第3章 程序控制结构

3.1 程序控制结构简介

程序控制结构主要包括顺序结构、选择结构、循环结构。

顺序结构是按语句的书写顺序执行的程序结构。

选择结构是根据特定的条件决定执行哪个语句的程序结构，常用 if 语句。

循环结构是在满足特定的条件时重复执行某些语句的程序结构，常用 for 和 while 语句。

顺序结构流程如图 3-1 所示。

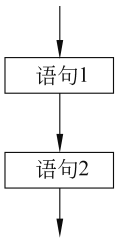

图 3-1 顺序结构流程

例如，下面的代码按顺序依次执行下来，先把 a、b 分别赋值为 1、3，然后交换这两个变量的值，再输出它们的值。

```
a,b=1,3
a,b=b,a
print(a,b)
```

3.2 选 择 结 构

例 3.2.1 求两者中的大者

输入 2 个整数 a、b，找出其中的大者并输出。

解析：

思路 1：先假设第一个数大，然后与后一个数比较，若后一个数大，则大者为后一个。可用单分支 if 语句实现，具体代码如下。

```
a,b=map(int,input().split())
c=a                #假设第一个数大,存放在假设的大者 c 中
if b>c:            #若第二个数大于 c,则把 c 修改为第二个数
    c=b
print(c)
```

运行结果如下。

```
1 3↵
3
```


思路 2：直接比较两个数，若前一个数大，则把它作为结果，否则结果为后一个数。可用双分支 if 语句实现，具体代码如下。

```
a,b=map(int,input().split())
if a>=b:                #若第一个数不小于第二个数,则大者为第一个数
    c=a
else:                   #若第二个数大于第一个数,则大者为第二个数
    c=b
print(c)
```

本题上述两种思路的代码中分别使用了单分支、双分支 if 语句。

上述代码是为了引入 if 语句，而在线编程时或程序设计竞赛时，可直接调用内置函数 max(a, b)求得 a、b 中的大者。

1. 基本的 if 语句

基本的 if 语句格式如下。

```
if 条件：语句 1
[ else：语句 2 ]
```

注意，if 的条件及 else 之后都需要添加冒号“:”。若语句 1 或语句 2 包含多条语句，可以在一行上把多条语句用分号间隔，也可以把多条语句放到冒号的后续几行(注意保持缩进量的一致)。描述语法时，[ ]表示[ ]中的内容是可选项，即 if 语句可为单分支 if 语句，格式如下。

```
if 条件：语句 1
```

此 if 语句在满足条件时执行语句 1，否则不执行任何语句，其流程如图 3-2 所示。

或为双分支 if 语句，格式如下。

```
if 条件：
    语句 1
else：
    语句 2
```

此 if 语句在满足条件时执行语句 1，否则执行语句 2，其流程如图 3-3 所示。

可见，if 语句可带 else 子句(双分支选择结构)，也可以不带 else 子句(单分支选择结构)。

if 语句的条件一般是一个为 True 或 False 的表达式，否则一切 0 值转换为 False，一切非 0 值转换为 True。

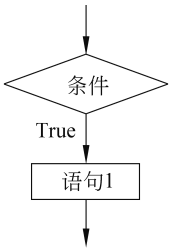

图 3-2　单分支 if 语句流程

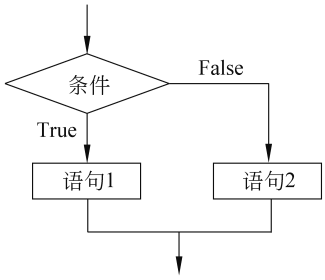

图 3-3　双分支 if 语句流程

语句 1 和语句 2 可以是一条语句，也可以是多条语句（在一行上写时以分号间隔，分行写时注意保持缩进量的一致）。例如：

```
x=int(input())
if x:                                    #x 相当于 x!=0
    print("x is non-zero")
else:
    printf("x is zero\n")
if x==1:
    a=1                                  #缩进量与下一句一致
    b=2                                  #缩进量与上一句一致
else:
    a=-1                                 #缩进量与下一句一致
    b=-2                                 #缩进量与上一句一致
```

条件表达式类似于双分支选择结构，区别在于前者是表达式，后者是语句。条件表达式形式如下。

```
表达式 1 if 条件 else 表达式 2
```

当“条件”为 True 时，取“表达式 1”的值为条件表达式的值，否则取“表达式 2”的值为条件表达式的值。例如：

```
>>>x=1
>>>y=3
>>>x if x>=y else y          #取 x、y 中的大者作为表达式的值
3
```

2. 嵌套的 if 语句

嵌套的 if 语句是指在 if 语句中又使用 if 语句。if 语句可以嵌套在 if 子句中，也可以嵌套在 else 子句中。例如：

```
x=int(input())
```

```
if x==0:
    print("zero")
else:
    if x>0:                    #if 语句嵌套在 else 子句中
        print("positive")
    else:
        print("negative")
```

若 if 嵌套在 else 子句中,则可以缩写为 elif 子句,上述代码可缩写如下。

```
x=int(input())
if x==0:
    print("zero")
elif x>0:                      #if 语句嵌套在 else 子句中,缩写为 elif
    print("positive")
else:
    print("negative")
```

3. if 选择结构示例

例 3.2.2　求三者的最大值

输入 3 个整数,找出其中最大的一个并显示出来。

解析:

思路:假设第一个数最大并放到结果 d 中,若后面的数大于 d,则把 d 变为该数。可用单分支语句实现,代码如下。

```
a,b,c=map(int,input().split())
d=a                            #假设第一数最大,存放在 d 中
if b>d:                        #若第二数比假设的最大数 d 更大,则把 d 改为该数
    d=b
if c>d:                        #若第三数比假设的最大数 d 更大,则把 d 改为该数
    d=c
print(d)
```

运行结果如下。

```
1 3 5↵
5
```

这种思路比较简单,而且容易扩展到多个数的情况(结合循环结构)。读者可以思考本题还有哪些实现方法,并自行编写代码实现。

在线编程时或程序设计竞赛时,可直接调用内置函数 max(a, b, c)求得 a、b、c 中的大者。

例 3.2.3　三数排序

输入 3 个整数,然后按从大到小的顺序把它们显示出来。

这个问题该如何实现呢？仔细思考，可以想到多种方法。下面给出两种方法，分别用到选择排序和冒泡排序的思想。

解析：

思路 1：采用选出当前最大者放到当前最前面位置（选择排序）的思想。具体代码如下。

```
a, b, c=map(int, input().split())
if a<b:                  #若第一个数比第二个数更小，则交换
    a, b=b, a
if a<c:                  #若第一个数比第三个数更小，则交换，至此最大数放在第一个位置
    a, c=c, a
if b<c:                  #若第二个数比第三个数更小，则交换，至此第二大数放在第二个位置
    b, c=c, b
print(a, b, c)
```

运行结果如下。

```
1 3 5↵
5 3 1
```

思路 2：采用把当前最小者放到当前最后面位置（冒泡排序）的思想。具体代码如下。

```
a, b, c=map(int, input().split())
if a<b:                  #若第一个数比第二个数更小，则交换
    a, b=b, a
if b<c:                  #若第二个数比第三个数更小，则交换，至此最小数放在第三个位置
    b, c=c, b
if a<b:                  #若第一个数比第二个数更小，则交换，至此第二小数放在第二个位置
    a, b=b, a
print(a, b, c)
```

例 3.2.4　成绩转换

百分制成绩转换为五级计分制时，90～100 分以上等级为 A，80～89 分等级为 B，70～79 分等级为 C，60～69 分等级为 D，0～59 分等级为 E。输入一个百分制成绩，请输出等级。

解析：

本例是多分支结构，可以用嵌套 if 语句实现，这里使用把 if 语句嵌套在 else 子句中，从而缩写为 elif 的方法。具体代码如下。

```
score=int(input())
if score>=90:                #若不小于 90，则为 A
    rank='A'
elif score>=80:              #若小于 90，但不小于 80，则为 B
    rank='B'
elif score>=70:              #若小于 80，但不小于 70，则为 C
```

```
    rank='C'
elif score>=60:                          #若小于 70,但不小于 60,则为 D
    rank='D'
else:                                    #若小于 60,则为 E
    rank='E'
print(rank)
```

运行结果如下。

```
85↵
B
```

因 if 和 elif 之后的条件是逐个判断下来的,故在判断第一个 elif 中的条件“score>=80”时,已经不满足 if 中的“score>=90”条件,而在判断第二个 elif 中的条件“score>=70”时,已经不满足“score>=80”条件。其他的 elif 和 else 中满足的条件可类似理解。

例 3.2.5　求某月的天数

输入年份 year、月份 month,判断该月的天数。

已知大月有 31 天,小月有 30 天,2 月有 28 天或 29 天(闰年)。大月共 7 个月,即 1、3、5、7、8、10、12 月;小月共 4 个月,即 4、6、9、11 月。

解析:

本题可用 if 语句根据不同的情况(大月、小月、2 月)得到该月的天数。具体代码如下。

```
year,month=map(int,input().split())
if month==2:                                              #若为 2 月
    if year%4==0 and year%100!=0 or year%400==0:          #若是闰年,则有 29 天
        days=29
    else:                                                 #若是非闰年,则有 28 天
        days=28
elif month==4 or month==6 or month==9 or month==11:       #若为小月,则有 30 天
    days=30
else:                                                     #若为大月,则有 31 天
    days=31
print(days)
```

运行结果如下。

```
2021 1↵
31
```

3.3 循环结构

3.3.1 引例

例 3.3.1 求 n 个整数中的最大值

首先在第一行输入一个整数 n，然后在第二行输入 n 个整数，请输出这 n 个整数中的最大值。

解析：

在例 3.2.2 中，使用两条类似的单分支 if 语句可求得三个数中的最大值。现需求 n 个数中的最大值，若程序运行前已知 n 的值，则也可以写 $n-1$ 个单分支语句完成，但 n 是一个变量，在程序运行时输入后才能确定其值，因此无法在程序运行前写好 $n-1$ 个 if 语句。因 $n-1$ 条类似的 if 语句重复在求两者中的大者，故可写成一条 if 语句并使其执行 $n-1$ 次。这可以使用循环结构完成。循环结构中的 for 语句常用于实现次数固定且重复执行的要求。本例的具体代码如下。

```
n=int(input())
a=list(map(int,input().split()))          #输入多个整数并存放到列表 a 中
maxVal=a[0]                          #假设第一个数最大,存放在假设的最大数变量 maxVal 中
#从第二个数开始,maxVal 逐个与每个当前数比较,若小于当前数,则改为当前数
for i in range(1,n):
    if a[i]>maxVal:
        maxVal=a[i]
print(maxVal)
```

运行结果如下。

```
10↵
11 5 21 54 77 2 45 43 87 9↵
87
```

上面的代码中，for 语句中循环变量 i 从 1 到 $n-1$ 进行循环，共执行 $n-1$ 次循环体（每次循环比较 $a[i]$和 maxVal，把大者保存在 maxVal 中）。实际上，在 Python 中可以直接用内置函数 max 或 min 求列表等可迭代对象中的最大值或最小值。具体代码如下。

```
n=int(input())
a=list(map(int,input().split()))          #输入多个整数存放到列表 a 中
print(max(a))                             #直接用 max 函数求列表 a 中的最大值
```

因内置函数 max 也可直接求得映射对象中的最大值，故“a=list(map(int,input().split()))”可改写为“a=map(int,input().split())”。

3.3.2 for 语句与 while 语句

1. for 语句

for 循环语句的基本格式如下。

```
for 循环变量 in 可迭代对象:
    循环体
[else: 其他语句]
```

其中,for、in 和 else 是关键字,else 子句可选。

for 后的循环变量也可称为迭代变量或迭代项。当还可以从可迭代对象中取得新的迭代项时则执行循环体,否则,若有 else 子句,则执行该子句后再结束循环,否则直接结束循环。for 循环流程如图 3-4 所示。

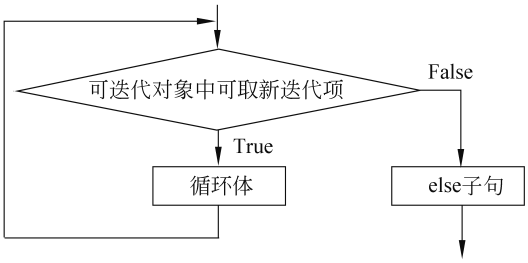

图 3-4 for 循环流程

注意,执行 else 子句时,迭代项仍处于可迭代对象中,其值为可迭代对象中最后一项的值。例如:

```
for i in range(3):
    print(i * i)
else:
    print(i)
```

运行结果如下。

```
0
1
4
2
```

一般而言,若循环中未使用 break 语句,则该 for 循环无需带有 else 子句。

若可迭代对象一开始为空,则迭代变量不会被创建。此时,若在 else 子句中使用迭代变量,则将产生错误。例如:

```
for i in range(12,9):            #range(12,9)产生的可迭代对象为空
    print(i)
else:
    print(i)
```

运行情况如下(程序文件名是 test.py)。

```
Traceback (most recent call last):
  File "D:/Python/test.py", line 4, in <module>
    print(i)
NameError: name 'i' is not defined
```

另外,在循环体中更改迭代项不会影响循环的执行次数,因为下次循环时迭代项自动取可迭代对象中的下一项。例如:

```
for i in range(1,4):
    i *=3                        #改变迭代变量不影响循环执行次数
    print(i)
```

运行结果如下。

```
3
6
9
```

2. while 语句

while 循环语句的格式如下。

```
while 循环条件:
    循环体
[else: 其他语句]
```

其中,while 和 else 是关键字,else 子句可选。while 语句在满足循环条件时反复执行循环体,否则,若有 else 子句,则执行该子句后再结束循环,否则直接结束循环。一般而言,若 while 循环中未使用 break 语句,则该 while 语句无须带有 else 子句。while 循环流程如图 3-5 所示。

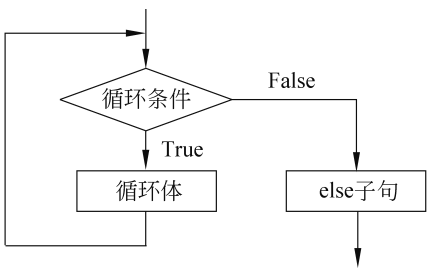

图 3-5　while 循环流程

循环体中一般需要有改变循环变量使循环趋于结束的语句或跳出循环的语句,以避免死循环。若循环体中有多条语句,则它们的缩进量需相同。另外,循环条件后需添加一个冒号。

3. for 语句与 while 语句示例

例 3.3.2　求和

输入一个正整数 n,求出由 1 加至 n 的总和。

解析:

本题解法较多,例如,可采用循环逐项累加,可使用等差数列求和公式,可调用内置函数 sum 求解。

方法 1:使用 for 循环。

具体代码如下。

```
n=int(input())
#使用 for 语句
s=0
for i in range(1,n+1):
    s+=i
print(s)
```

运行结果如下。

```
10↵
55
```

range(1,n+1)产生包含在闭区间[1, n]中各个整数的数列。

方法 2:使用 while 循环。

具体代码如下。

```
n=int(input())
s,i=0,1                          #累加单元清 0,循环变量置 1
while i<=n:                      #当 i<=n 时进行循环
    s+=i                         #将 i 加到 s 中
    i+=1                         #改变循环变量,使循环趋于结束
print(s)
```

上述代码中,循环变量 i 从 1 开始循环,当满足 $i<=n$ 条件时,反复执行循环体:将 i 逐个加到累加单元 s 中,并使 i 增 1;当 i 不断增加使得 $i<=n$ 条件不成立,即 $i>n$ 时,结束循环。

方法 3:使用等差数列求和公式。

具体代码如下。

```
n=int(input())
print(n*(n+1)//2)
```

方法 4:使用内置函数 sum 求 range(1, n+1)之和。

具体代码如下。

```
n=int(input())
print(sum(range(1,n+1)))
```

sum 函数可以直接求得数值型可迭代对象中的所有元素之和。

4. 列表产生式

for 循环可用在列表产生式(也称列表生成式、列表推导式)中。创建列表产生式的方式如下。

```
[表达式 for 迭代项 in 可迭代对象 if 条件 ]
```

此产生式在列表界定符[ ]中使用 for 循环和 if 条件(若不需要则省略)。列表产生式中的表达式通常与迭代项相关,可迭代对象可以是内置函数 range 产生的数列,也可以是列表、字符串和元组等序列及集合等对象。

例如:

```
>>>s=[i for i in range(10)]          #可迭代对象由 range 创建,产生 0～9 构成的列表
>>>s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>s=[i for i in range(1,10,2)]      #创建 1～9 中的奇数构成的列表
>>>s
[1, 3, 5, 7, 9]
>>>s=[i for i in range(10,-1,-1)] #创建由 10～0 构成的列表
>>>s
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>>s=[i for i in range(10) if i%2==0]   #带 if 条件的列表产生式
>>>s
[0, 2, 4, 6, 8]
>>>s=[i*i for i in [1,2,3,4,5]]    #可迭代对象为列表,表达式为迭代变量的平方
>>>s
[1, 4, 9, 16, 25]
>>>s=[it for it in "abcdef"]         #可迭代对象为字符串
>>>s
['a', 'b', 'c', 'd', 'e', 'f']
>>>s=[it for it in (3,6,9,12)]       #可迭代对象为元组
>>>s
[3, 6, 9, 12]
>>>s=[it for it in {2,4,6,8}]        #可迭代对象为集合
>>>s
[8, 2, 4, 6]
```

列表产生式中的表达式还可以是一个条件表达式。例如:

```
>>>s=[i if i%2==1 else 1/i for i in range(1,10)]
>>>s
[1, 0.5, 3, 0.25, 5, 0.16666666666666666, 7, 0.125, 9]
```

其中,"i if i%2==1 else 1/i"是一个条件表达式,若 i 为奇数,则取其本身,否则取其倒数。

若将列表产生式中的列表界定符[ ]改为集合界定符{ },则为集合产生式。例如:

```
>>>s={i for i in range(1,10) if i%2==1}          #集合产生式
>>>s
{1, 3, 5, 7, 9}
```

3.3.3 continue 语句与 break 语句

continue 语句用于提前结束本次循环的执行,即本次循环不执行 continue 之后的语句,继续进行下一次循环的准备与条件判断。

break 语句用来跳出其所在的循环,接着执行该循环之后的语句。若 break 语句所在的循环带有 else 子句,则执行 break 语句将跳过该 else 子句并结束循环。

例 3.3.3 求偶数之和

输入 n,求[1, n]范围内的所有偶数之和。

解析:

本题可使循环变量 i 从 0 开始,依次递增 2,逐个把 i 加到求和单元中。具体代码留给读者自行实现。

这里采用循环变量从 1 开始,依次递增 1,但在循环体中增加判断语句,跳过累加奇数。具体代码如下。

```
n=int(input())
s=0                                 #累加单元清 0
for i in range(1,n+1):              #从 1~n 进行循环
    if i%2==1: continue             #若是奇数,则不执行循环体中 continue 之后的语句
    s+=i                            #累加当前项
print(s)
```

运行结果如下。

```
10↵
30
```

例 3.3.4 求最大公约数

求两个正整数 m、n 的最大公约数(Greatest Common Divisor,GCD)。

解析:

本例的直观求解方法是采用穷举法,即从 m、n 这两个数中的小者到 1 逐个尝试,找第

一个能同时整除 m、n 的因子(找到则保存结果并用 break 语句跳出循环),具体代码如下。

```
m,n=map(int,input().split())
k=min(m,n)                          #用 min 函数求得 m,n 中的小者存放到 k 中
for i in range(k,0,-1):             #i 从 k 到 1 进行循环
    if m%i==0 and n%i==0:           #若 i 能同时整除 m、n,则 i 为最大公约数
        gcd=i
        break
print(gcd)
```

运行结果如下。

```
27 63↵
9
```

上述代码效率较低,在线提交时可能得到超时反馈。可用欧几里得(Euclid)算法提高求最大公约数的程序运行效率。欧几里得算法又称辗转相除法,用于计算两个整数 m、n 的最大公约数。若用 gcd(m,n)表示 m、n 的最大公约数,则欧几里得算法的计算原理如下。

```
gcd(m, n)=gcd(n, m%n)
```

例如,若要求 $m=70$、$n=16$ 的最大公约数,则计算过程如图 3-6 所示。

轮次	m	n	$t=m\%n$
1	70	16	6
2	16	6	4
3	6	4	2
4	4	2	0
5	2	0	

图 3-6　欧几里得算法求最大公约数的过程

计算时,m、n 的值不断用新值代替旧值(迭代法),直到 n 为 0 时,m 为最大公约数。具体代码如下。

```
m,n=map(int,input().split())
while n>0:                          #当 n 大于 0 时迭代
    r=m%n                           #求余数
    m=n                             #用原来的 n 替换原来 m 的值,得到新的 m
    n=r                             #用余数 r 替换原来 n 的值,得到新的 n
print(m)                            #n 为 0 时,m 为最大公约数
```

上述代码可简写如下。

```
m,n=map(int,input().split())
while n>0:                          #当 n 大于 0 时迭代
    m,n=n,m%n                       #把原来的 n 给 m,把原来的 m%n 给 n
print(m)                            #n 为 0 时,m 为最大公约数
```

实际上,数学模块 math 中提供了最大公约数函数 gcd,可直接调用该函数求两个正整数的最大公约数。具体代码如下。

```
from math import gcd                #从 math 模块导入求最大公约数的函数 gcd
m,n=map(int,input().split())
print(gcd(m,n))                     #直接调用 gcd 函数求 m、n 的最大公约数
```

思考:怎么求 m,n 的最小公倍数(Least Common Multiple,LCM)lcm?

思路 1:从 m、n 中的大者出发,逐个检查该数的 1 倍、2 倍……是否是另一个数的倍数。

思路 2:基于求得的最大公约数 gcd,设原来的 m、n 已经分别保存在 a、b 中,则最小公倍数 lcm=$a*b//$gcd。

思路 3:直接使用数学模块 math 中提供的最小公倍数函数 lcm。

思考:如何求 n 个正整数的最小公倍数?

思路:可设最小公倍数 lcm 的初值为 1,在执行 n 次的循环中对于每个输入的整数 t,求 lcm 与 t 的最小公倍数并保存在 lcm 中,则最终的 lcm 为结果。

具体代码留给读者自行实现。

3.3.4 在线做题基本程序结构

在线做题是指在 OJ 上提交代码求解问题。OJ 用户可以在线提交多种程序设计语言(如 Python、C、C++ 、Java)编写的源代码,OJ 对源代码进行编译和执行,并通过预先设计的测试数据来检验源代码的正确性。

源代码提交到 OJ 后,可能得到类似于表 3-1 所示的常见返回结果。

表 3-1 OJ 常见返回结果

返 回 结 果	返回结果缩写	备 注
Accepted	AC	程序正确,通过所有测试数据
Wrong Answer	WA	答案错,有测试数据不通过
Compile Error	CE	编译错,程序编译不能通过;此时应单击错误链接查看错误信息
Presentation Error	PE	格式错,程序没按规定的格式输出答案;一般应检查是否少了或多了空格符、换行符
Time Limit Exceeded	TLE	超时,程序没在规定时间内得出答案
Memory Limit Exceeded	MLE	超内存,程序没在规定空间内得出答案
RunTime Error	RTE	程序运行出错,意外终止等

提交代码到 OJ 测评前，至少需保证按输入样例得到输出样例，不能有任何多余或遗漏的内容，即便多一个空格或少一个“.”都不能得到 AC 反馈。

在 GPLT 等比赛及其相应 OJ 做题时，一般写一组测试的代码即可。而在 ICPC、CCPC 等比赛及相应 OJ 做题时，一般需要控制多组测试数据，常用“处理 T 次”“处理到特值结束”“处理到文件尾”3 种基本程序结构。

例 3.3.5　又见 $a+b$(1)

求两个整数之和。

输入格式：

首先输入一个正整数 T，表示测试数据的组数，然后输入 T 组测试数据。每组测试在一行上输入两个整数 a、b。

输出格式：

对于每组测试，输出一行，包含一个整数，表示 a、b 之和。

输入样例：	输出样例：
2 1 2 3 4	3 7

解析：

对于此例，使用 for 循环语句的代码如下。

```
T=int(input())                          #输入测试组数 T
for i in range(T):                      #控制从 0 到 T-1 共 T 次循环
    a,b=map(int, input().split())       #输入两个整数 a、b
    c=a+b                               #求 a+b 并保存到 c 中
    print(c)                            #输出 c 的值
```

运行结果如下。

```
2↵
1 2↵
3
3 4↵
7
```

这个运行结果看起来与输入样例和输出样例分别是一个整体不太一致，但这就是在线做题正确的输入输出，并不需要一次性输入所有数据再一次性输出所有结果，只需要根据每组输入数据都得到相应的预期输出即可。

也可使用 while 循环控制 T 组测试。具体代码如下。

```
T=int(input())                          #输入测试组数 T
while T:                                #当 T 不等于 0 时执行循环体
```

```
    a,b=map(int, input().split())        #输入两个整数 a、b
    c=a+b                                #求 a+b 并保存到 c 中
    print(c)                             #输出 c 的值
    T-=1                                 #T-=1 相当于 T=T-1,使得 T 的值减 1
```

“while T”相当于“while T!=0”,表示当 T 不等于 0 时执行循环体。

例 3.3.6　又见 $a+b$(2)

求两个整数之和。

输入格式:

测试数据有多组,处理到文件尾。每组测试在一行上输入两个整数 a、b。

输出格式:

对于每组测试,输出一行,包含一个整数,表示 a、b 之和。

输入样例:	输出样例:
1 2 3 4	3 7

解析:

Python 程序在遇到文件尾时返回 EOFError 异常,因此可以在“while True”永真循环外套一个 try 语句,使得程序在捕获到 EOFError 异常时执行 except 子句后的空语句 pass 而结束程序运行。具体代码如下。

```
try:                                     #用 try 语句处理异常
    while True:
        a,b=map(int,input().split())     #输入两个整数
        c=a+b                            #求 a+b 并保存到 c 中
        print(c)                         #输出 c 的值
except EOFError: pass                    #except 子句,遇到 EOFError 异常执行空语句
```

运行结果如下。

```
1 2↵
3
3 4↵
7
```

上面的代码结构在捕获到 EOFError 时执行空语句 pass,结束执行“while True”永真循环。在本地测试时可用组合键 Ctrl+D 表示输入结束(相当于文件尾)。

因是在输入时才会遇到文件尾,故可仅对输入语句使用 try…except 语句处理 EOFError 异常。具体代码如下。

```
while True:
    try:                                 #用 try 语句处理异常
```

```
        a,b=map(int,input().split())    #输入两个整数
    except EOFError: break              #遇到 EOFError 异常时用 break 语句跳出循环
    c=a+b                               #求 a+b 并保存到 c 中
    print(c)                            #输出 c 的值
```

“while True”是一个永真循环，若循环体中无结束循环的语句，则循环将一直执行，成为无限循环(或称死循环)。上述代码在捕获到 EOFError 时通过 break 语句结束循环。

控制到文件尾也可使用 for 循环，当 for 循环的迭代变量能从系统模块 sys 的标准输入 sys.stdin 中取得数据时继续执行循环。具体代码如下。

```
import sys                              #引入系统模块 sys
for it in sys.stdin:                    #当能从 sys.stdin 中取得数据时执行循环
    a,b=map(int,it.split())             #输入两个整数
    c=a+b                               #求 a+b 并保存到 c 中
    print(c)                            #输出 c 的值
```

注意，使用标准输入 sys.stdin 之前，需用 import 语句导入系统模块 sys。

例 3.3.7　又见 $a+b$(3)

求两个整数之和。

输入格式：

测试数据有多组。每组测试在一行上输入两个整数 a、b，当 a、b 同时为 0 时，输入结束。

输出格式：

对于每组测试，输出一行，包含一个整数，表示 a、b 之和。

输入样例：	输出样例：
1 2 3 4 0 0	3 7

解析：

本题的特值是指 a、b 同时为 0。可用“while True”永真循环，当输入的 a、b 同时为 0 时，执行 break 语句跳出循环，从而结束循环。具体代码如下。

```
while True:
    a,b=map(int,input().split())    #输入两个整数
    if a==0 and b==0:               #若 a、b 为 0，则结束循环
        break                       #break 语句用于跳出循环
    c=a+b                           #求 a+b 并保存到 c 中
    print(c)                        #输出 c 的值
```

运行结果如下。

```
1 2↵
3
3 4↵
7
0 0↵
```

上面是 3 种基本的在线做题程序结构,在线做题时可能会遇到综合运用各种程序结构的情况。读者可以在不断的解题过程中逐步熟悉和掌握在线做题的程序结构。初学者在 OJ 做题时遇到多组测试,可以直接套用以下在线做题基本程序结构,只要把一组测试的代码替换为具体题目的解题代码即可。

1. 处理 *T* 次

(1) 用 for 循环控制 T 组测试的代码结构如下。

```
T=int(input())
for i in range(T):
    #一组测试的代码
```

(2) 用 while 循环控制 T 组测试的代码结构如下。

```
T=int(input())
while T:
    T-=1
    #一组测试的代码
```

2. 处理到特值结束

处理到特值结束的代码结构(设控制到整数 n 为特值 0 时结束)如下。

```
while True:
    n=int(input())              #输入根据具体题目调整
    if n==0:                    #n==0 这个条件根据具体题目调整
        break                   #break 语句用于跳出循环
    #一组测试的代码
```

3. 处理到文件尾

(1) 用 while 循环控制到文件尾的代码结构如下。

```
try:
    while True:
        #一组测试的代码
except EOFError: pass
```

(2) 用 for 循环控制到文件尾的代码结构如下。

```
import sys
for it in sys.stdin:
    #一组测试的代码
```

3.3.5 循环结构运用举例

例 3.3.8 数据统计

在一行上先输入一个整数 n，再输入 n 个整数，请统计其中负数、0 和正数的个数。

解析：

本例可以设置 3 个计数器(统计个数的变量，初值为 0)，先将输入的 n 个整数存放到列表 a 中，再遍历列表 a 判断其中的每个元素是正、负或 0 中的哪一种，把对应计数器增 1。具体代码如下。

```
zero=0                                  #0 的计数器
positive=0                              #正数的计数器
negative=0                              #负数的计数器
n, *a=list(map(int,input().split()))    #输入整数 n 及 n 个整数存放到列表 a 中
for it in a:                            #遍历列表 a,迭代变量 it 依次为列表中的每一项
    if it==0:                           #若当前项为 0,则相应计数器加 1
        zero+=1
    elif it>0:                          #若当前项为正,则相应计数器加 1
        positive+=1
    else:                               #若当前项为负,则相应计数器加 1
        negative+=1
print(negative,zero,positive)
```

运行结果如下。

```
10 -5 0 0 9 6 -8 -7 -8 0 9↵
4 3 3
```

本题也可用列表产生式求解。具体代码如下。

```
n, *a=list(map(int,input().split()))    #输入整数 n 及 n 个整数存放到列表 a 中
zero=sum([1 for it in a if it==0])      #统计 0 的个数
positive=sum([1 for it in a if it>0])   #统计正数的个数
negative=sum([1 for it in a if it<0])   #统计负数的个数
print(negative, zero, positive)         #输出结果
```

例 3.3.9 亲和数判断

亲和数是古希腊数学家毕达哥拉斯(Pythagoras)在自然数研究中发现的。若两个自然数中任何一个数都是另一个数的真约数(即不是自身的约数)之和，则它们就是亲和数。例

如,220和284是亲和数,因为220的所有真约数之和为1+2+4+5+10+11+20+22+44+55+110=284,而且284的所有真约数之和为1+2+4+71+142=220。请判断输入的两个整数是否是亲和数,若是则输出YES,否则输出NO。

解析:

本题根据亲和数的定义把两个整数的真约数之和各自求出,再判断各自是否等于另一个数即可。具体代码如下。

```
a,b=map(int,input().split())
sa=0                              #sa存放a的因子之和
for i in range(1,a//2+1):         #从1到a//2把a的因子累加到sa中
    if a%i==0:
        sa+=i
sb=0                              #sb存放b的因子之和
for i in range(1,b//2+1):         #从1到b//2把b的因子累加到sb中
    if b%i==0:
        sb+=i
if a==sb and b==sa:               #若满足亲和数的条件则输出YES,否则输出NO
    print("YES")
else:
    print("NO")
```

运行结果如下。

```
220 284↵
YES
```

本题也可用列表产生式求解,具体代码如下。

```
a,b=map(int,input().split())
sa=sum([i for i in range(1,a//2+1) if a%i==0])   #从1到a//2把a的因子求和到sa中
sb=sum([i for i in range(1,b//2+1) if b%i==0])   #从1到b//2把b的因子求和到sb中
print('YES' if a==sb and b==sa else 'NO')        #若满足条件则输出YES,否则输出NO
```

例3.3.10　求数位之和

输入一个正整数,求其各个数位上的数字之和。例如,输入12345,输出15。

解析:

本题需要数位分离,即把一个整数的个位、十位、百位等数位分离出来。可以不断地取得个位相加,再把个位去掉,直到该数等于0为止。具体代码如下。

```
n=int(input())                    #输入整数n
s=0                               #累加单元清0
while n>0:                        #当n>0时循环
    s+=n%10                       #n%10取得n的个位
```