(count) + "个数字互不相同且无重复数字的 3 位数")
```

运行结果：

```
135 138 153 158 183 185 315 318 351 358
381 385 513 518 531 538 581 583 813 815
831 835 851 853
一共 24 个数字互不相同且无重复数字的 3 位数
```

视频讲解

3.5 光盘行动餐饮系统——勤俭节约

3.5.1 案例背景

2013 年 1 月 16 日，北京一个名为“IN_33”的团体发起“光盘行动”的公益活动。“光盘行动”的宗旨是：餐厅不多点、食堂不多打、厨房不多做，倡导厉行节约，反对铺张浪费，引导大家珍惜粮食，制止餐饮浪费行为。活动一经提出，就得到社会各方的大力支持。

在 2018 年世界粮食日，光盘打卡应用系统在清华大学正式发布。参与者用餐后手机拍照打卡，经由人工智能识别为“光盘”后可获得积分奖励，通过这种奖励的方式逐步引导人们养成节约的习惯，让中华民族勤俭节约的传统美德在新时代发扬光大。

2019 年，共青团中央《“美丽中国·青春行动”实施方案(2019—2023 年)》提出：“深化光盘行动，开展光盘打卡等线上网络公益活动”。4 月 22 日，共青团中央联合中华环保基金会和光盘打卡推出“2020 重启从光盘做起”光盘接力挑战赛。4 月 22 日—28 日，参与者用餐后通过光盘打卡小程序，人工智能识别光盘，成功光盘即可获得食光认证卡。除了高校的接力，活动也受到了环保、公益领域的关注与支持。

除此以外，文化和旅游部，商务部等发文号召餐饮行业“节俭消费提醒制度”。2020 年 8 月 11 日，习近平总书记对制止餐饮浪费行为作出重要指示。他指出，餐饮浪费现象，触目惊

心、令人痛心！“谁知盘中餐，粒粒皆辛苦。”尽管我国粮食生产连年丰收，对粮食安全还是始终要有危机意识，2020 年全球新冠肺炎疫情所带来的影响更是给我们敲响了警钟。

3.5.2 案例任务

“光盘行动餐饮系统”是一个具有点餐、进餐和结算功能的建议系统。在“点餐”功能模块中，根据人数 n 进行点餐，执行 n－1 的点餐规则。在“进餐”功能模块中，设置一个计数器进行模拟。在“结算”功能模块中，模拟 AI 机器人，通过扫描盘中剩余食品克数进行费用计算：如果总剩余量小于或等于 50g * n，则总餐费打八折；如果总剩余量小于或等于 100g * n，则总餐费打九折；如果总剩余量大于或等于 200 * n，则总餐费为应付餐费的 1.5 倍。

3.5.3 案例分析和实现

根据任务描述，程序可以分为如下几步：

(1) 输出相关提示信息，并且输入进餐人数 n。

(2) 根据进餐人数 n 进行点餐，餐品数量为 n－1。故在这个步骤，需要采用循环结构来完成点餐。同时累计原始应付餐费。

(3) 模拟进餐过程。

(4) 模拟机器人扫描剩余餐品，并且根据剩余餐品克数进行应付餐费的计算。需要采用多分支结构实现。

流程图如图 3.18 所示。

源程序如下：

```
print('欢迎光临 Python 餐馆,本餐馆实行"光盘行动",有几条规则请大家遵守: ')
print('1.根据人数进行点餐,餐品数量为人数 - 1.')
print('2.进餐时间为人数 * 15 分钟.')
print('3.根据剩余食品克数进行收费: ')
print( '' * 4 + '如果总剩余量小于或等于 50g * 人数,则总餐费打八折; \n' + '' * 4 +
'如果总剩余量小于或等于 100g * 人数,则总餐费打九折; \n' + '' * 4 + '如果总剩余量大
于或等于 200g * 人数,则总餐费为应付餐费的 1.5 倍.')
print('光盘行动,从我做起!')
n = int(input('请输入进餐人数: '))
print('请点餐' + str(n - 1) + "份,注意荤素搭配!")
food, money = '', 0
for i in range(n - 1):
    f = input("请输入您的第" + str(i + 1) + "份餐品: ")
    food += f
    food += ''
    m = int(input("请服务员报价: "))
    money += m
print("您一共点了" + str(n - 1) + "份餐品,分别为: " + food + ",当前总费用为: " +
str(money))
print("现在是您的用餐时间,时间为" + str(n * 15) + "分钟.")
print("=" * 40)
print("现在请 AI 机器人扫描您盘中剩余食物: ")
residus = float(input("请 AI 机器人报剩余餐品克数: "))
```

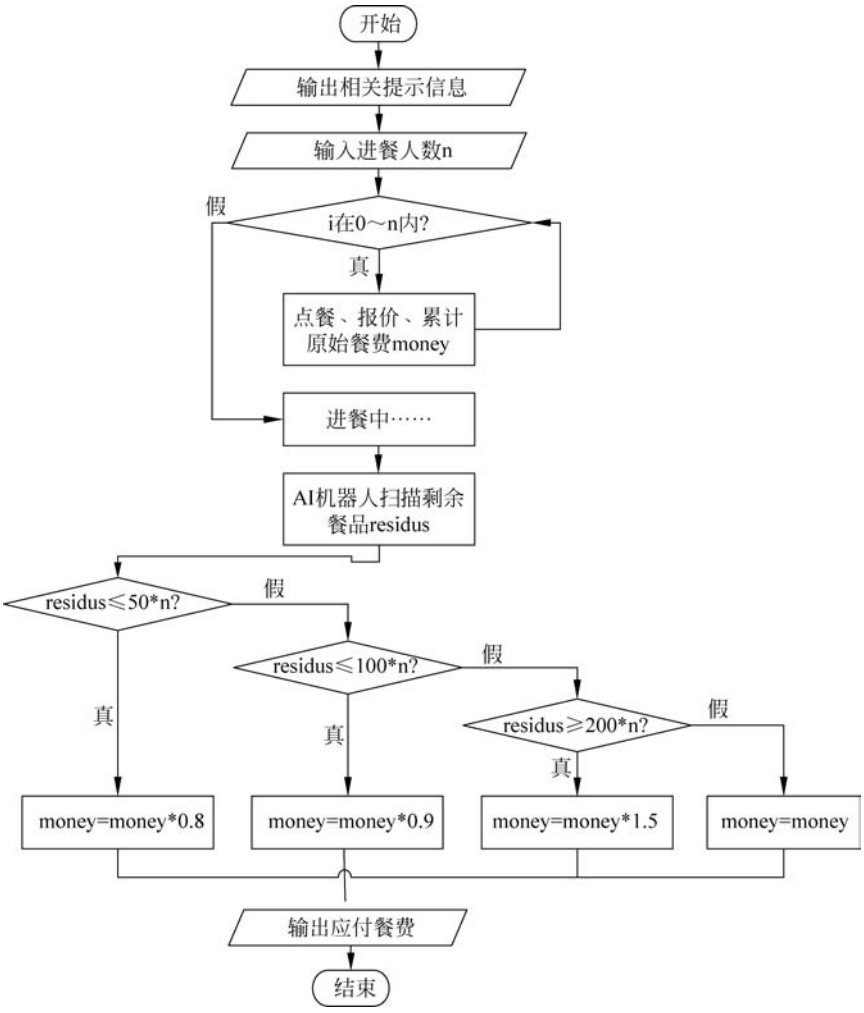


图 3.18 “光盘行动餐饮系统”流程图

```
if residus <= 50 * n:
    money = money * 0.8
    print("因您剩余餐品克数小于或等于" + str(50 * n) + ",为您打八折,您最终需要支付" + str(money) + "元.\n感谢您为光盘行动做的贡献,谢谢您!")
elif residus <= 100 * n:
    money = money * 0.9
    print("因您剩余餐品克数小于或等于" + str(100 * n) + ",为您打九折,您最终需要支付" + str(money) + "元.\n感谢您为光盘行动做的贡献,谢谢您!")
elif residus >= 200 * n:
    money = 1.5 * money
    print("因您剩余餐品克数大于或等于" + str(200 * n) + ",您需要支付1.5倍的餐费,您最终需要支付" + str(money) + "元.\n您可以选择打包带回家,希望您下次注意,谢谢配合!")
else:
    print("因您剩余餐品克不满足优惠条件,也没有超过惩罚标准,您最终需要支付" + str(money) + "元.\n感谢您为光盘行动做的贡献,谢谢您!")
```

运行结果：

```
欢迎光临 Python 餐馆,本餐馆实行"光盘行动",有几条规则请大家遵守:
1.根据人数进行点餐,餐品数量为人数 - 1.
2.进餐时间为人数 * 15 分钟.
3.根据剩余食品克数进行收费:
    如果总剩余量小于或等于 50g * 人数,则总餐费打八折;
    如果总剩余量小于或等于 100g * 人数,则总餐费打九折;
    如果总剩余量大于或等于 200g * 人数,则总餐费为应付餐费的 1.5 倍.
光盘行动,从我做起!
请输入进餐人数: 3
请点餐 2 份,注意荤素搭配!
请输入您的第 1 份餐品: 番茄炒鸡蛋
请服务员报价: 12
请输入您的第 2 份餐品: 红烧肉
请服务员报价: 48
您一共点了 2 份餐品,分别为: 番茄炒鸡蛋 红烧肉,当前总费用为: 60
现在是您的用餐时间,时间为 45 分钟.
========================================
现在请 AI 机器人扫描您盘中剩余食物:
请 AI 机器人报剩余餐品克数: 100
因您剩余餐品克数小于或等于 150,为您打八折,您最终需要支付 48.0 元.
感谢您为光盘行动做的贡献,谢谢您!
```

3.5.4 总结和启示

本案例模拟“光盘行动”号召下的餐饮系统,使用字符串存储所点餐品,使用数字类型变量存放餐费；使用循环结构进行点餐,使用选择结构模拟 AI 机器人扫描剩余食品并分段结算费用的功能,即综合使用前面两章所学知识进行设计和模拟。通过这个案例,可以很好地理解和掌握数据类型和程序控制结构。当然,该案例所实现功能较为简单,但是随着后续知识的讲授和掌握,大家可以使用列表或者字典存储所点餐品和相应的费用,也可以使用文件或者数据库存放菜谱,从而实现更复杂、更真实的餐饮系统。

“历览前贤国与家,成由勤俭破由奢。”尽管我们的物质资源逐渐丰富,但勤俭节约的观念和习惯仍未过时,也绝不能丢。今年的新冠肺炎疫情对全球粮食安全造成的冲击更向我们敲响了警钟。杜绝浪费行为,必须要将勤俭节约的观念根植于心、付诸于行。光盘行动不只是打包剩菜剩饭,更透露出尊重劳动的品德,合理消费的意识。让我们共同告别舌尖上的浪费,践行餐桌上的文明。

3.6 本章小结

本章详细介绍了 Python 的流程控制,主要包括顺序结构、单分支选择结构、双分支选择结构、多分支选择结构、while 循环结构、for…in 循环结构、break 语句和 continue 语句的概念和用法。在讲解过程中,结合大量实例,生动形象地演示了每种结构和语句的使用。并在最后结合“光盘行动餐饮系统”进行思政引导——勤俭节约。在学习本章内容时,可以模

仿实例，梳理算法流程，动手实践，熟练掌握 Python 流程控制语句的使用。

3.7 巩固训练

【训练 3.1】 编写程序判断用户输入的年份是否为闰年(判断闰年的条件是：能被 400 整除或者能被 4 整除但不能被 100 整除)。

【训练 3.2】 已知三角形边长利用海伦公式求三角形面积和周长(海伦公式：三角形的 3 条边长为 a,b,c,则 p=(a+b+c)/2,面积 area=$\sqrt{p*(p-a)*(p-b)*(p-c)}$)。

【训练 3.3】 某小学的学优生评定标准：语文、数学、英语和科学 4 门课程的总分不低于 380 分，且每科成绩不低于 95 分，编程判断某位同学是否为学优生。

【训练 3.4】 (简易版个税计算器)某公司员工小王每月税前工资为 salary，五险一金等扣除为 insurance，其他专项扣除为 other，请编程计算小王每月应缴纳税额 tax 和实发工资 payroll。(结果保留两位小数)

注：应缴纳税额＝税前收入－5000(起征点)－五险一金扣除－其他扣除

个人所得税＝应缴纳税额×适用税率－速算扣除数

实发工资＝税前工资－个人所得税－五险一金

税率表如表 3.3 所示。

表 3.3 2020 年居民个人工资、薪金所得税税率表

级数	应纳税所得额/元	税率(%)	速算扣除数/元
1	不超过 3000 元的	3	0
2	超过 3000 元至 12 000 元的部分	10	210
3	超过 12 000 元至 25 000 元的部分	20	1410
4	超过 25 000 元至 35 000 元的部分	25	2660
5	超过 35 000 元至 55 000 元的部分	30	4410
6	超过 55 000 元至 80 000 元的部分	35	7160
7	超过 80 000 元的部分	45	15 160

【训练 3.5】 幸运 52 猜数游戏(模仿幸运 52 中猜价钱游戏，编写程序，计算机随机产生一个正整数，让用户猜，并提醒用户猜大了还是猜小了，直到用户猜对为止，计算用户猜对一个数所用的秒数)。

【训练 3.6】 求出所有的水仙花数(水仙花数是指一个 3 位数，它的每个位上的数字的 3 次幂之和等于它本身。例如，1*1*1+5*5*5+3*3*3=153)。

【训练 3.7】 模拟输出超市购物小票。输入商品名称、价格、数量，算出应付金额。用户输入整钱，实现找零和抹零的功能，最后输出购物小票。运行效果图如图 3.19 所示。

【训练 3.8】 有一个分数数列$\frac{2}{1},\frac{3}{2},\frac{5}{3},\frac{8}{5},\frac{13}{8},\cdots$，编程计算这个数列的前 20 项之和(结果保留两位小数)。

【训练 3.9】 每行十个输出所有的 4 位“回文数”(“回文数”是一种特殊数字，从左边读和从右边读的结果是一模一样的)。

```
Python超市收银系统
商品个数:2
商品名称  单价      数量
egg 5.85 1.89
milk 48.5 1
应付金额:59.56
实收:100
Python超市购物小票
共购买2件商品
商品名称  单价      数量
egg      5.85     1.89
milk         48.5     1.0
应付:59.56
实收:100.0
找零40.4
```

图 3.19　输出超市购物小票

【训练 3.10】 编程实现：输出 1～1000 之间包含 3 的数字。如果 3 是连在一起的(如 233)则在数字前加上 &；如果这个数字是质数则在数字后加上 *(例如,3,13*,23*,&33,43*,…,&233*,…)。