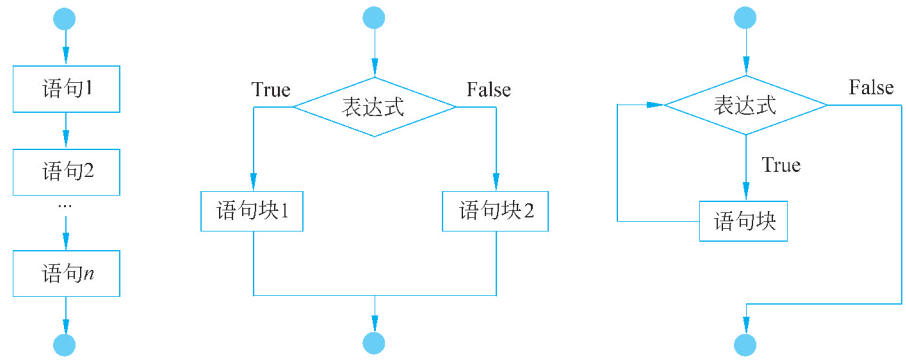
第3章 程序控制结构

人们生活或工作中做事通常有一定的流程或者顺序，程序执行也有一定的顺序，默认是按照从上而下按顺序执行，但在实际的代码中，程序经常需要做判断、循环等，因此需要有多种流程控制语句来实现程序的跳转和循环等功能，如分支控制语句、循环控制语句和退出程序语句等。

本章首先介绍程序控制结构的三种类型，然后介绍选择语句、循环语句和跳转语句。

3.1 程序控制结构概述

Python 程序具有三种典型的控制结构，如图 3-1 所示。



(a) 顺序结构　(b) 选择结构　(c) 循环结构

图 3-1　三种控制结构

(1) 顺序结构：在程序执行时，按照语句的顺序，从上而下一条一条地顺序执行，是结构化程序中最简单的结构。

（2）选择结构：又称为分支结构，分支语句根据一定的条件决定执行某一部分的语句序列。

（3）循环结构：使某个代码块根据一定的条件执行若干次。采用循环结构可以实现有规律的重复运行。

3.2 选择控制结构

选择结构也称为分支结构，就是对语句中的条件进行判断，根据判断的结果选择不同的分支执行。

选择语句可以分为以下三种形式。

（1）简单的 if 语句。

（2）if…else 语句。

（3）if…elif…else 多分支语句。

3.2.1 if 语句

if 语句是最简单的选择语句，通常表现为“如果满足某条件，那么就执行某代码块”，其结构如图 3-2 所示。

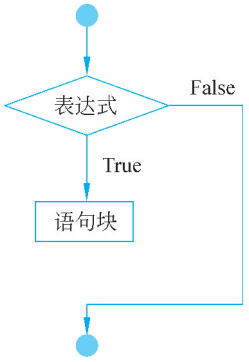


图 3-2 if 语句结构

其中，表达式可以是一个单一的值或者变量，也可以是由运算符组成的复杂语句。如果表达式的值为真，则执行该语句块，执行完语句块之后继续向下执行。如果表达式的值为假，则跳过这一语句块，继续执行后面的语句。参考示例如下。

【文件 3.1】 demo9.py

```
a =66
b =200
if b >a:
    print("b is greater than a")
print("End")
```

运行结果：

```
b is greater than a
End
```

3.2.2 if…else 语句

if…else 语句与 if 语句相比，多了一个分支，通常表现为“如果满足某条件，那么就执行某代码块，否则执行另一代码块”，其结构如图 3-3 所示。

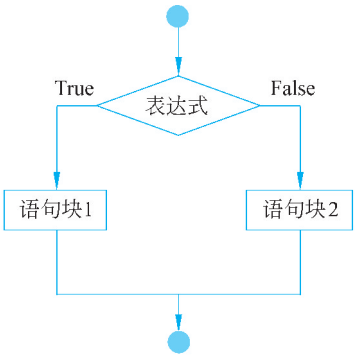


图 3-3 if…else 语句结构

其中，表达式可以是一个单一的值或者变量，也可以是由运算符组成的复杂语句。如果表达式的值为真，则执行语句块 1，执行完语句块 1 之后继续向下执行。如果表达式的值为假，则执行语句块 2，然后继续执行后面的语句。需要注意的是，else 不能单独使用，必须和 if 组合使用。参考示例如下。

【文件 3.2】 demo10.py

```
a =int(input())
b =int(input())
if b >a:
    print("b is greater than a")
else:
    print("b is smaller than a")
print("End")
```

在上述程序中，变量 a、b 分别接收用户从键盘的输入，然后比较 a 和 b 的大小，如果 b 大于 a，输出结果：

```
b is greater than a
End
```

如果 b 不大于 a，输出结果：

```
b issmaller than a
End
```

3.2.3 if…elif…else 多分支语句

if…else 语句可用于两个分支的情况，如果是多分支，则可以使用 if…elif…else 多分支语句。elif 语句可以重复多次，最后一个为 else 语句。通常表现为“如果满足某条件，那么就执行某代码块；否则，如果满足某另外一个条件，则执行另一代码块”，其结构如图 3-4 所示。

其中，表达式可以是一个单一的值或者变量，也可以是由运算符组成的复杂语句。如果表达式 1 的值为真，则执行语句块 1，执行完语句块 1 之后继续向下执行。如果表达式 1 的值为假，则执行表达式 2。如果表达式 2 为真，则执行语句块 2；若为假，则执行表达式 3。以此类推，如果所有的表达式都为假，才执行 else 语句。需要注意的是，elif 和 else 都不能

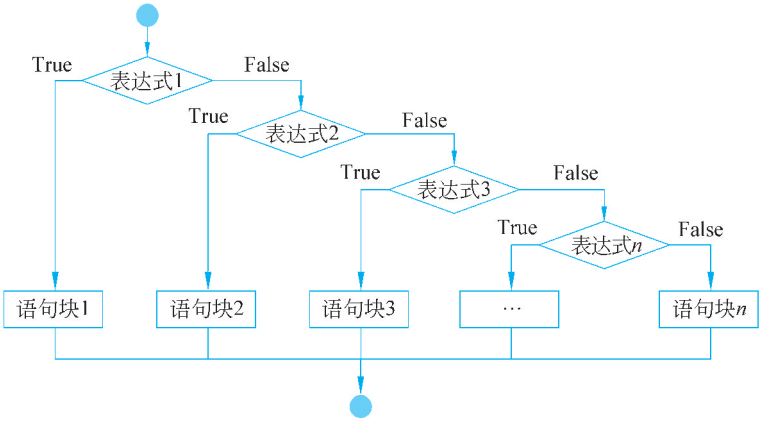


图 3-4　if…elif…else 语句结构

单独使用，必须和 if 组合使用。参考示例如下。

【文件 3.3】　demo11.py

```
a =int(input())
b =int(input())
if b >a:
    print("b is greater than a")
elif b <a:
    print("b is smaller than a")
else:
    print("b is equal to a")
print("End")
```

在上述程序中，变量 a、b 分别接收用户从键盘的输入，然后比较 a 和 b 的大小，如果 b 大于 a，输出结果：

```
b is greater than a
End
```

如果 b 小于 a，输出结果：

```
b is smaller than a
End
```

如果 b 既不大于 a，也不小于 a，输出结果：

```
b is equal to a
End
```

3.2.4　if 语句的嵌套

if 语句、if…else 语句和 if…elif…else 语句可以相互嵌套，嵌套时，注意处于同一缩进的关键字为一对组合。

在 if 语句中嵌套 if…else 语句，形式如下。

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

第二个 if 和下面的 else 处于同一个缩进，它俩为一对组合。

在 if…else 语句中嵌套 if…else 语句，形式如下。

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

由缩进可以明显看出 if 和 else 的对应关系，如果处于同一缩进，遵循同一代码块内的就近原则。

嵌套使用 if 语句的示例如下，接收用户输入的一个数字，并判断该数是 0、正数还是负数，然后判断是奇数还是偶数，示例代码如文件 3.4 所示。

【文件 3.4】 digit.py

```
num =int(input("请输入一个整数: "))

if num >0:
    print("这是一个正数")
    if num %2 ==0:
        print("这是一个偶数")
    else:
        print("这是一个奇数")
elif num <0:
    print("这是一个负数")
    if num %2 ==0:
        print("这是一个偶数")
    else:
        print("这是一个奇数")
else:
    print("这是零")
```

若输入“12”，输出结果为

```
请输入一个整数: 12
这是一个正数
这是一个偶数
```

若输入“－13”，输出结果为

```
请输入一个整数：-13
这是一个负数
这是一个奇数
```

若输入“0”，输出结果为

```
请输入一个整数：0
这是零
```

3.3 循环控制结构

生活中有很多循环的例子，如一页一页印刷图书、绕着操场一圈一圈跑步。循环语句将根据指定的条件，多次执行同一段代码。在 Python 中，循环语句主要有以下两种形式。

（1）while 循环。

（2）for 循环。

3.3.1 while 循环

while 循环在每次循环开始前先判断条件是否成立。如果计算结果为 true，就把循环体内的语句执行一遍；如果计算结果为 false，就直接跳到 while 循环的末尾，继续往下执行，其结构如图 3-5 所示。

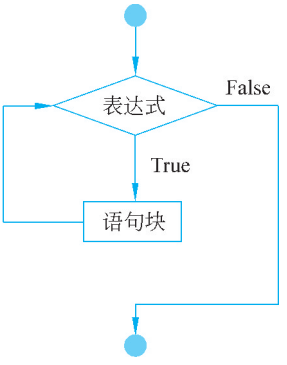


图 3-5 while 循环结构

下面使用 while 循环计算 1～100 的和，代码如文件 3.5 所示。

【文件 3.5】 demo12.py

```
n =1
sum =0
while(n <=100):
    sum =sum +n
    n +=1
print(sum)
```

while 循环语句的特点是：如果第 3 行 while(n <= 100)的条件不成立，则循环一次都

不执行。在写循环时，要注意不要造成“死循环”，即无限循环下去，不会停止，如上例中，如果将循环体中的语句 n ＋＝ 1 去掉，则判别条件恒为真，循环会无休止地执行下去。

3.3.2 for 循环

for 循环一般用于循环次数已知的情况，常用于迭代序列（即列表、元组、字典、集合或字符串），其结构如图 3-6 所示。

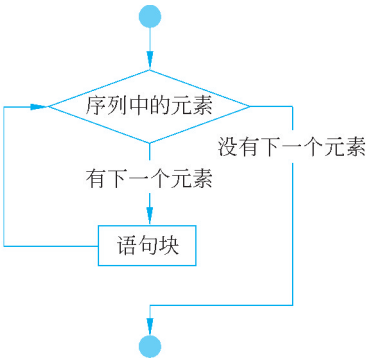


图 3-6 for 循环结构

下面使用 for 循环计算 1～100 的和，示例代码如文件 3.6 所示。

【文件 3.6】 demo13.py

```
sum = 0
for i in range(1,101):      # range()函数可以生成其范围内的整数，含左不含右
    sum = sum + i
print(sum)
```

for 循环可用于迭代列表中的元素：

```
sites = ["Baidu", "Google","Runoob","Taobao"]
for site in sites:
    print(site)
```

其输出结果为

```
Baidu
Google
Runoob
Taobao
```

for 循环遍历元组、字典、集合中元素同理，也可用于打印字符串中的每个字符。

```
word = 'Python'

for letter in word:
    print(letter)
```

其输出结果为

```
p
y
t
h
o
n
```

3.3.3 嵌套循环

前面学习了 while 循环和 for 循环，循环不但可以单独使用，还可以相互嵌套。嵌套循环就是一个循环体，包含另外一个完整的循环结构。而在这个完整的循环体内，还可以继续嵌套其他的循环结构。while 循环结构和 for 循环结构都可以嵌套，它们也可以相互嵌套。

在 while 循环中嵌套 while 循环的格式如下，注意留意缩进。

```
while 表达式 1:
    while 表达式 2:
        语句块 2
    语句块 1
```

在 for 循环中嵌套 for 循环的格式如下。

```
for 迭代变量 1 in 对象 1:
    for 迭代变量 2 in 对象 2:
        语句块 2
    语句块 1
```

在 while 循环中嵌套 for 循环的格式如下。

```
while 表达式:
    for 迭代变量 in 对象:
        语句块 2
    语句块 1
```

在 for 循环中嵌套 while 循环的格式如下。

```
for 迭代变量 in 对象:
    while 表达式:
        语句块 2
    语句块 1
```

使用 for 循环嵌套实现打印九九乘法表。

【文件 3.7】 demo14.py

```
for i in range(1, 10):
    for j in range(1, 10):
        if i >=j:
            print("{} * {}={}".format(j, i, j * i), end='\t')
    print()  #换行
```

其输出结果为

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

使用 while 循环嵌套实现打印九九乘法表的示例如下。

【文件 3.8】 demo15.py

```
row =1
while row <=9:
    column =1
    while column <=row:
        print('{} * {}={}'.format(column, row, row * column), end='\t')
        column +=1
    print()
    row +=1
```

其输出结果与 for 嵌套循环相同。

3.4 跳转语句

Python 语言支持多种跳转语句，可与选择控制语句和循环控制语句配合使用。常见的跳转语句有 break、continue 和 pass 语句。

3.4.1 break 语句

通过使用 break 语句，可以在循环遍历所有项之前停止循环，即只要程序执行到 break 语句，就会中止循环体，不再做后续的循环。如果在嵌套循环的内层循环中遇到 break 语句，则跳出当前的循环体，回到外层循环，结构如图 3-7 所示。

在 while 循环语句中使用 break 跳转语句的形式如下。

```
while 表达式 1:
    语句块
    if 表达式 2:
        break
```

在 for 循环语句中使用 break 跳转语句的形式如下。

```
for 迭代变量 in 对象:
    语句块
if 表达式:
    break
```

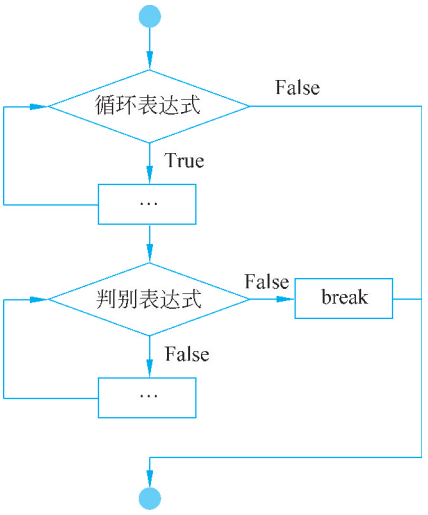


图 3-7　break 语句结构

示例如文件 3.9 所示，如果 x 是“banana”，则退出循环。

【文件 3.9】　demo16.py

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
print("End")
```

运行结果为

```
apple
banana
End
```

可以看到在第二次循环时，x 为“banana”，所以就退出循环，没有继续输出“cherry”。如果想要在 x 为“banana”时退出循环，并且在打印之前中断，即不打印出“banana”，代码如下。

【文件 3.10】　demo17.py

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        break
    print(x)
print("End")
```

运行结果

```
apple
End
```

可以看出，修改后的程序先判断 x 是否等于“banana”，如果不是，才打印输出，如果 x 为

“banana”，则直接中断，退出循环，不执行后面的 print(x)。退出循环体后，继续执行后续语句 print("End")。

while 循环中 break 语句的示例如文件 3.11 所示，当 n 大于 3 时，即停止循环。

【文件 3.11】 demo18.py

```
n =1
while True:
    print(n)
    n +=1
    if n >3:
        break
```

运行结果

```
1
2
3
```

下面查看在嵌套循环中使用 break 语句，示例代码如文件 3.12 所示。先看如果不使用 break 语句的执行效果。

【文件 3.12】 demo19.py

```
color =["red", "green", "blue"]
fruits =["apple", "banana", "cherry"]

for x in color:
  for y in fruits:
    print(x, y)
```

运行结果：

```
red apple
red banana
red cherry
green apple
green banana
green cherry
blue apple
blue banana
blue cherry
```

使用 break 语句的示例代码如文件 3.13 所示。

【文件 3.13】 demo20.py

```
color =["red", "green", "blue"]
fruits =["apple", "banana", "cherry"]

for x in color:
  for y in fruits:
    print(x, y)
```

```
    if y =="banana":
        break
```

运行结果：

```
red apple
red banana
green apple
green banana
blue apple
blue banana
```

内层循环执行到 y 等于“banana”时，遇到 break 语句，跳出内层循环，继续外层循环，x 继续等于下一个元素。

3.4.2 continue 语句

continue 语句也可以跳出循环，但是与 break 语句有所不同：break 语句跳出循环，并停止循环，继续执行循环体之外的语句；而 continue 语句只能跳出本次循环，不执行本次循环的循环体中的后续语句，但需要继续执行下一次循环，continue 语句的结构如图 3-8 所示。

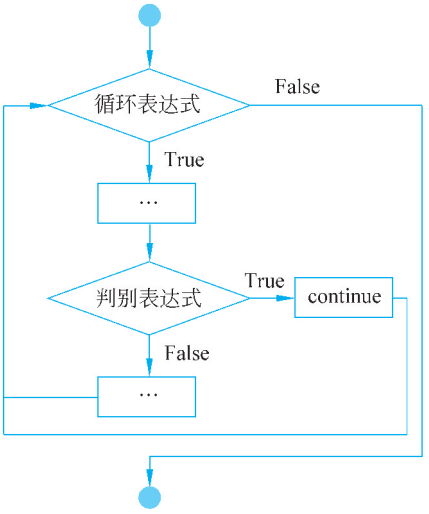


图 3-8 continue 语句结构

在 while 循环语句中使用 continue 跳转语句的形式如下。

```
while 表达式 1:
    语句块
    if 表达式 2:
        continue
```

在 for 循环语句中使用 continue 跳转语句的形式如下。

```
for 迭代变量 in 对象:
    语句块
```

```
if 表达式:
    continue
```

示例如下，如果 x 是“banana”，则退出本次循环，不执行后续的打印语句，然后继续执行下一轮的循环。

【文件 3.14】 demo21.py

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
print("End")
```

运行结果：

```
apple
cherry
End
```

可以看到，在第二次循环时，x 为“banana”，遇到 continue 语句，所以就跳出本次循环，不执行打印语句，没有输出“banana”。然后继续下一轮循环，x 继续等于“cherry”，然后打印输出“cherry”。

下面是 while 循环中 continue 语句的示例。

【文件 3.15】 demo22.py

```
i = 5
while i > 0:
    i -= 1
    print("i=", i)
    if i >= 3:
        print("我在 continue 之前，会执行")
        continue
        print("我在 continue 之后，不会执行")
```

运行结果：

```
i=4
我在 continue 之前，会执行
i=3
我在 continue 之前，会执行
i=2
i=1
i=0
```

可以看到，遇到 continue 语句后，循环体后续语句不再执行，进入下一轮循环，直到循环结束。

下面查看在嵌套循环中使用 continue 语句。

【文件 3.16】 demo23.py

```
color = ["red", "green", "blue"]
fruits = ["apple", "banana", "cherry"]

for x in color:
    for y in fruits:
        if y == "banana":
            continue
        print(x, y)
```

运行结果：

```
red apple
red cherry
green apple
green cherry
blue apple
blue cherry
```

可以看到，当 y 等于“banana”时，遇到 continue 语句，不打印输出，继续执行下一轮循环，y 等于“cherry”。

注意和 break 语句区分，代码如下。

【文件 3.17】 demo24.py

```
color = ["red", "green", "blue"]
fruits = ["apple", "banana", "cherry"]

for x in color:
    for y in fruits:
        if y == "banana":
            break
        print(x, y)
```

运行结果：

```
red apple
green apple
blue apple
```

3.4.3 pass 语句

pass 语句表示空操作，在执行时没有任何反应。使用 pass 语句可以保证格式完整和语义完整。

在编写一个程序时，执行语句部分思路还没有完成，或需后续再写，这时可以使用 pass 语句来占位，也可以当作一个标记，以后再继续完成这部分代码。例如：

```
def iplayPython():
    pass
```

pass 也常用于为复合语句编写一个空的主体，比如想要一个 while 语句的无限循环，每

次迭代时不需要任何操作，即可以使用 pass 语句实现。

```
while True:
    pass
```

以上只是理论上可以，实际写代码时最好不要这样写，因为执行代码块为 pass，也就是空，什么也不做，这时 Python 会进入死循环。要尽量避免程序进入死循环。

实训 3　功能选择

需求说明

根据实训 1 的功能菜单，依据用户输入的数字，打印输出相应的功能语句。循环重复实现此功能，若用户选择退出，输出提示语，需用户确认退出再结束程序。

训练要点

循环控制语句、选择控制语句。

实现思路

使用循环控制语句 while 控制多次打印功能菜单和提示用户输入。

使用选择控制语句根据用户的输入来打印相应的功能语句和控制用户退出。

解决方案及关键代码

```
while True:
    print('=' * 30)
    print('欢迎使用学生管理系统')
    print('1.添加学生信息')
    print('2.删除学生信息')
    print('3.修改学生信息')
    print('4.查询所有学生信息')
    print('5.组合学生信息')
    print('6.查询学生电话')
    print('0.退出系统')
    print('=' * 30)
    key =input("请输入功能对应的数字:")   #获取用户输入的序号
    if key =='1':                         #添加学生信息
        print('添加学生信息')
    elif key =='2':                       #删除学生信息
        print('删除学生信息')
    elif key =='3':                       #修改学生信息
        print('修改学生信息')
    elif key =='4':                       #查询所有学生信息
        print('查询所有学生信息')
    elif key =='5':                       #组合所有学生信息
        print('组合所有学生信息')
    elif key =='6':                       #查询电话
        print('查询电话')
    elif key =='0':
```

```
        quit_confirm = input('亲,真的要退出么?(Yes or No):').lower()
        if quit_confirm == 'yes':
            print("谢谢使用!")
            break                          #跳出循环
        elif quit_confirm == 'no':
            continue
        else:
            print('输入有误!')
```

小结

本章介绍了程序控制结构,从结构化程序设计的角度来看,程序有三种控制结构:顺序结构、选择结构和循环结构。若是在程序中没有选择语句或循环语句,系统则默认自上而下一行一行地执行,这类程序的结构就为顺序结构。控制语句在各种不同的语言中基本类似。如果遇到 if、else 等选择控制语句,则根据条件判断,进入不同的分支执行。若遇到 while、for 等循环语句,则根据条件,多次执行循环体。若在分支语句或循环体中遇到 break、continue 和 pass 跳转语句,则根据关键字不同执行不同的跳出循环体操作。控制语句在程序运行过程中控制程序的流程,并最终实现业务逻辑。

课后练习

1. 简述 Python 程序的三种控制结构。
2. 下面(　　)关键字是循环语句的。

   A. while　　B. when　　C. where　　D. what

3. 在 Python 中,遇到(　　)关键字需要结束循环。

   A. break　　B. continue　　C. pass　　D. end

4. 有如下程序段:

```
a,b,c = 70,50,30
if a>b:
    a=b
    b=c
    c=a
print(a,b,c)
```

程序的输出结果为(　　)。

A. 50 30 50　　B. 50 30 70　　C. 70 50 30　　D. 70 30 70

5. 有如下程序段:

```
k = 10
while k == 0:
```

```
    k -=1
    print(k)
```

下列说法正确的是(　　)。

A. while 循环执行 10 次　　B. 无限循环

C. 循环不执行　　D. while 循环执行 1 次

6. 下列程序可以正常结束的是(　　)。

A.

```
i=3
while(i>0):
    i+=1
```

B.

```
i=-3
while(i<0):
    i-=1
```

C.

```
i=3
while(i<=3):
    i-=1
```

D.

```
i=3
while(i>0):
    i-=1
```

7. 编程实现如下功能：输入层数 x,打印出如下效果的等腰三角形(图中示例 x=5)。

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

8. 编程打印 1～100 的所有偶数。