第3章 选择结构

选择结构是程序设计中常用的基本结构，其功能是对给定的条件进行比较和判断，并根据判断结果采取不同的操作。Python 中常见的选择结构有三种：单分支选择结构、双分支选择结构和多分支选择结构。

3.1 单分支选择结构

单分支选择结构中，if 语句是最基本的选择结构语句，其语法格式如下：

```
if 表达式：
    语句块
```

功能：如果表达式的值为真，则执行其后的语句块，否则不做任何操作。其结构如图 3-1 所示。

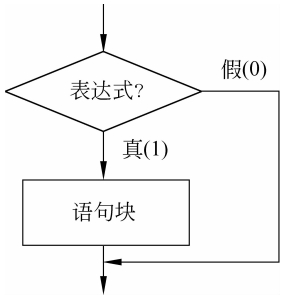


图 3-1 单分支选择结构

说明：if 语句中，表达式表示一个判断条件，可以是任何能够产生 True(1)或 False(0)的表达式或函数。表达式中一般包含关系运算符、成员运算符或逻辑运算符。

在 Python 中，整数 1 代表真(True)，0 代表假(False)。非零数字或非空对象也被视为真，任意的空数据结构或空对象以及特殊对象 None 都被视为假。

例如，选拔身高 T 超过 1.7 米且年龄 age 小于 25 岁的人，则该条件的逻辑表达式如下：

```
T>=1.7  and  age<25
```

例如，在某个游戏过程中，如果本关积分达到 1000，则提示“进入下一关”。用选择结构来实现此功能，部分代码如下：

```
if fen=1000:
   fen=0
   print ('进入下一关')
```

注意：这里的两条语句“fen＝0”和“print ('进入下一关') ”是 if 条件成立时要执行的同一个语句块，所以语句前的缩进要一致。Python 最具特色的功能就是使用缩进表示语句块，不需要使用大括号。缩进的字符数是可变的，但是同一个语句块的语句必须包含相同的缩进字符数，如果缩进不一致会导致逻辑错误。

3.2 双分支选择结构

双分支选择结构的语句用于区分条件的两种执行结果，语法格式如下：

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

功能：当表达式的值为真时，执行 if 后面的语句块 1，否则执行 else 后面的语句块 2，其结构如图 3-2 所示。

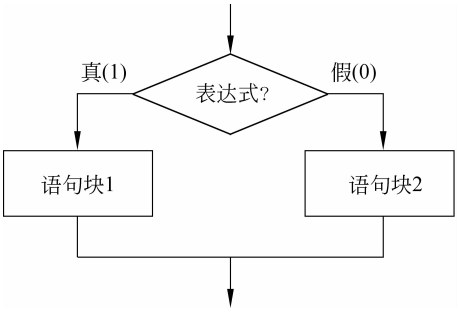


图 3-2 双分支选择结构

注意：双分支选择结构里，if 和 else 后面的冒号不能缺失，而且其后语句块 1 和语句块 2 都必须要有缩进，否则会产生语法错误。

【例 3-1】 已知某书店图书均九折销售，一次购书 100 元以上（包括 100 元）打八折。编写程序，要求根据输入的购书金额计算并输出应付款。

分析：设购书金额为 x 元，应付款为 y 元。由题意可知：

$$y=\begin{cases}0.9x, & x<100\\ 0.8x, & x\geqslant 100\end{cases}$$

计算 y 的值可利用双分支选择结构的语句来实现。

程序代码如下：

```
x=eval(input("请输入购书金额："))
if x<100:
```

```
    y=0.9*x
else:
    y=0.8*x
print ("优惠后应付金额是{:.2f}".format(y))
```

【例 3-2】　输入选拔赛分数，如果达到 60 分，就输出“合格”，否则输出“淘汰”。

程序代码如下：

```
x=input('输入分数: ')
fen=int(x)
if fen>=60:
  print('合格')
else:
  print('淘汰')
print('评价完成')
```

在 Python 中，双分支选择结构还有一种更简洁的表达式，适合在判断后直接返回特定的结果值，其语法格式如下：

```
表达式 1  if  条件  else  表达式 2
```

其中，表达式 1 和表达式 2 一般是一个数字类型或字符串类型的值。例如，例 3-2 中的双分支选择语句块可以改造为一条语句：

```
print('合格' if fen>=60 else '淘汰')
```

程序代码如下：

```
fen=int (input('输入分数: '))
print('合格' if fen>=60 else '淘汰')
print('评价完成')
```

运行结果如下：

```
输入分数: 78
合格
评价完成
>>>
```

3.3　多分支选择结构

Python 的多分支选择结构使用 if…elif…else 语句表达，其语法格式如下：

```
if  表达式 1:
    语句块 1
elif 表达式 2:
```

```
    语句块 2
    …
else:
    语句块 n
```

说明：该语句适用于多个变量、多种条件的情况，判断的顺序为条件表达式 1、条件表达式 2、条件表达式 3……一旦与某一条件匹配就执行该条件下的语句块，随即跳出整个 if 结构，即使下面仍有条件匹配也不再执行，因此要注意语句中条件表达式的排列次序，以免某些情况不被处理。如果没有任何条件成立，else 下面的语句块将被执行，else 子句是可选的。

if…elif…else 语句的多分支选择结构如图 3-3 所示。

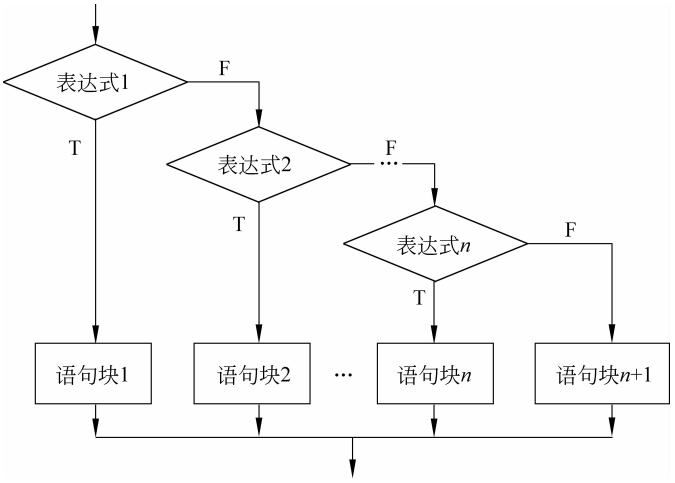


图 3-3　if…elif…else 语句的多分支选择结构

【例 3-3】 根据输入的学生百分制成绩，判定成绩的等级，输出相应的等级评价。等级评价标准如表 3-1 所示。

表 3-1　等级评价标准

等级	优	良	中	及格	不及格
分数	分数≥90	90＞分数≥80	80＞分数≥70	70＞分数≥60	分数＜60

该程序是一个多分支选择的问题，可以使用多分支选择结构 if…elif…else 语句实现。程序代码如下：

```
stumark=int(input("输入学生的成绩："))
if stumark>=90:
    grade="优"
elif stumark>=80:
    grade="良"
elif stumark>=70:
    grade="中"
elif stumark>=60:
```

```
    grade="及格"
else:
    grade="不及格"
print("此学生的等级评价是：", grade)
```

注意：多分支选择结构的一系列条件判断会从上到下依次执行，如果某个条件判断为 True，则执行该条件判断对应的语句块后，后面的条件判断就直接忽略，不再执行了。

思考：例 3-3 中 if…elif…else 部分代码如果写成下面的形式，结果会如何？

```
if stumark>=60:
    grade="及格"
elif stumark>=70:
    grade="中"
elif stumark>=80:
    grade="良"
elif stumark>=90:
    grade="优"
else:
    grade="不及格"
```

3.4　选择结构的嵌套

如果 if 语句中又包含一个或多个 if 语句，就称为 if 语句的嵌套。例如：

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

【例 3-4】 输入平面坐标系里某个点的坐标值(x,y)，判定输出该点在第几象限。

分析：在直角坐标系中，点所在的象限有以下几种情况。

(1) 当 $x=0$ 或 $y=0$ 时，点在坐标轴上，不在任何象限。

(2) 当 $x>0$ 时，有两种情况：若 $y>0$，点在第一象限；若 $y<0$，点在第四象限。

(3) 当 $x<0$ 时，有两种情况：若 $y>0$，点在第二象限；若 $y<0$，点在第三象限。

该程序可使用 if 语句的嵌套实现。程序代码如下：

```
import random
x=int(random.randint(-100,100))
y=int(random.randint(-100,100))
```

```
print("随机生成的坐标点为: ",x,",",y,end="\n")
if x==0 or y==0:
   print("该点在坐标轴上,不在任何象限")
elif x>0:
      if y>0:
        print ("该点在第一象限")
      elif y<0:
        print( "该点在第四象限")
elif x<0:
      if y>0:
        print( "该点在第二象限")
      elif y<0:
        print( "该点在第三象限")
print( "判断完成!")
```

程序运行结果如下：

```
随机生成的坐标点为: -93 ,-38
该点在第三象限
判断完成!
>>>
随机生成的坐标点为: 22 ,-58
该点在第四象限
判断完成!
>>>
```

3.5 应用实例

【例 3-5】 设计一个简易四则运算计算器，输入任意两个整数及运算符后，输出计算结果。如果输入的运算符不是＋、－、＊、/之一，则提示“运算符错误，无法计算!”。

分析：本例利用了选择结构的嵌套模式，先判断输入的运算符是否在列表['＋' ,'－', '＊','/']里，如果不在，就输出提示语并结束程序；如果输入的运算符在列表内，则进入一个多分支选择语句，根据不同的运算符求解并输出数值。

程序代码如下：

```
x,y=eval(input("请输入两个整数(用逗号分隔): "))
op=input("输入算术运算符: ")
operator=[ '+' , '-' , '*' , '/' ]
if op not in operator:
    print("运算符错误,无法计算!")
else:
  if op=='+':
    z=x+y
  elif op=='-':
    z=x-y
```

```
  elif op=='*':
    z=x*y
  elif op=='/':
    z=x/y
  print('%d %c %d='%(x,op,y), z)
```

程序运行结果如下：

```
请输入两个整数(用逗号分隔): 23,6
输入算术运算符: *
23 * 6=138
>>>
请输入两个整数(用逗号分隔): 78,12
输入算术运算符: /
78 / 12=6.5
>>>
请输入两个整数(用逗号分隔): 45,100
输入算术运算符: $
运算符错误,无法计算!
>>>
```

【例 3-6】 求 $ax^2+bx+c=0$ 方程的根。系数 a、b、c 由键盘输入。

分析：根据系数 a、b、c 的值，可得出方程有以下几种可能。

(1) $a=0$，不构成一元二次方程。

(2) 如果 $b^2-4ac=0$，一元二次方程有两个相等实根。

(3) 如果 $b^2-4ac>0$，一元二次方程有两个不等实根。

(4) 如果 $b^2-4ac<0$，一元二次方程有两个共轭复根，形式为 $m+ni$ 和 $m-ni$，其中 $m=-\frac{b}{2a}$、$n=\frac{\sqrt{b^2-4ac}}{2a}$。

本例中，在双分支选择结构里嵌套多分支选择来区分上面几种方程根的情况。程序开始，先判断是否是一元二次方程，如果是，再根据 p 的值即 b^2-4ac 的情况来分类判断属于哪种根的求解。如果是复根，先分别计算出实部 m 和虚部 n，再利用 print 的格式输出功能输出“$m+n$i”这样的复数形式。

程序代码如下，请注意这里的分支结构及其嵌套的含义和缩进格式。

```
import math
a,b,c=eval(input("输入方程式 ax^2+bx+c=0 的系数 a,b,c: "))
print("方程式为: %dx^2+%dx+%d=0 " %(a,b,c))
if a==0:
        print("这不是一元二次方程!")
else:
```

```
        print("是一元二次方程:")
        p=pow(b,2)-4*a*c
        if p==0:
            print("有两个相等实根,X1=X2=%f " %(-b / (2*a)), "\n")
        elif p>0:
            x1=(-b+math.sqrt(p))/(2*a)
            x2=(-b-math.sqrt(p))/(2*a)
            print("有两个不同的根:X1=%f , X2=%f " %(x1,x2))
        else:
            m=-b / (2*a)
            n=math.sqrt(-p) / (2*a)
            print("有两个复根:X1=%f+%fi , X2=%f-%fi \n" %(m,n,m,n))
```

程序运行结果如下：

```
输入方程式 ax^2+bx+c=0 的系数 a,b,c: 1,2,1
方程式为: 1x^2+2x+1=0
是一元二次方程:
有两个相等实根,X1=X2=-1.000000
>>>
输入方程式 ax^2+bx+c=0 的系数 a,b,c: 2,6,1
方程式为: 2x^2+6x+1=0
是一元二次方程:
有两个不同的根:X1=-0.177124 , X2=-2.822876
>>>
输入方程式 ax^2+bx+c=0 的系数 a,b,c: 3,2,4
方程式为: 3x^2+2x+4=0
是一元二次方程:
有两个复根:X1=-0.333333+1.105542i , X2=-0.333333-1.105542i
>>>
输入方程式 ax^2+bx+c=0 的系数 a,b,c: 0,2,5
方程式为: 0x^2+2x+5=0
这不是一元二次方程!
>>>
```

习题 3

一、简答题

1. 简述选择结构的种类。
2. 简述 if、if…else 和 if…elif…else 语句的语法格式和使用时的区别。
3. 什么情况下条件表达式会认为结果是 False?
4. Python 的多分支选择结构中，多个 elif 分支随意调整位置会有什么后果?

5. 当多分支选择结构中有多个表达式条件同时满足时，则每个与之匹配的语句块都被执行，这种说法是否正确？

二、选择题

1. 可以用来判断某语句是否在分支结构的语句块内的是(　　)。

A. 缩进　　B. 括号　　C. 冒号　　D. 分号

2. 以下各项中，不是选择结构里的保留字是(　　)。

A. if　　B. elif　　C. else　　D. elseif

3. 以下条件选项中，合法且在 if 中判断是 False 的是(　　)。

A. 24<=28&&28>25　　B. 24<28>25

C. 35<=45<75　　D. 24<=28<25

4. 以下针对选择结构的描述中，错误的是(　　)。

A. 每个 if 条件后或 else 后都要使用冒号

B. 在 Python 中，没有 select-case 语句

C. Python 中的 pass 是空语句，一般用作占位语句

D. elif 可以单独使用，也可以写为 elseif

5. Python 中，(　　)是一种更简洁的双分支选择结构。

A. '合格' if fen>=60 else '淘汰'

B. if fen>=60 '合格'else '淘汰'

C. if fen>=60：'合格'：'淘汰'

D. if fen>=60：'合格' elseif '淘汰'

三、填空题

1. Python 中，用于表示逻辑与、逻辑或、逻辑非运算的关键字分别是_____、_____和_____。

2. 表达式 1 if 2>3 else (4 if 5>6 else 7) 的值为_____。

3. if…else 双分支选择结构中，else 与 if 语句后面必须有_____符号。

4. Python 中，表示条件真或假的两个关键字是_____和_____。

5. 算术运算符、关系运算符、逻辑运算符中优先级最高的是_____。

四、编程题

1. 由键盘输入三个整数，请利用多分支选择结构语句编程，输出其中最大的数。

2. 编程查询某日汽车限行车牌尾号。限行规则为工作日每天限行两个号：车牌尾数为 1 和 6 的机动车周一限行；车牌尾数为 2 和 7 的周二限行；车牌尾数为 3 和 8 的周三限行；车牌尾数为 4 和 9 的周四限行；车牌尾数为 5 和 0 的周五限行。请输入星期几的代号(1～7)，输出相应的限行车辆尾号。

3. 按照联合国公布的人类年龄划分标准，人的一生分为 5 个年龄段：未成年人(小于 18 岁)、青年人(18～65 岁)、中年人(66～79 岁)、老年人(80～99 岁)、长寿老人(大于或等于 100 岁)。编写程序，根据输入的实际年龄，判断该年龄属于哪个人生阶段。

4. 编写程序，输入三角形的三条边长，如果能构成三角形，则判断是等腰、等边、直角三角形，还是一般三角形。

5. 某汽车运输公司开展整车货运优惠活动，货运收费根据各车型货运里程的不同而定，其中一款货车的收费标准如下，编程实现自动计算运费问题。

(1) 距离在 100km 以内：只收基本运费 1000 元。

(2) 距离为 100～500km：除基本运费外，超过 100km 的部分，运费为 3.5 元/千米。

(3) 距离超过 500km：除基本运费外，超过 100km 的部分，运费为 5 元/千米。

第4章 循环结构

在解决问题的过程中，有时需要有规律性地重复操作许多次，对于计算机来说，就需要重复执行某些语句，解决方式就是采用循环结构。

循环结构是在一定条件下反复执行某段程序的流程结构，被反复执行的程序段被称为循环体，能否继续重复执行，由循环的终止条件来决定。

Python 中使用 for 语句和 while 语句实现循环结构。

4.1 while 循环结构

while 语句是条件循环语句，当条件满足时执行循环体，常用于控制循环次数未知的循环结构。while 语句的语法格式如下：

```
while 表达式：
循环体语句块
```

功能：先判断 while 表达式的值，如果为真（True）则执行一次循环体语句块，然后返回 while 处再次判断表达式的值，若仍为 True 则再一次执行循环体，如此反复，直到表达式的值为假（False）或者发生异常时，循环终止。while 循环结构如图 4-1 所示。

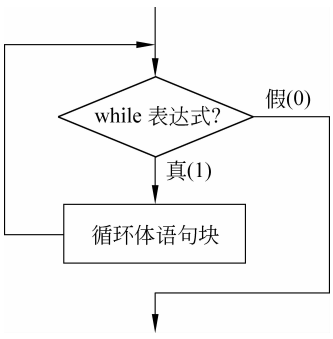


图 4-1 while 循环结构

说明：

（1）while 循环是先判断再执行，因此循环体语句块有可能一次也不执行。

（2）为避免造成死循环，循环体语句块里一定要有能终止循环的语句，比如能改变 while 表达式值的变量，或辅助控制语句 break 和 continue。

（3）要注意循环体语句块的缩进格式，while 语句只执行其后的一条或一组同一缩进格式的语句块。

【例 4-1】 求 1+2+3+…+100 的值。

分析：本例中为了实现求和运算，重复执行加法操作，可以用累加指令 s=s+i 实现。其中，i 是一个从 1 变化到 100 的变量，每次递增 1，可用 i=i+1 或 i+=1 指令实现。当 i 增加到 100 后需要终止循环，这可以用 i<=100 作为循环控制表达式。

程序代码如下：

```
s=0
i=1
while i<=100:
    s=s+i          #也可以写为 s+=i
    i=i+1          #也可以写为 i+=1
print('1 到 100 的和为：',s)
```

思考：

(1) 如果要计算 1～100 所有奇数或者偶数的和，应怎样编写程序代码？

(2) 例 4-1 是对变量 i 从小到大递增求和，若改为从大到小递减并求和，应怎样编写程序代码？

【例 4-2】 有一阶梯，如果每步跨 2 阶，最后余 1 阶；每步跨 3 阶，最后余 2 阶；每步跨 5 阶，最后余 4 阶；每步跨 6 阶，最后余 5 阶；每步跨 7 阶，正好到达阶梯顶，问阶梯至少有多少阶？

分析：本例可以采用穷举法(也称枚举法)，即对可能出现的各种情况一一进行测试，判断是否满足条件。对于不可预测循环次数的问题，一般选择无限循环 while 语句。

本例中，控制 while 循环是否继续的条件表达式是一个常量 True，它与循环体内的 break 语句结合，避免了死循环的产生。这里台阶数 n 从 0 开始判断是否符合要求，如果不符合，台阶数就增加 1，然后返回 while 处再次循环去判断台阶数，如此反复，直到某一个 n 值符合要求就执行 break 中止循环。

程序代码如下：

```
n=0
while True:
    if n %2==1 and n %3==2 and n %5==4 and n %6==5 and n %7==0:
        break
    n=n+1
print('第一个符合的台阶数是：',n)
```

运行结果如下：

```
第一个符合的台阶数是：119
>>>
```

while 循环还有一种使用保留字 else 的扩展模式，其格式如下：

```
while 表达式：
```

```
    语句块 1
else:
    语句块 2
```

功能：如果 while 循环体内没有被 break、return 等打乱，而是正常执行完循环体，或者因 while 循环条件不满足而不能进行循环，这时就要执行 else 语句下的内容，所以可在 else 下放置一些描述 while 循环执行情况的语句。

【例 4-3】 输出某人英文名字中的字母。

分析：从第一个字母开始到最后一个字母为止，源程序正常执行循环体，分别打印名字中的每个字母，循环完成后执行 else 部分，输出提示语“---拼写结束---”。

程序代码如下：

```
name="Tom"
x=0
while x<len(name):
    print("第"+str(x+1)+"个字母: "+name[x])
    x+=1
else:
    print("---拼写结束---")
```

运行结果如下：

```
第 1 个字母: T
第 2 个字母: o
第 3 个字母: m
---拼写结束---
>>>
```

把这个程序改造一下，加入一个 break 保留字，如例 4-4。

【例 4-4】 某一个应用程序中，注册用户信息时，要求密码是字母和数字组合，如果输入的密码不符合要求，则要求重新输入密码。编写程序实现此功能。

程序代码如下：

```
string=input("请输入新密码: ")
i=0
while i<len(string):
    if not string[i].isalnum():
        msg="密码只能是字母和数字组合,请重新输入!"
        break
    i+=1
else:
    msg="密码输入成功。"
print(msg)
```

此程序里，在 while 循环过程中发现某一个字符不是数字或字母，就输出“密码只能

是字母和数字组合，请重新输入！”提示语，并通过 break 跳出循环。如果 while 循环是根据循环条件正常结束的，则执行 else 下的语句“密码输入成功。”。所以根据这两个状态即可判断循环的执行情况。

程序运行结果如下：

```
请输入新密码：abc$ #123
密码只能是字母和数字组合，请重新输入！
>>>
请输入新密码：abc123qq
密码输入成功。
>>>
```

4.2 for 循环结构

在 Python 中，如果循环次数很明确，一般采用遍历循环来构造循环结构。循环次数通过遍历结构中的元素个数来体现，采用 for 语句依次把列表或元组中的每个元素迭代出来。在 Python 中，可以使用 for 循环来遍历列表、元组、字典、字符串、函数等许多含有序列的项目。

for 循环结构的语法格式如下：

```
for 循环变量 in 遍历结构：
    循环体语句块
```

功能：循环开始时，从遍历结构中逐一提取元素，给 for 指定循环变量，对于每个提取的元素都执行一次语句块，如此反复，直到遍历完每一个元素。

【例 4-5】 编程计算 1－2＋3－4＋5－6＋…＋n 的值，n 由用户输入。

分析：首先输入用户要求的数字 n，确定遍历方式 rang()，循环开始对提取的值做累加，用(－1)**(x＋1)或者 pow(－1，x＋1)来控制加减的变化，遍历循环结束后输出最后结果。

程序代码如下：

```
n=eval(input("输入数列 s=1-2+3-4+…n 中的 n 值:"))
s=0
for x in range(1,n+1):
    s=s+pow(-1,x+1) * x
print("1-2+3-4+…+"+str(n)+"的值为：",s)
```

range()函数返回一个可迭代对象，常用在 for 循环结构中。其语法格式如下：

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

参数说明：

start：从 start 开始计数，可省略，默认是从 0 开始。例如，range(5)等价于 range(0，5)。

end：计数到 end 结束，但不包括 end 值。例如，range(0,5)表示[0,1,2,3,4]，没有 5。

step：步长，可省略，默认为 1。例如，range(0,5) 等价于 range(0,5,1)。

例如：

```
>>>for x in range(1,11):                #1~10 循环
      print x
>>>for x in range(1,11,2):              #只循环奇数 1、3、5、7、9
      print x
```

如果要遍历序列的索引，也可以将 range()和 len()组合起来使用，例如：

```
>>>city=['北京','上海', '西安','成都']
>>>for x in range()len(city)):
      print(x, city[x])
```

运行结果如下：

```
0 北京
1 上海
2 西安
3 成都
>>>
```

【例 4-6】 输入一串字符，统计空格、数字、字母和其他字符的个数。

分析：利用 for 循环语句，对这串字符进行遍历，循环过程中对提取的字符进行判断，采用 if 语句，结合字符串方法 isspace()、isdigit()、isalpha()分别判断统计。

程序代码如下：

```
string=input("请输入要统计的字符串(回车结束): ")
space_n,digit_n,letter_n,other_n=0,0,0,0
for i in range(len(string)):
    if string[i].isspace():
        space_n+=1
    elif string[i].isdigit():
        digit_n+=1
    elif string[i].isalpha():
        letter_n+=1
    else:
        other_n+=1
print('空格个数是: ',space_n)
print('数字个数是: ',digit_n)
print('字母个数是: ',letter_n)
print('其他字符个数是: ',other_n)
```

4.3 循环控制辅助语句

循环结构里用来辅助控制循环执行的语句有 break 和 continue。

4.3.1 break 语句

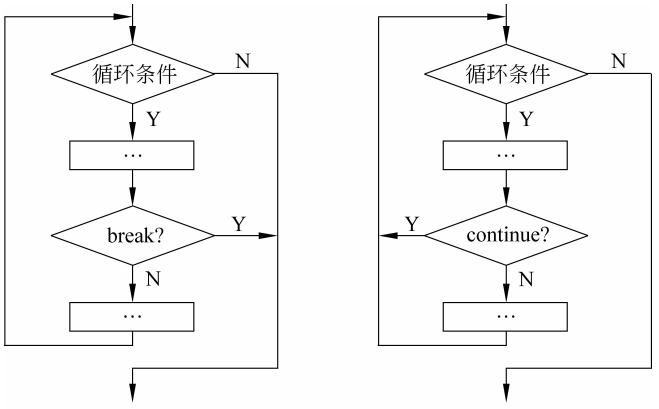
在 for 或 while 循环体的执行过程中，当满足某个条件，需要中止当前循环跳出循环体时，就需要使用 break 语句来实现这个功能。此时，break 语句结束了整个还没有执行完的循环过程，而不考虑此时循环体的条件是否成立。

注意：如果存在循环嵌套，break 语句只停止执行它所在层的循环，但仍然继续执行外层循环体。

4.3.2 continue 语句

continue 语句用来跳过当前这一轮循环的剩余语句，继续进行下一轮循环。对于 while 循环，执行 continue 语句会返回循环开始处判断循环条件。而对于 for 循环，则接着遍历循环列表。

break 语句和 continue 语句的结构如图 4-2 所示。



(a) break 语句　　(b) continue 语句

图 4-2　break 语句和 continue 语句的结构

【例 4-7】 搜索字符串“鲁豫皖”，遇到字符“豫”时，分别采用 break 语句或 continue 语句中断，观察结果。

程序代码如下：

```
for s in "鲁豫皖":
    if s=="豫":
        break
    print(s,end="")          #结果为"鲁"
print("")
```

```
for t in "鲁豫皖":
    if t=="豫":
        continue
    print(t,end="")      #结果为"鲁皖"
```

上面程序中，同样是搜索字符串“鲁豫皖”时遇到“豫”，但采用 break 语句和 continue 语句导致的结果不一样。break 语句的功能是结束整个循环体，不再继续，所以对字符串“鲁豫皖”遍历到“豫”时即停止，只输出了前面的“鲁”字。而 continue 语句只是结束当前这次循环，跳过字符“豫”，继续后面的遍历，所以输出结果是“鲁皖”。

4.4　循环的嵌套

在一个循环体内又包含了循环结构，这种结构称为循环嵌套。循环嵌套对 while 循环和 for 循环语句都适用。例如，可以在 while 循环中嵌入 for 循环，也可以在 for 循环中嵌入 while 循环。

使用嵌套循环时应注意，内循环控制变量与外循环变量不能同名，并且外循环必须完全包含内循环，不能相互交叉。

【例 4-8】　打印九九乘法表。

分析：可采用循环嵌套来实现九九乘法表的打印。用外层循环变量 i 控制每行的输出：“i in range(1,10) ”；内层循环变量 j 控制每行内各个等式的输出：“j in range(1,10)”。除了循环流程，还要考虑到乘法表每个等式的打印位置，即利用 print 的格式化输出来组合每个等式，用“end＝" ”来保证每行内各个等式连续输出不换行，当内层循环结束后执行 print()实现输出换行。

程序代码如下：

```
for i in range(1,10):
    for j in range(1,10):
        print('%d*%d=%2d  '%(i,j,i*j),end='')
    print()
```

程序运行结果如下：

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

如果把内循环 for j in range(1,10)的遍历范围终止值由常量 10 改为变量 i＋1，则输

出的乘法表由矩形变为三角形，效果如下：

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

思考：如果把 print('%d*%d=%2ld'%(i,j,i*j),end="")中的(i,j,i*j)修改成(j,i,i*j)，输出内容又有何改变？

【例 4-9】 输出 2～100 的所有素数。

分析：素数是一个只能被 1 和它本身整除的整数。判断一个整数 num 是否为素数，一个简单的方式就是逐一用 2～num－1 的每个数去除它，如果能除尽则不是，否则就是。判断一个整数是否是素数需要用一个循环结构，判断多个整数是否是素数，就需要用循环嵌套来实现。

程序代码如下：

```
for num in range(2,101):     #此循环实现对 2~100 的每个整数进行判断
   i=2
   while (i<=num-1):         #此循环判断某一个 num 是否是素数
      if  num %i==0:
        break                #当前整数不是素数,跳出内循环开始判断下一个整数
      i+=1
   else:                     #由 i 递增到 i=num,导致循环正常结束,得出结论"是素数"
      print(num,"是素数")
```

本例中，判断某一个整数是否是素数的 while 循环条件除了 i<=num－1 以外，还可以是 i<=(num/2)或者 i<=math.sqrt(num)。

4.5 应用实例

【例 4-10】 我国数学家张丘建在其著作《算经》一书中提出了“百鸡问题”：“鸡翁一值钱 5，鸡母一值钱 3，鸡雏三值钱 1。百钱买百鸡，问鸡翁、母、雏各几何？”对于这个数学问题可列出数学方程如下：

$$\text{Cock} + \text{Hen} + \text{Chick} = 100$$

$$\text{Cock} \times 5 + \text{Hen} \times 3 + \text{Chick}/3 = 100$$

显然，这是一个不定方程，适合采用穷举法求解。依次取 Cock、Hen、Chick 值域中的一个值，代入方程式进行判断，如果满足条件即可得到答案。

程序代码如下：

```
#三个变量分别是公鸡 cock、母鸡 hen、鸡雏 chick
cock=0
while cock<=20:                    #公鸡最多不可能大于 20
    hen=0
    while hen<=33:                 #母鸡最多不可能大于 33
        chick=100-cock-hen         #鸡雏的数量
        if cock*5+hen*3+chick/3==100:
            print("公鸡=%d,母鸡=%d,雏鸡=%d"%(cock,hen,chick))
        hen=hen+1
    cock=cock+1
```

思考：此例也可以采用 for 循环结构来解决，应如何编写程序代码？

注意：穷举法的基本思想是根据题目的部分条件确定答案的大致范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。

【例 4-11】 斐波那契数列(Fibonacci Sequence)又称黄金分割数列，因以兔子繁殖为例而引入，故又称“兔子数列”。斐波那契数列是指 1、1、2、3、5、8、13、21、34、…，在数学上，斐波那契数列以递推的方法定义如下：

$f(1)=1,\quad f(2)=1,\quad \cdots,\quad f(n)=f(n-1)+f(n-2)\quad (n\geqslant 3, n\in \mathbf{N})$

编程时，这里可以设置数列前两项为 $f1$、$f2$，然后从第 3 项开始，运用递推公式 $f3=f1+f2$ 先推出下一项的值，随后向后更新 $f1$ 和 $f2$，即 $f1=f2$、$f2=f3$，再返回计算新的 $f3=f1+f2$。如此反复，依次递推得出所需项数的斐波那契数列。

程序代码如下：

```
n=eval(input("请输入斐波那契数列的项数 n:"))
f1,f2,f3=1,1,0
print('%d\n%d'%(f1,f2))
for i in range(3,n+1):
    f3=f1+f2
    print(f3)
    f1=f2
    f2=f3
```

思考：代码中也可以不引入变量 f3，利用 f1,f2=f2,f1+f2 也能实现递推功能，应如何编写程序代码？

注意：递推是计算机中的一种常用算法，是指按照一定的规律来计算序列中的每项，通常是通过序列前面的一些项来推算出后续项的值。其思想是把一个复杂、庞大的计算过程转化为简单过程的多次重复。

【例 4-12】 “石头剪子布”是一种流传多年的猜拳游戏。简单明了的规则使游戏没有任何规则漏洞可钻，单次玩法比拼运气，多回合玩法比拼心理博弈，使这个古老的游戏同时拥有运气与技术两种特性，深受世界人民喜爱。

游戏规则：石头打剪刀，布包石头，剪刀剪布。互相克制的原则：剪子剪不动石头(石头胜利)；布被剪子剪开(剪子胜利)；石头被布包裹(布胜利)。如果双方出示了一样的

手势，则为平局。通常一局游戏可能会被重复多次，以三局两胜或五局三胜来决定最终的胜负。

该游戏的编程过程描述如下。

（1）游戏开始，计算机产生一个 1～3 的随机数，分别代表“石头”“剪子”“布”。

（2）真人玩家输入自己出拳的名称。如果输入的不是“石头”“剪子”“布”中的一个，则提示重新输入；如果输入的是“石头”“剪子”“布”，则分别与计算机值比较，判断此次输赢。

（3）每次判断都统计有效出拳的次数，如果达到 3 次，则结束游戏。

完整程序代码如下：

```
from random import randint
idx=0
while True:
    x=randint(1,3)
    if x==1:
      machine="石头"
    if x==2:
      machine="剪子"
    if x==3:
      machine="布"
    man=input("输入你出的拳(石头、剪子、布),然后回车：")
    list1=["石头","剪子","布"]
    if (man not in list1):
        print ("输入有误！请重新输入!")
        idx-=1
    elif man==machine:
        print ("计算机出了:"+machine+",-----平局!")
    elif (man=='石头'and machine=='剪子') or (man=='剪子' and machine=='布')
    or (man=='布' and machine=='石头'):
        print ("计算机出了:"+machine+",-----你赢了!")
    elif (man=='剪子' and machine=='石头') or (man=='布' and machine=='剪子')
    or (man=='石头' and machine=='布'):
        print ("计算机出了:"+machine+",-----你输了!")
    idx+=1
    if idx==3:
        print("-----------------------------")
        print("已经有效出拳 3 次,游戏结束。")
        break
```

程序执行结果如下：

```
输入你出的拳(石头、剪子、布),然后回车：布
计算机出了:剪子,-----你输了!
```