

Python语言及数据分析

王学军

wangxuejun@stdu.edu.cn

22-03

第2章 程序控制结构

- 2.1 布尔表达式
- 2.2 选择结构
- 2.3 条件表达式
- 2.4 选择结构程序举例
- 2.5 while循环
- 2.6 循环控制策略
- 2.7 for循环
- 2.8 循环中的break, continue和else
- 2.9 循环结构程序举例

第2章 程序控制结构

Python程序中的语句默认是按照书写顺序依次被执行的，这样的语句是**顺序结构**的。（prolog 非顺序结构）

- 在顺序结构中，各语句是按自上而下的顺序执行的，执行完上一条语句就自动执行下一条语句，语句之间的执行是不作任何判断的，无条件的。
- 有时候我们需要根据特定的情况，有选择地执行某些语句，这时我们就需要一种选择结构的语句。
- 有时候我们还可以在给定条件下重复执行某些语句，这时我们称这些语句是循环结构的。
- 有了**顺序、选择和循环这三种基本的结构**，我们就能够构建任意复杂的程序了。

第2章 程序控制结构

✓ 2.1 布尔表达式

□ 2.2 选择结构

□ 2.3 条件表达式

□ 2.4 选择结构程序举例

□ 2.5 while循环

□ 2.6 循环控制策略

□ 2.7 for循环

□ 2.8 循环中的break, continue和else

□ 2.9 循环结构程序举例

2.1 布尔表达式

选择结构和循环结构都会使用布尔表达式作为选择的条件和循环的条件。

- 布尔表达式是由关系运算符和逻辑运算符按一定语法规则组成的式子。
- 关系运算符有：<（小于）、<=（小于等于）、==（等于）、>（大于）、>=（大于等于）、!=（不等于）。
- 逻辑运算符有：and、or、not。
- 布尔表达式的值只有两个，True和False。
- 在Python中False、None、0、“”、’、()、[]、{}作为布尔表达式的时候，会被解释器看作假（False）。
- 其他的一切都被解释为真，包括特殊值True。

2.1 布尔表达式

True和 **False**属于布尔数据类型（**bool**），它们都是保留字。一个布尔变量可以代表**True**或**False**值中的一个。**bool**函数可以用来（和**list**、**str**以及**tuple**一样）转换其他值。

```
>>> type(True)
```

```
<class 'bool'>
```

```
>>> bool('Practice makes perfect.')
```

```
True
```

```
>>> bool('')
```

```
False
```

```
>>> print(bool(4))
```

```
True
```

2.1 布尔表达式

True和 **False**属于布尔数据类型（**bool**），它们都是保留字。一个布尔变量可以代表**True**或**False**值中的一个。**bool**函数可以用来（和**list**、**str**以及**tuple**一样）转换其他值。

```
>>> type(True)
```

```
<class 'bool'>
```

```
>>> bool('Practice makes perfect.')
```

```
True
```

```
>>> bool('')
```

```
False
```

```
>>> print(bool(4))
```

```
True
```

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ✓ 2.2 选择结构
- ☐ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ☐ 2.6 循环控制策略
- ☐ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.2 选择结构

选择结构通过判断某些特定条件是否满足来决定下一步执行哪些语句。Python有多种选择语句类型：

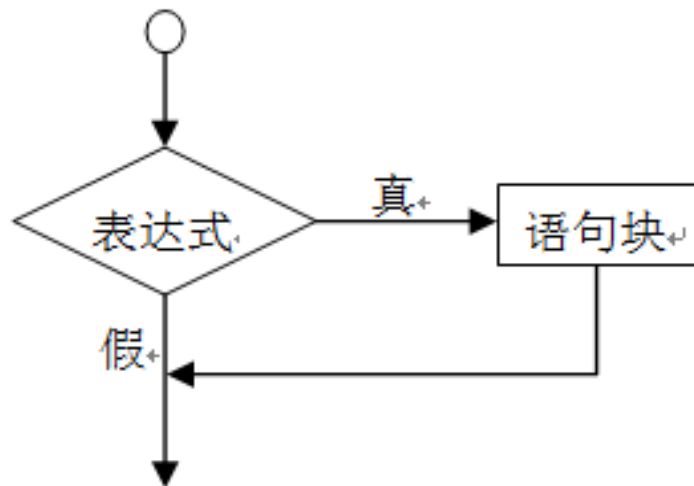
- 单向if语句
- 双向if-else语句
- 嵌套if语句
- 多向if-elif-else语句
- 条件表达式

2.2.1 单向if选择语句

if语句的语法形式如下所示：

if 布尔表达式：

语句块



- if语句的**语句块**只有当表达式的值为真，即非零时，才会执行；否则的话，程序就会直接跳过这个**语句块**，去执行紧跟在这个**语句块**之后的语句。
- 这里的**语句块**，既可以包含多条语句，也可以只有一条语句。当**语句块**由多条语句组成时，要有统一的缩进形式，相对于if向右至少缩进一个空格，否则就会出现逻辑错误，即语法检查没错，但是结果却非预期。

2.2.1 单向if选择语句

例2-1. 输入一个整数，如果这个数字大于100，那么就输出一行字符串；否则，直接退出程序。

```
>>> a = input('请输入一个整数: ') #取得一个字符串
```

请输入一个整数: 123

```
>>> a = int(a)                #将字符串转换为整数
```

```
>>> if a>100:
```

```
    print('输入的整数%d 大于100' %a) #a以10进制输出
```

输入的整数123 大于100

2.2.2 双向if-else选择语句

if-else语句是一种双选结构，根据表达式是真还是假来决定执行那些语句，它选择的不是做与不做的问題，而是在两种备选操作中选择哪一个操作的问题。

if-else语句的语法形式：

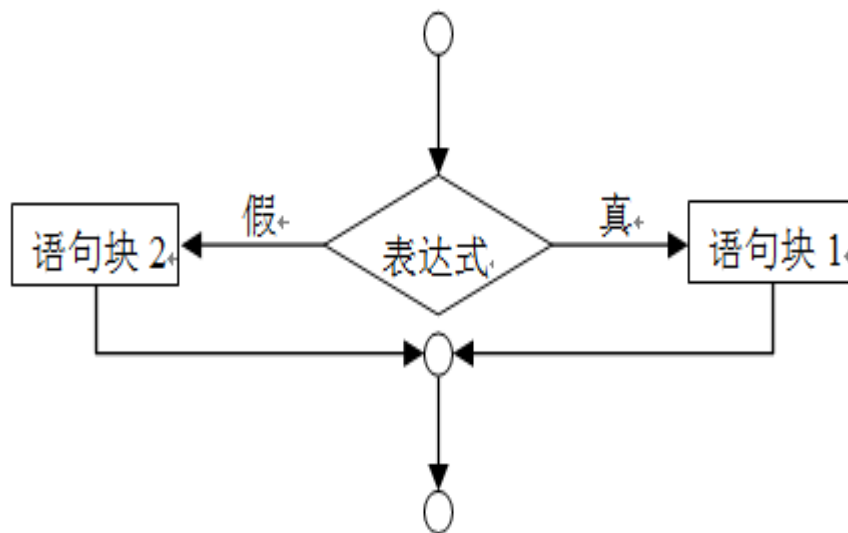
if 布尔表达式：

 语句块1

else：

 语句块2

if-else语句的流程示意图：



从流程示意图中可以看出：当表达式为真（即表达式的值为非零）时，执行语句块1；当表达式为假（即表达式的值为零）时，执行语句块2。

2.2.2 双向if-else选择语句

例2-3. 编写一个两位数减法的程序，程序随机产生两个两位数，然后向学生提问这两个数相减的结果是什么，在回答问题之后，程序会显示一条信息表明答案是否正确。（2-3.py）

```
import random
num1, num2= random.randint(10, 100), random.randint(10, 100)
if num1 < num2:
    num1, num2 = num2, num1
answer=int( input(str(num1) + '-' + str(num2) + '=' + ' ? ' ) )
if num1-num2==answer:
    print('你是正确的!')
else:
    print('你的答案是错误的!')
    print(str(num1), '-', str(num2), '=', str(num1- num2))
```

2.2.3 嵌套if选择语句和多向if-elif-else选择语句

- 将一个if语句放在另一个if语句中就形成了一个嵌套if语句。
- 如果需要在多组操作中选择一组执行，就会用到多选结构，就是if-elif-else语句。该语句可以利用一系列布尔表达式进行检查，并在某个表达式为真的情况下执行相应的代码。
- 虽然if-elif-else语句的备选操作较多，但是有且只有一组操作被执行，该语句的语法形式如下所示：

if 表达式1 :

 语句块1

elif 表达式2 :

 语句块2

.....

else :

 语句块n

2.2.3 嵌套if选择语句和多向if-elif-else选择语句

例2-4. 编写程序利用多分支选择结构将成绩从百分制变换到等级制。

(score_degree.py)

```
score=float(input('请输入一个分数: '))
```

```
if score>=90.0:
```

```
    grade='A'
```

```
elif score>=80.0:
```

```
    grade='B'
```

```
elif score>=70.0:
```

```
    grade='C'
```

```
elif score>=60.0:
```

```
    grade='D'
```

```
else:
```

```
    grade='F'
```

```
print(grade)
```

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ✓ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ☐ 2.6 循环控制策略
- ☐ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.3 条件表达式

有时候我们可能想给一个变量赋值，但又受一些条件的限制。例如下面的语句在x大于0时将1赋给y，在x小于等于0时将-1赋给y。

```
>>> x=2
>>> if x>0:
    y=1
    else:
    y=-1
>>> print(y)
1
```

在Python中，还可以使用条件表达式 `y=1 if x>0 else -1` 来获取同样的结果。

```
>>> x=2
>>> y=1 if x>0 else -1
>>> print(y)
1
```

2.3 条件表达式

条件表达式的语法结构如下所示：

表达式1 if 布尔表达式 else 表达式2

- 如果布尔表达式为真，那么这个条件表达式的结果就是表达式1；否则，这个结果就是表达式2。
- 若想将变量number1和number2中较大的值赋给max，可以使用下面的条件表达式简洁地完成。

max=number1 if number1>number2 else number2

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ☐ 2.3 条件表达式
- ✓ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ☐ 2.6 循环控制策略
- ☐ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.4 选择结构程序举例

例2-6. 开发一个玩彩票的程序，程序随机产生一个三位数的数字，然后提示用户输入一个三位数的数字，并根据以下规则判定用户是否赢得奖金。（练习）

- 1) 如果用户输入的数字和随机产生的数字完全相同（包括顺序），则奖金为3000美元。
- 2) 如果用户输入的数字和随机产生的数字有两位连着相同，则奖金为1000美元，对应一个三位数字 abc ，两位连着相同指的是 ab^* 或 *bc 。
- 3) 如果用户输入的数字和随机产生的数字有一位相同，则奖金为500美元。（2-6.py）

2.4 选择结构程序举例

(2-6.py)

```
import random
lottery = random.randint(100,999)
guess=eval(input("输入你想要的彩票号码: "))
lotteryD1,lotteryD2,lotteryD3=lottery//100,(lottery//10)%10, lottery%100
guessD1 , guessD2, guessD3= guess//100, (guess//10)%10, guess%100
print("开奖号码是: ",lottery)
if guess== lottery:
    print("号码完全相同: 奖金为3000美元")
elif (lotteryD1==guessD1 and lotteryD2==guessD2) or
    (lotteryD3==guessD3 and lotteryD2==guessD2):
    print("有两位号码连着相同: 奖金为2000美元")
elif len((set(str(lottery))&set(str(guess))))==1:
    print("有一号码相同: 奖金为500美元")
else:
    print("对不起, 这次没中奖! ")
```

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ☐ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ✓ 2.5 while循环
- ☐ 2.6 循环控制策略
- ☐ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.5 while循环

while语句用于在某条件下循环执行某段程序，以处理需要重复处理的任务。其基本形式为：

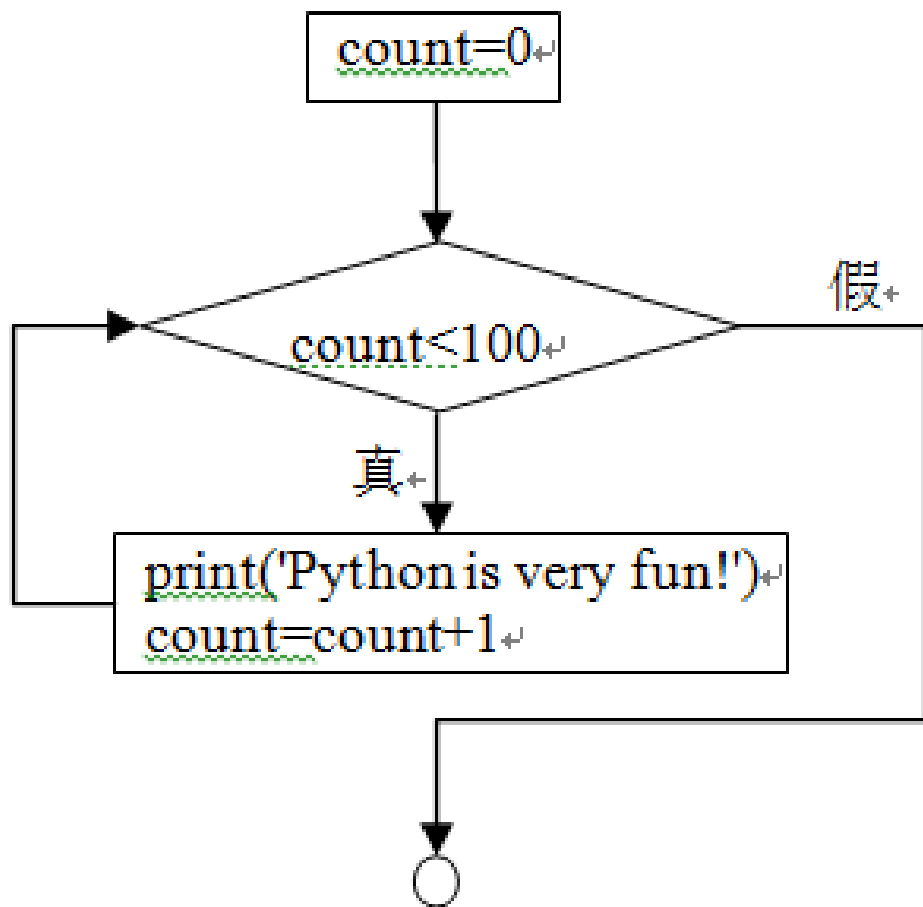
while 循环继续条件：

循环体

- 循环体可以是一个单一的语句或一组具有统一缩进的语句。
- 每个循环都包含一个循环继续条件，即控制循环执行的布尔表达式，每次都计算该布尔表达式的值，如果它的计算结果为真，则执行循环体；否则，终止整个循环并将程序控制权转移到while循环后的语句。
- while循环是一种条件控制循环，它是根据一个条件的真假来控制的。

2.5 while循环

显示Python is very fun!一百次的while循环的流程图如图3-2所示。



注意：要确保循环继续条件最终变成False以便结束循环。编写循环程序时，常见的程序设计错误是循环继续条件总是为True，循环变成无限循环。

2.5 while循环

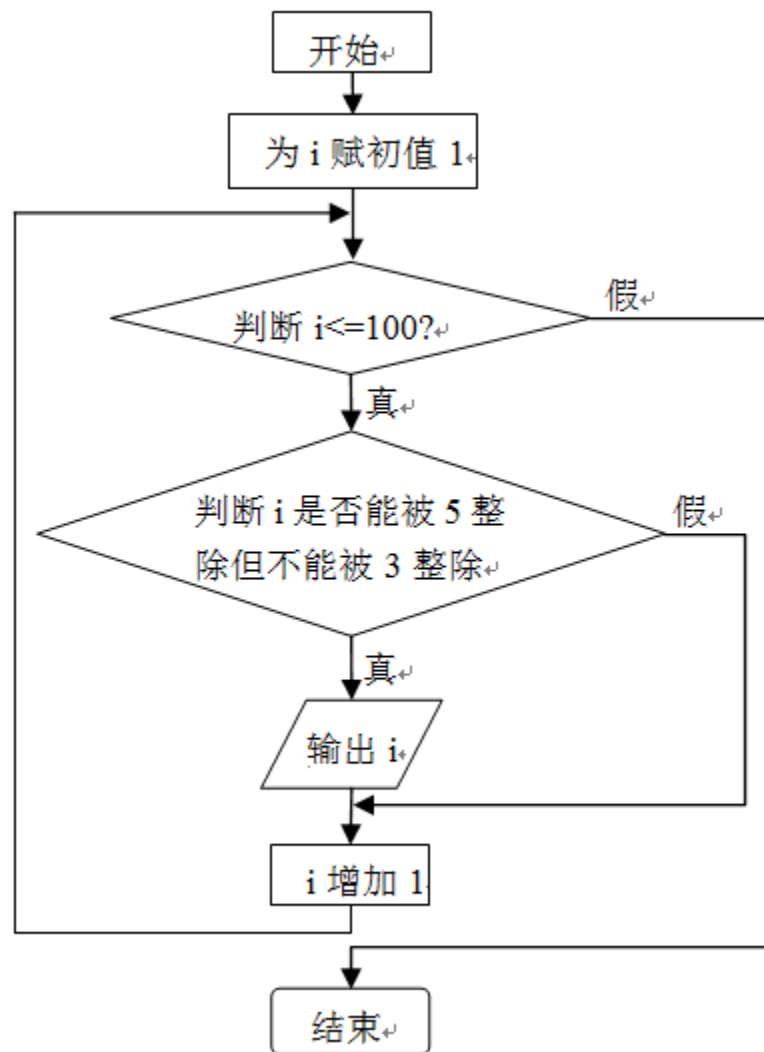
例2-8. 求1~100之间能被5整除,但不能同时被3整除的所有整数。

(2-8.py)

问题分析:

- (1) 本题需要对1~100范围内的所有数一一进行判断。
- (2) 本题的循环次数是100次。
- (3) 在每次循环过程中需要用if语句进行条件判断。

本题整除问题的框图如图所示:



2.5 while循环

例2-8. 求1~100之间能被5整除，但不能同时被3整除的所有整数。（2-8.py）

`i=1` #i既是循环变量，同时又是被判断的数

`print("1~100之间能被5整除，但不能同时被3整除的所有数是：")`

`while i<=100:`

`if i%5==0 and i%3!=0: #判断本次的i是否满足条件`

`print(i,end=' ') #打印满足条件的i`

`i=i+1 #每次循环i被加1`

2-8.py在IDLE中运行的结果如下：

1~100之间能被5整除，但不能同时被3整除的所有数是：

5 10 20 25 35 40 50 55 65 70 80 85 95 100

2.5 while循环

练习：打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位的十进制数，其各位数字立方和等于该数本身。

问题分析：

（1）“水仙花数”是一个三位的十进制数，因而本题需要对100 ~ 999范围内的每个数进行是否是“水仙花数”的判断。

（2）每次需要判断的数是有规律的，后一个数比前一个数多1，这样在判断完上一个数*i*后，使*i*加1就可以得到下一个数，因而变量*i*既是循环变量，同时也是被判断的数。

```
i = 100          #变量i赋初始值
print('所有的水仙花数是:', end='')
while i <= 999: #循环继续的条件
    c = i%10      #获得个位数
    b = i//10%10  #获得十位数
    # (//10 表示除以10向下取整)
    a = i//100    #获得百位数
    #判断是否是“水仙花数”
    if a**3+b**3+c**3==i:
        print(i,end=' ')
        #打印水仙花数
    i = i+1       #变量i增加1
```

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ☐ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ✓ 2.6 循环控制策略
- ☐ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.6 循环控制策略

要想编写一个能够正确工作的while循环，需要考虑以下三步：

- 第1步：确认需要循环的循环体语句，即确定重复执行的语句序列。
- 第2步：把循环体语句放在循环内。
- 第3步：编写循环继续条件，并添加合适的语句以控制循环能在有限步内结束，即能使循环继续条件的值变成False。

2.6.1 交互式循环

交互式循环是无限循环的一种，允许用户通过交互的方式重复循环体的执行，直到用户输入特定的值结束循环。

例2-12. 编写小学生100以内加法训练程序，并在学生结束测验后能报告正确答案的个数和测验所用的时间，并能让用户自己决定随时结束测验。（2-12.py）

```
import random
```

```
import time
```

```
correctCount=0          #记录正确答对数
```

```
count=0                 #记录回答的问题数
```

```
continueLoop='y'        #让用户来决定是否继续答题
```

```
startTime=time.time()   #记录开始时间
```

2.6.1 交互式循环

```
while continueLoop=='y':  
    number1=random.randint(0,50)  
    number2=random.randint(0,50)  
    answer=eval(input(str(number1)+'+'+str(number2)+'='+'?'))  
    if number1+number2==answer:  
        print('你的回答是正确的！')  
        correctCount+=1  
    else:  
        print('你的回答是错误的。')  
        print(number1,'+',number2,'=',number1+number2)  
    count+=1  
    continueLoop=input('输入y继续答题，输入n退出答题：')
```

2.6.1 交互式循环

```
endTime=time.time() #记录结束时间
```

```
testTime=int(endTime-startTime)
```

```
print("    正确率： %.2f%%\n测验用时： %d秒"    %  
      ((correctCount/count)*100, testTime))
```

2-12.py在IDLE中运行的结果如下：

2+36=?38

你的回答是正确的！

输入y继续答题，输入n退出答题： y

40+47=?87

你的回答是正确的！

...

2.6.2 哨兵式循环

另一个控制循环结束的技术是指派一个特殊的输入值，这个值称作哨兵值，它表明输入的结束。所谓哨兵式循环是指执行循环语句直到遇到哨兵值，循环体语句才终止执行的循环结构设计方法。

- 哨兵循环是求平均数的较好方案，思路如下：
 - 设定一个哨兵值作为循环终止的标志
 - 任何值都可以做哨兵，但要与实际数据有所区别

2.6.2 哨兵式循环

例2-13. 计算不确定人数的班级平均成绩。(StatisticalMeanValue.py)

```
total = 0
```

```
gradeCounter = 0 #记录输入的成绩个数
```

```
grade = int(input("输入一个成绩，若输入-1结束成绩输入："))
```

```
while grade != -1:
```

```
    total = total + grade
```

```
    gradeCounter = gradeCounter + 1
```

```
    grade = int(input("输入一个成绩，若输入-1结束成绩输入："))
```

```
if gradeCounter != 0:
```

```
    average = total / gradeCounter
```

```
    print("平均分是： %.2f"%(average))
```

```
else:
```

```
    print('没有录入学生成绩')
```

2.6.3 文件式循环

例2-13中，如果要输入的数据很多，那么从键盘输入所有数据将是一件非常麻烦的事。我们可以事先将数据录入到文件中，然后将这个文件作为程序的输入，避免人工输入的麻烦。例如，我们可以把数据存储在一个文本文件（命名为input.txt）里，并使用下面的命令来运行这个程序，可以实现与人工输入相同的效果：

python StatisticalMeanValue.py < input.txt

- 这个命令称作输入重定向。用户不再需要程序运行时从键盘录入数据，而是从文件input.txt中获取输入数据。
- 同样地，输出重定向是把程序运行结果输出到一个文件里而不是输出到屏幕上。输出重定向的命令为：

python StatisticalMeanValue.py > output.txt

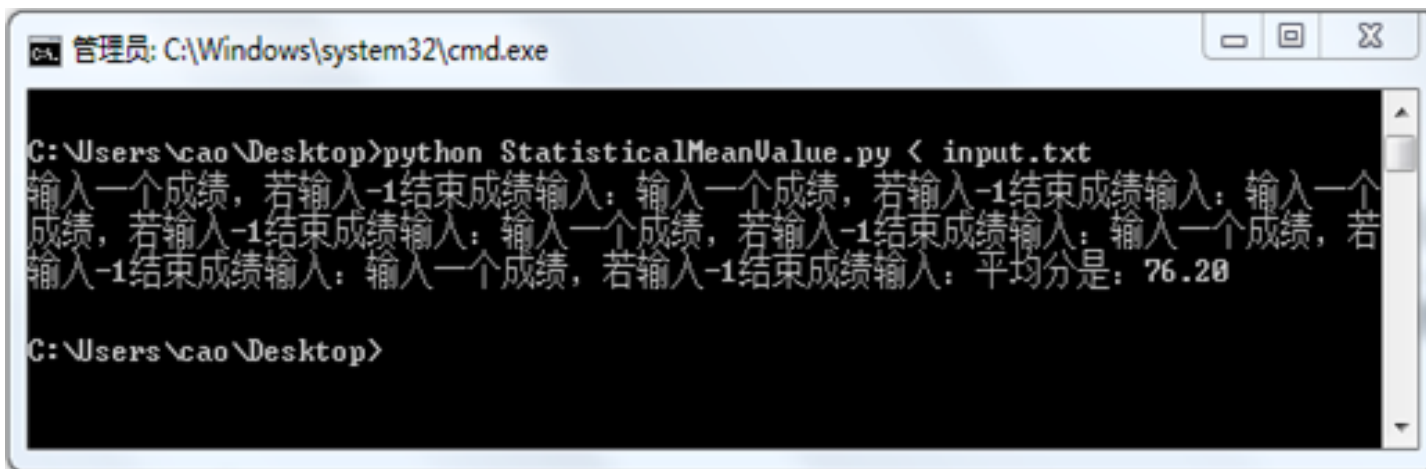
3.2.3 文件式循环

- 同一条命令里可以同时使用输入重定向与输出重定向。例如下面这个命令从input.txt中读取输入数据，然后把输出数据写入文件output.txt中。

`python StatisticalMeanValue.py < input.txt > output.txt`

假设input.txt这个文件包含下面的数字，每行一个：

45
80
90
98
68
-1



The screenshot shows a Windows command prompt window titled "管理员: C:\Windows\system32\cmd.exe". The command entered is `C:\Users\cao\Desktop>python StatisticalMeanValue.py < input.txt`. The output displayed is a repeated prompt "输入一个成绩, 若输入-1结束成绩输入:" followed by the calculation "平均分是: 76.28".

在命令行窗口中，StatisticalMeanValue.py从文件input.txt中获取输入数据执行的结果：

3.2.3 文件式循环

例2-13的程序实现可改写为更为简洁的文件读取的方式来实现：

```
FileName=input('输入数据所在的文件的文件名：')
```

```
infile=open(FileName,'r') #打开文件
```

```
sum=0
```

```
count=0
```

```
line=infile.readline() #按行读取数据
```

```
while line!='-1':
```

```
    sum=sum+eval(line)
```

```
    count=count+1
```

```
    line=infile.readline()
```

```
if count!= 0:
```

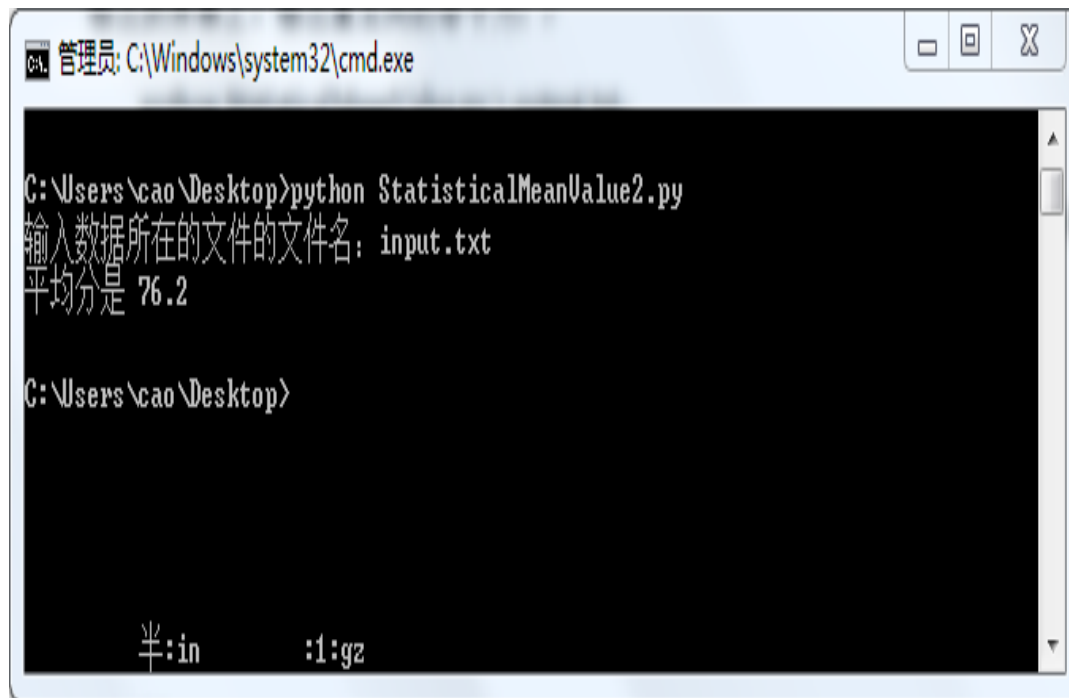
```
    average = float(sum) / count
```

```
    print("平均分是", average)
```

```
else:
```

```
    print('没有录入学生成绩')
```

```
infile.close() #关闭文件
```



The screenshot shows a Windows command prompt window titled "管理员: C:\Windows\system32\cmd.exe". The prompt is at "C:\Users\cao\Desktop>". The user has entered "python StatisticalMeanValue2.py". The script has prompted for the filename: "输入数据所在的文件的文件名: input.txt". The user has entered "input.txt". The script has calculated and displayed the average: "平均分是 76.2". The prompt is now at "C:\Users\cao\Desktop>".

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ☐ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ☐ 2.6 循环控制策略
- ✓ 2.7 for循环
- ☐ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.7.1 for循环的基本用法

for循环是一种遍历型的循环，因为它会依次对某个序列中全体元素进行遍历，遍历完所有元素之后便终止循环。

for循环的一般格式如下：

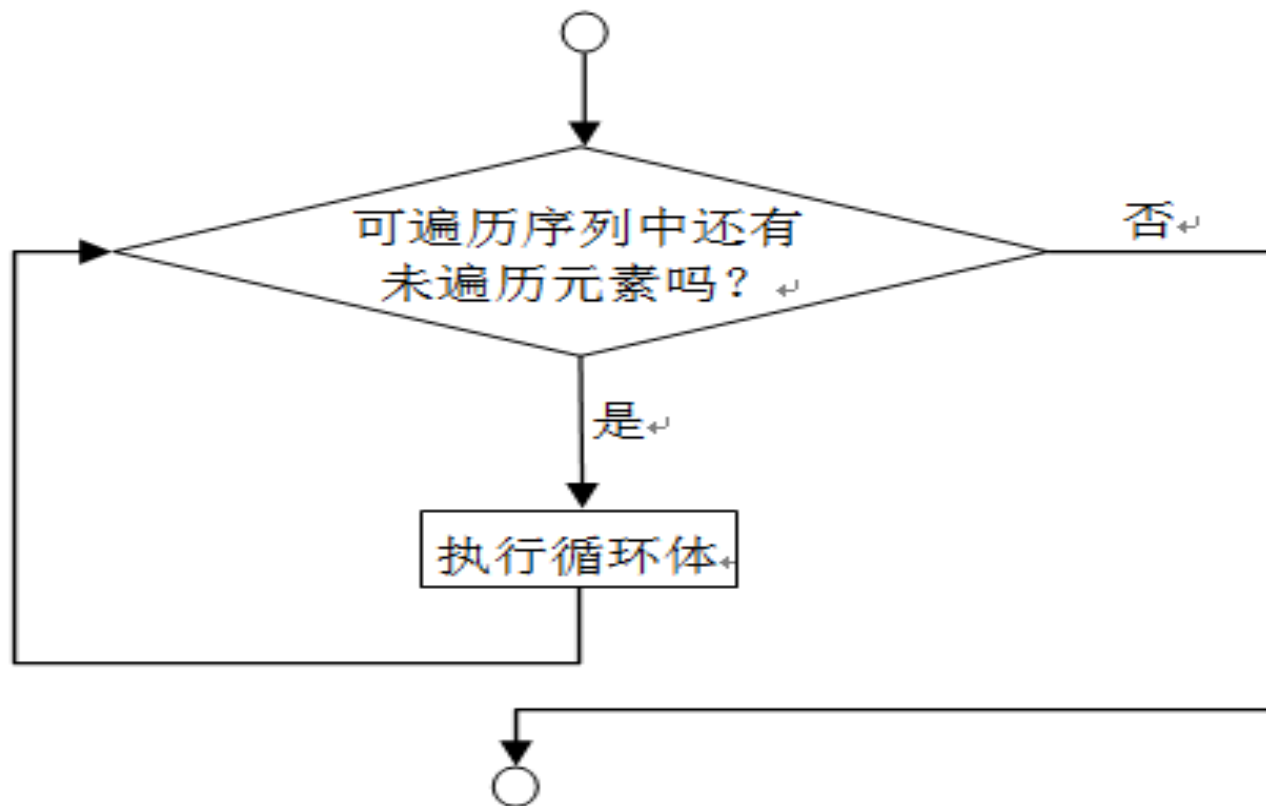
for 控制变量 in 可遍历序列：

循环体

- 这里的关键字in是for循环的组成部分，而非运算符in。
- “可遍历序列”里保存了多个元素，且这些元素按照一个接一个的方式存储。
- “可遍历序列”被遍历处理，每次循环时，都会将“控制变量”设置为“可遍历序列”的当前元素，然后执行循环体。当“可遍历序列”中的元素被遍历一遍后，即没有元素可供遍历时，退出循环。

2.7.1 for循环的基本用法

for语句的示意图如下所示：



“可遍历序列”可以是列表、元组、字典、集合、文件，甚至可以是自定义类或者函数

2.7.1 for循环的基本用法

for循环可用于迭代容器对象中的元素，这些对象可以是列表、元组、字典、集合、文件，甚至可以是自定义类或者函数，例如：

（2）作用于元组

例2-15. 遍历元组。（2-15.py）

```
test_tuple = [("a",1),("b",2),("c",3),("d",4)]#元组构成的列表
print("准备遍历的元组列表:", test_tuple)
print('遍历列表中的每一个元组:',end='')#表示不换行
for i, j in test_tuple:
    print(i,j,end=';')
```

2-15.py在IDLE中运行的结果如下：

```
准备遍历的元组列表: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]
遍历列表中的每一个元组:a 1;b 2;c 3;d 4;
```

2.7.1 for循环的基本用法

(3) 作用于字符串

例2-16. 遍历输出字符串中的汉字，遇到标点符号换行输出。(2-16.py)

```
import string
str1 = "大梦谁先觉?平生我自知,草堂春睡足,窗外日迟迟."
for i in str1:
    if i not in string.punctuation:#表示字符是否属于标点
        print(i,end='')
    else:
        print(' ')#换行
```

2-16.py在IDLE中运行的结果如下：

```
大梦谁先觉
平生我自知
草堂春睡足
窗外日迟迟
```

2.7.1 for循环的基本用法

(4) 作用于字典

例2-17. 遍历输出字典元素。(2-17.py)

```
person={'姓名':'李明','年龄':'26','籍贯':'北京'}
```

item()方法把字典中每对key和value组成一个元组，并把这些元组放在列表中返回。

```
for key,value in person.items():
```

```
    print('key=',key,', value=',value)
```

#只有一个控制变量时，返回的是每

```
for x in person.items():
```

```
    print(x)
```

```
for x in person:           #不使用items()
```

```
    print(x)
```

2-17.py在IDLE中运行的结果如下：

```
key= 姓名 , value= 李明
```

```
key= 年龄 , value= 26
```

```
key= 籍贯 , value= 北京
```

```
('姓名','李明')
```

```
('年龄','26')
```

```
('籍贯','北京')
```

```
姓名
```

```
年龄
```

```
籍贯
```

2.7.1 for循环的基本用法

(5) 作用于集合

例2-18. 遍历输出集合元素。(2-18.py)

```
weekdays = {'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'}  
# for循环在遍历set时，元素的顺序和list的顺序很可能是不同的  
for d in weekdays:  
    print(d,end=' ')
```

2-18.py在IDLE中运行的结果如下：

THU TUE MON FRI WED SAT SUN

2.7.1 for循环的基本用法

（6）作用于文件

例2-19. for循环遍历文件，打印文件的每一行。（2-19.py）

文件1.txt存有两行文字：

向晚意不适，驱车登古原。

夕阳无限好，只是近黄昏。

```
fd = open('D:\\Python\\1.txt')
```

```
for line in fd:
```

```
    print(line,end=' ')
```

2-19.py在IDLE中运行的结果如下：

```
===== RESTART: D:/Python/2-19.py =====
```

向晚意不适，驱车登古原。

夕阳无限好，只是近黄昏。

2.7.2 for循环与range()函数的结合使用

range()函数用来生成整数数字数列，其语法格式如下：

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

参数说明：

- **start**: 计数从start 开始。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。例如：range(0, 5)是[0, 1, 2, 3, 4]没有5，也就是说range(a,b)函数返回连续整数a、a+1、...、b-2和b-1的序列；
- **step**: 步长，默认为1。例如：range(0, 5)等价于range(0, 5, 1)。

注意：

range(5)的返回值类型是range类型，如果想得到一个列表，使用list(range(5))得到的就是一个列表[0, 1, 2, 3, 4]。如果想得到一个元组，使用tuple(range(5))得到的就是一个元组(0, 1, 2, 3, 4)。

2.7.2 for循环与range()函数的结合使用

例2-20. 输出斐波那契数列的前n项。斐波那契数列以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、13、21、34、.....，可通过递归的方法定义：

$F(1)=1$ ， $F(2)=1$ ， $F(n)=F(n-1)+F(n-2)$ ($n \geq 3$ ， $n \in \mathbb{N}^*$)。 (2-20.py)

问题分析：从斐波那契数列可以看出，从第三项起每一项的数值都是前两项（可分别称为倒数第二项、倒数第一项）的数值之和，斐波那契数列每增加一项，对下一个新的项来说，刚生成的项为倒数第一项，其前面的项为倒数第二项。

2.7.2 for循环与range()函数的结合使用

例2-20. 输出斐波那契数列的前n项。斐波那契数列以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、13、21、34、.....，可通过递归的方法定义：

$F(1)=1$, $F(2)=1$, $F(n)=F(n-1)+F(n-2)$ ($n \geq 3$, $n \in \mathbb{N}^*$)。 (2-20.py)

a, b=1,1

n=int(input('请输入斐波那契数列的项数(>2的整数): '))

print('前%d项斐波那契数列为: '%(n),end='')

print(a,b,end='')

for k in range(3,n+1):

 c=a+b

 print(c,end='')

 a=b

 b=c

2-20.py在IDLE中运行的结果如下：

请输入斐波那契数列的项数(>2的整数): 8
前8项斐波那契数列为: 1 1 2 3 5 8 13 21

第2章 程序控制结构

- ☐ 2.1 布尔表达式
- ☐ 2.2 选择结构
- ☐ 2.3 条件表达式
- ☐ 2.4 选择结构程序举例
- ☐ 2.5 while循环
- ☐ 2.6 循环控制策略
- ☐ 2.7 for循环
- ✓ 2.8 循环中的break, continue和else
- ☐ 2.9 循环结构程序举例

2.8 循环中的break, continue和else

break语句和**continue**语句提供了另一种控制循环的方式。

- **break**语句用来终止循环语句，即循环条件没有False或者序列还没被完全遍历完，也会停止执行循环语句。如果使用嵌套循环，**break**语句将停止执行最深层的循环，并开始执行下一行代码。
- **continue**语句终止当前迭代而进行循环的下一一次迭代。
- 循环语句也可以带有**else**子句，**else**子句在序列遍历结束（**for**语句）或循环条件为假（**while**语句）时执行，但循环被**break**终止时不执行。

2.8.3 循环语句的else 子句

带有else子句的while循环语句的完整形式如下所示：

while 循环继续条件：

 循环体

else：

 语句体

带有else子句的for语句的完整形式：

for 控制变量in 可遍历序列：

 循环体

else：

 语句体

例. for循环正常结束执行else子句。

```
for i in range(2, 11):
```

```
    print(i)
```

```
else:
```

```
    print('for statement is over.')
```

程序运行的结果是：

2,3,4,5,6,7,8,9,10,

for statement is over.（序列遍历结束后执行）

第2章 程序控制结构

- 2.1 布尔表达式
- 2.2 选择结构
- 2.3 条件表达式
- 2.4 选择结构程序举例
- 2.5 while循环
- 2.6 循环控制策略
- 2.7 for循环
- 2.8 循环中的break, continue和else
- ✓ 2.9 循环结构程序举例

2.9 循环结构程序举例

练习：编写程序，输出由1、2、3、4这四个数字组成的各位数字互不相同三位数及总个数。（3-23.py）

```
digits=[1,2,3,4]
counter=0
for i in digits:
    for j in digits:
        if(i!=j):
            for k in digits:
                if(i!=j and j!=k and i!=k):
                    print(i*100+j*10+k,end=' ')
                    counter+=1
print("\n1、2、3、4一共组成%d个互不相同的三位数"%(counter))
```

2-29.py在IDLE中运行的结果如下：

```
123 124 132 134 142 143
213 214 231 234 241 243
312 314 321 324 341 342
412 413 421 423 431 432
1、2、3、4一共组成24个
互不相同的三位数
```

第2章 程序控制结构

THE END

作业：发布到超星平台