第 3 章

顺序结构程序设计

程序由多条语句构成，描述计算机的执行步骤。人们利用计算机解决问题，必须预先将问题转化为用计算机语句描述的解题步骤，即程序。也就是说，程序在计算机上执行时，程序中的语句完成具体的操作并控制计算机的执行流程，但程序并不一定完全按照语句序列的书写顺序来执行。程序中语句的执行顺序称为“程序结构”。程序包含三种基本结构：顺序结构、选择结构和循环结构。如果程序中的语句是按照书写顺序执行，则称其为“顺序结构”；如果程序中某些语句按照某个条件来决定是否执行，则称其为“选择结构”；如果程序中某些语句反复执行多次，则称其为“循环结构”。

顺序结构是最简单的一种结构，它只需按照处理顺序依次写出相应的语句即可。因此，学习程序设计，首先从顺序结构开始。本章主要介绍算法的概念、程序设计的基本结构、数据的输入/输出及顺序程序设计方法。

3.1 算　　法

开发程序的目的，就是要解决实际问题。然而，面对各种复杂的实际问题，如何编写程序，往往令初学者感到茫然。程序设计语言只是一个工具，只懂得语言的规则并不能保证编写出高质量的程序。程序设计的关键是算法设计，算法和程序设计与数据结构密切相关。简单地讲，算法是解决问题的策略、规则和方法。算法的具体描述形式很多，但计算机程序是对算法的一种精确描述，而且可在计算机上运行。

3.1.1 算法的概念

算法就是解决问题的一系列操作步骤的集合。比如，厨师做菜时，要经过一系列的步骤——洗菜、切菜、配菜、炒菜和装盘。用计算机解题的步骤就称为算法，编程人员必须告诉计算机先做什么，再做什么，这可以通过高级语言的语句来实现。通过这些语句，一方面体现了算法的思想，另一方面指示计算机按算法的思想去工作，从而解决实际问题。程序就是由一系列的语句组成的。

著名的计算机科学家沃思（Niklaus Wirth）曾经提出一个著名的公式：

数据结构＋算法＝程序

数据结构是指对数据（操作对象）的描述，即数据的类型和组织形式；算法则是对操作步骤的描述。也就是说，数据描述和操作描述是程序设计的两项主要内容。数据描述的主要内容是基本数据类型的组织和定义，数据操作则是由语句来实现的。算法具有下列

特性。

1. 有穷性

任意一组合法输入值，在执行有穷步骤之后一定能结束，即算法中的每个步骤都能在有限时间内完成。

2. 确定性

算法的每一步必须是确切定义的，使算法的执行者或阅读者都能明确其含义及如何执行，并且在任何条件下，算法都只有一条执行路径。

3. 可行性

算法应该是可行的，算法中的所有操作都必须足够基本，都可以通过已经实现的基本操作运算有限次实现。

4. 有输入

一个算法应有零个或多个输入，它们是算法所需的初始量或被加工对象的表示。有些输入量需要在算法执行过程中输入，而有的算法表面上可以没有输入，实际上已被嵌入到算法之中。

5. 有输出

一个算法应有一个或多个输出，它是一组与输入有确定关系的量值，是算法进行信息加工后得到的结果，这种确定关系即为算法的功能。

6. 有效性

在一个算法中，要求每一个步骤都能有效地执行。

以上这些特性是一个正确的算法应具备的特性，在设计算法时应该注意。

3.1.2 算法的评价标准

什么是“好”的算法，通常从下面几个方面衡量算法的优劣。

1. 正确性

正确性是指算法能满足具体问题的要求，即对任何合法的输入，算法都会得出正确的结果。

2. 可读性

可读性指算法被理解的难易程度。算法主要是为了人的阅读与交流，其次才是为计算机执行，因此算法应该更易于人的理解。另一方面，晦涩难读的程序易于隐藏较多错误而难以调试。

3. 健壮性（鲁棒性）

健壮性即对非法输入的抵抗能力。当输入的数据为非法时，算法应当恰当地做出反应或进行相应处理，而不是产生奇怪的输出结果。并且，处理出错的方法不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

4. 高效率与低存储量需求

通常，效率指的是算法执行时间，存储量指的是算法执行过程中所需的最大存储空间，两者都与问题的规模有关。尽管计算机的运行速度提高很快，但这种提高无法满足问题规模加大带来的速度要求。所以追求高速算法仍然是必要的。相比起来，人们会更多地关注算法的效率，但这并不因为计算机的存储空间是海量的，而是由人们面临的问题的本质决定的。二者往往是一对矛盾，常常可以用空间换时间，也可以用时间换空间。

3.1.3　算法的表示

算法就是对特定问题求解步骤的描述，可以说是设计思路的描述。在算法定义中，并没有规定算法的描述方法，所以它的描述方法可以是任意的。既可以用自然语言描述，也可以用数学方法描述，还可以用某种计算机语言描述。若用计算机语言描述，就是计算机程序。

为了能清晰地表示算法，程序设计人员采用更规范的方法。常用的有自然语言描述、流程图、N-S 结构流程图、伪代码等。

1. 自然语言描述

自然语言就是人们日常生活中应用的语言，用自然语言表示通俗易懂，容易被人们接受，也更容易学习和表达，但自然语言文字冗长，而且容易产生歧义。假如有这样一句话："他看到我很高兴。"请问这句话是表达了他高兴还是我高兴？仅从这句话本身很难判断。此外，用自然语言描述包含分支和循环的算法十分不方便。因此，除了一些十分简单的算法外，一般不采用自然语言来描述算法。

2. 流程图

流程图是描述算法最常用的一种方法，利用集合图形符号来代表不同性质的操作，用流程线来指示算法的执行方向，ANSI（美国国家标准化协会）规定的一些常用流程图符号如图 3.1 所示。这种表示直观、灵活，很多程序员采用这种表示方法，因此又称为传统的流程图。本书中的算法将采用这种表示方法描述，读者应对这种流程图熟练掌握。

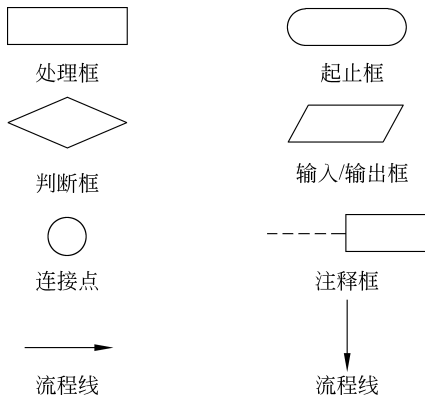

图 3.1　常用流程图符号

【例 3.1】　设计一个算法，求三个整数之和，画出流程图。

求三个整数之和的算法流程图如图 3.2 所示。

注意：画流程图时，每个框内要说明操作内容，描述要确切，不要有"二义性"。画箭头时要注意箭头的方向，箭头方向表示程序执行的流向。

【例 3.2】　求两个正整数的最大公约数。

求两个正整数的最大公约数的算法流程图如图 3.3 所示。

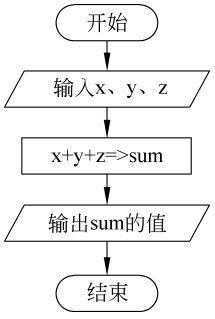

图 3.2　求三个整数之和的算法流程图

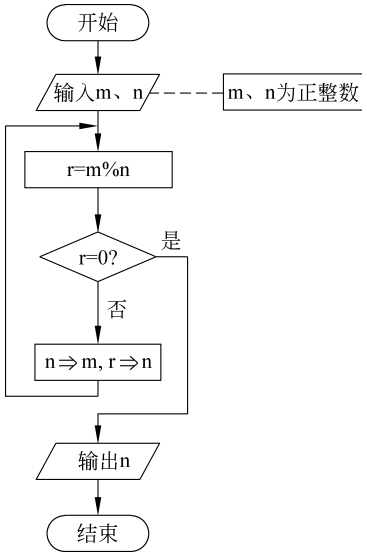

图 3.3　求两个正整数的最大公约数的算法流程图

【例 3.3】 设计解一元二次方程 $ax^2+bx+c=0(a\neq 0)$ 的算法，画出流程图。

分析：求解一元二次方程可按照以下步骤完成。

(1) 计算 $\Delta=b^2-4ac$。

(2) 如果 $\Delta<0$，则原方程无实数解。

否则($\Delta\geqslant 0$)，计算：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}\quad;\quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

(3) 输出解 x_1、x_2 或无实数解信息。

流程图如图 3.4 所示。

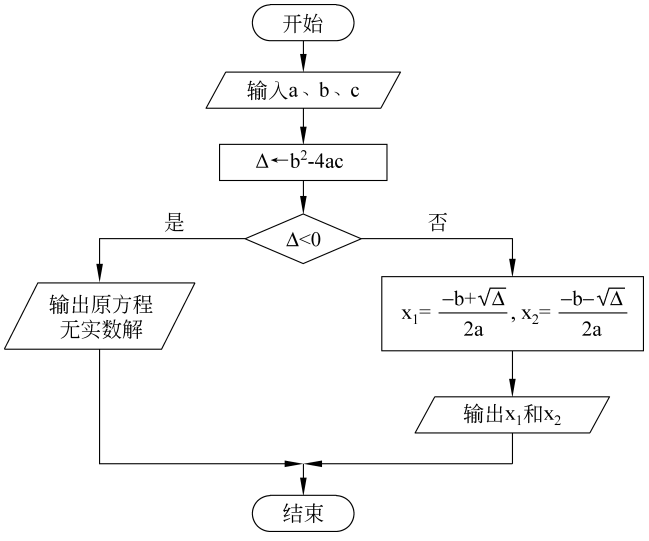

图 3.4　求解一元二次方程的算法流程图

3. N-S 结构流程图

N-S 结构流程图是美国学者 I.Nassi 和 B.Shneiderman 于 1973 年提出的一种新的流程图形式。在这种流程图中完全去掉了流程线，全部算法写在一个矩形框内，而且在框内还可以包含其他的框。这样算法只能从上到下顺序执行，从而避免了算法流程的任意转向，保证了程序的质量。

例 3.1 的 N-S 结构流程图如图 3.5 所示。

例 3.2 的 N-S 结构流程图如图 3.6 所示。

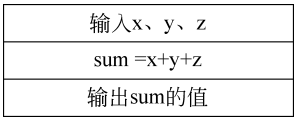

图 3.5　例 3.1 的 N-S 结构流程图

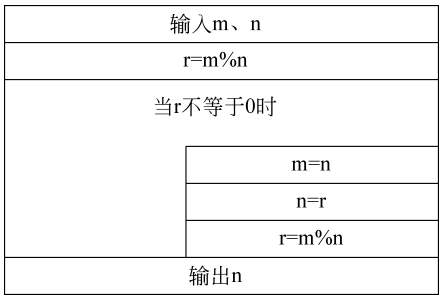

图 3.6　例 3.2 的 N-S 结构流程图

4. 伪代码

伪代码是介于自然语言和计算机语言之间的文字和符号，是帮助程序员制定算法的智能化语言，它不能在计算机上运行，但使用起来比较灵活，无固定格式规范，只要写出来自己或别人能看懂即可。由于它与计算机语言比较接近，因此易于转换为计算机程序。

```
input a,b,c
Δ= b²-4ac
if  Δ<0    then
    print "方程无实数解"
else
    x1=(-b+sqrt(Δ))/(2*a)
    x2=(-b-sqrt(Δ))/(2*a)
print  x1,x2
```

在以上几种描述算法的方法中，具有熟练编程经验的人士喜欢用伪代码，初学者使用流程图或 N-S 结构流程图较多，因为易于理解，比较形象。

3.2　程序的基本结构

随着计算机技术的发展，编制的程序越来越复杂。一个复杂程序多达数千万条语句，而且程序的流向也很复杂，常常用无条件转向语句去实现复杂的逻辑判断功能。因而造成程序质量差，可靠性很难保证，同时也不易阅读，维护困难，20 世纪 60 年代末期，国际

上出现了所谓的“软件危机”。

为了解决这一问题，就出现了结构化程序设计，它的基本思想是像玩积木游戏那样，只要有几种简单类型的结构，可以构成任意复杂的程序。这样可以使程序设计规范化，便于用工程的方法来进行软件生产。基于这样的思想，1966 年意大利的 Bobra 和 Jacopini 提出了 3 种基本结构，即顺序结构、选择结构和循环结构，由这 3 种基本结构组成的程序就是结构化程序。

3.2.1 顺序结构

顺序结构是最简单的一种结构，其语句是按书写顺序执行的，除非指示转移，否则计算机自动以语句编写的顺序一句一句地执行。顺序结构的语句程序流向是沿着一个方向进行，有一个入口(A)和一个出口(B)。流程图和 N-S 结构流程图如图 3.7 和图 3.8 所示，先执行程序模块 A，然后再执行程序模块 B。程序模块 A 和 B 分别代表某些操作。

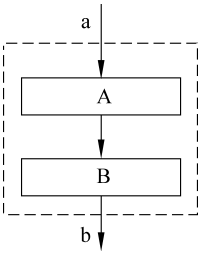

图 3.7 顺序结构的流程图

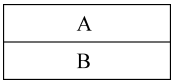

图 3.8 顺序结构的 N-S 结构流程图

3.2.2 选择结构

在选择结构中，程序可以根据某个条件是否成立，选择执行不同的语句。选择结构如图 3.9 和图 3.10 所示。当条件成立时执行模块 A；否则条件不成立时执行模块 B。模块 B 也可以为空，如图 3.11 所示。当条件为真时执行某个指定的操作(模块 A)，条件为假时跳过该操作(单路选择)。

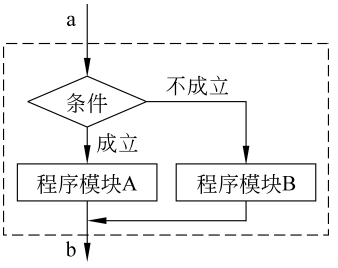

图 3.9 分支结构的流程图

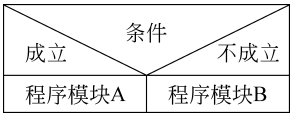

图 3.10 分支结构的 N-S 结构流程图

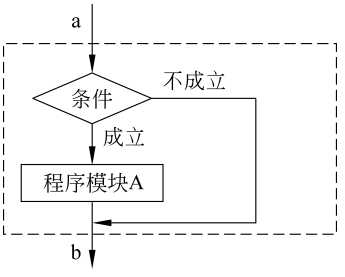

图 3.11 单分支结构的流程图

3.2.3　循环结构

在循环结构中，可以使程序根据某种条件和指定的次数，使某些语句执行多次。循环结构有两种形式：当型循环和直到型循环。

1. 当型循环

先判断，只要条件成立（为真）就反复执行程序模块；当条件不成立（为假）时则结束循环。当型循环结构的流程图和 N-S 结构流程图分别如图 3.12(a)和图 3.12(b)所示。

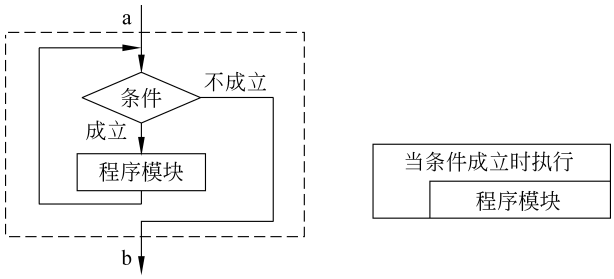

图 3.12　当型循环结构的流程图和 N-S 结构流程图

2. 直到型循环

先执行程序模块，再判断条件是否成立。如果条件成立（为真）则继续执行循环体；当条件不成立（为假）时结束循环。直到型循环结构的流程图和 N-S 结构流程图如图 3.13 所示。

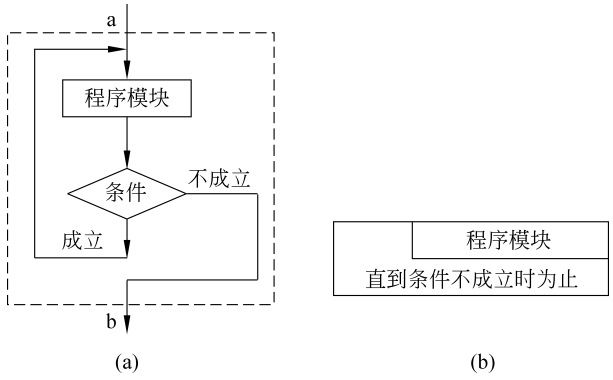

图 3.13　直到型循环结构的流程图和 N-S 结构流程图

注意：无论是顺序结构、选择结构还是循环结构，它们都有一个共同的特点，即只有一个入口和一个出口。从示意的流程图可以看到，如果把基本结构看作一个整体（用虚线框表示），执行流程从 a 点进入基本结构，而从 b 点脱离基本结构。整个程序由若干个这样的基本结构组成。三种结构之间可以是平行关系，也可以相互嵌套，通过结构之间的复合形成复杂的结构。结构化程序的特点就是单入口、单出口。

3.3 数据的输入与输出

通常，一个程序可以分成三步进行：输入原始数据、进行计算处理和输出运行结果。其中，数据的输入与输出是用户通过程序与计算机进行交互的操作，是程序的重要组成部分。本节详细介绍 Python 的输入与输出。

3.3.1 标准输入输出

1. 标准输入

Python 提供了内置函数 input()从标准输入设备读入一行文本，默认的标准输入设备是键盘。input()函数的基本格式为：

```
input([提示字符串])
```

说明：方括号中的提示字符串是可选项，如果有提示字符串，运行时原样显示，给用户提示。

在 Python 2.x 与 Python 3.x 中该函数的使用方法略有不同。

在 Python 2.x 中，该函数返回结果的类型由输入时所使用的界定符来决定。例如：

```
>>>x=input("Please enter your input: ")
Please enter your input: 5              #没有界定符,x 为整数
>>>x=input("Please enter your input: ")
Please enter your input: '5'            #单引号界定符,x 为字符串
>>>x=input("Please enter your input: ")
Please enter your input: [1,2,3]        #方括号界定符,x 为列表
>>>x=input("Please enter your input: ")
Please enter your input: (1,2,3)        #圆括号界定符,x 为元组
```

在 Python 2.x 中还提供一个内置函数 raw_input()函数用来接收用户输入的值，该函数将用户的所有输入都看作字符串，返回字符串类型。例如：

```
>>>x=raw_input ("Please enter your input: ")
Please enter your input: 5
>>>x
'5'
>>>x=raw_input ("Please enter your input: ")
Please enter your input: (1,2,3)
>>>x
'(1,2,3)'
```

在 Python 3.x 中，将 raw_input()和 input()进行了功能整合，去除了 raw_input()函

数，仅保留 input()函数。input()函数接收任意输入，将所有输入默认为字符串处理，并返回字符串类型，相当于 Python 2.x 中的 raw_input()函数。例如：

```
>>>x=input("Please enter your input: ")
Please enter your input: 5
>>>print(type(x))
<class 'str'>
```

说明：内置函数 type()用来返回变量类型。上例中当输入数值 5 赋给变量 x 之后，x 的类型为字符串。

```
>>>x=input ("Please enter your input:")
Please enter your input: (1,2,3)
>>>print(type(x))
<class 'str'>
```

如果要输入数值类型数据，可以使用类型转换函数把字符串转换为数值。例如：

```
>>>x=int(input ("Please enter your input:"))
"Please enter your input:5
>>> print(type(x))
<class 'int'>
```

说明：x 接收的是字符串 5，通过 int()函数将字符串转换为整型类型。

input()函数还可给多个变量赋值。例如：

```
>>>x,y=input()
3,4
>>>x
3
>>>y
5
```

2. 标准输出

在 Python 2.x 与 Python 3.x 中的输出方法也不完全一致。在 Python 2.x 中使用 print 语句进行输出，Python 3.x 中使用 print()函数进行输出。

本书给出的例子大部分是在 Python 3.10.2 环境下编写运行，因此这里重点介绍 print()函数的用法。

print()函数一般形式为：

```
print([输出项 1,输出项 2,…,输出项 n][,sep=分隔符][,end=结束符])
```

说明：输出项之间用逗号分隔，没有输出项时输出一个空行。sep 表示输出时各输出项之间的分隔符(缺省时用空格分隔)，end 表示输出时的结束符(缺省时以回车换行结束)。print()函数从左至右输出各项的值，并将各输出项的值依次显示在屏幕的同一行上。例如：

```
>>>x,y=2,3
>>>print(x,y)
2 3
>>>print(x,y,sep=':')
2:3
print(x,y,sep=':',end='%)
2:3%
```

3.3.2 格式化输出

在很多实际应用中需要将数据按照一定格式输出。

1. 字符串格式化%

Python 中 print()函数可以按照指定的输出格式在屏幕上输出相应的数据信息。其基本做法是：将输出项格式化，然后利用 print()函数输出。

在 Python 中格式化输出时，采用%分隔格式控制字符串与输出项，一般格式为：

```
格式控制字符串%(输出项 1,输出项 2,…,输出项 n)
```

功能是按照“格式控制字符串”的要求，将输出项 1，输出项 2，…，输出项 n 的值输出到输出设备上。

其中格式控制字符串用于指定输出格式，它包含如下两类字符：

(1) 常规字符：包括可显示的字符和用转义字符表示的字符。

(2) 格式控制符：以%开头的一个或多个字符，以说明输出数据的类型、形式、长度、小数位数等，如"%d"表示按十进制整型输出；"%c"表示按字符型输出等。格式控制符与输出项应一一对应。

对应不同类型数据的输出，Python 采用不同的格式说明符描述。格式说明如表 3.1 所示。

表 3.1 print()函数的格式说明

格式符	格 式 说 明
d 或 i	以带符号的十进制整数形式输出整数(正数省略符号)
o	以八进制无符号整数形式输出整数(不输出前导 0)
x 或 X	以十六进制无符号整数形式输出整数(不输出前导符 0x)。用 x 时，以小写形式输出包含 a、b、c、d、e、f 的十六进制数；用 X 时，以大写形式输出包含 A、B、C、D、E、F 的十六进制数

续表

格式符	格 式 说 明
c	以字符形式输出，输出一个字符
s	以字符串形式输出
f	以小数形式输出实数，默认输出 6 位小数
e 或 E	以标准指数形式输出实数，数字部分隐含 1 位整数，6 位小数。使用 e 时，指数以小写 e 表示；使用用 E 时，指数以大写 E 表示
g 或 G	根据给定的值和精度，自动选择 f 与 e 中较紧凑的一种格式，不输出无意义的 0

例如：

```
print("sum=%d"%x)
```

若 x=300，则输出为

```
sum=300
```

格式控制字符串中“sum =”照原样输出，“%d”表示以十进制整数形式输出。

对输出格式，Python 语言同样提供附加格式字符，用于对输出格式作进一步描述。在使用表 3.1 的格式控制字符时，在%与格式字符之间可以根据需要使用附加字符，使得输出格式的控制更加准确。附加格式说明符如表 3.2 所示。

表 3.2　附加格式说明符

附加格式说明符	格 式 说 明
m	域宽，十进制整数，用以描述输出数据所占宽度。如果 m 大于数据实际位数，输出时前面补足空格；如果 m 小于数据的实际位数，按实际位数输出。当为小数时，小数点或占 1 位
n	附加域宽，十进制整数，用于指定实型数据小数部分的输出位数。如果 n 大于小数部分的实际位数，输出时小数部分用 0 补足；如果 n 小于小数部分的实际位数，输出时将小数部分多余的位四舍五入。如果用于字串数据，表示从字串中截取的字符数
－	输出数据左对齐，默认时为右对齐
＋	输出正数时，以＋号开头
#	作为 o、x 的前缀时，输出结果前面加上前导符号 0、0x

这样，格式控制字符的形式为：

```
%[附加格式说明符]格式符
```

注意：书中语句格式描述时用方括号表示可选项，其余出现在格式中的非汉字字符

均为定义符，应原样照写。

例如，可在%和格式字符之间加入形如“m.n”（m、n 均为整数，含义如表 3.2 所示）的修饰。其中，m 为宽度修饰，n 为精度修饰。如%7.2f，表示用实型格式输出，附加格式说明符“7.2”表示输出宽度为 7，输出 2 位小数。

下面是一些格式化输出的示例。

```
>>>year = 2017
>>>month = 1
>>>day = 28
#格式化日期，将%02d 数字转换成 2 位整型，缺位补 0
>> print('%04d-%02d-%02d'%(year,month,day))
2017-01-28                    #运行结果
>>>value = 8.123
>> print('%06.2f'%value)      #保留宽度为 6，小数点后 2 位小数的数据
008.12                        #运行结果
>>>print('%d'%10)             #输出十进制数
>>>print('%o'%10)             #输出八进制数
12
>>>print('%02x'%10)           #输出两位十六进制数，字母小写，空缺位补 0
0a
>>>print('%04X'%10)           #输出四位十六进制数，字母大写，空缺位补 0
000A
>>>print('%.2e'%1.2888)       #以科学计数法输出浮点型数，保留 2 位小数
1.29e+00
```

3.3.3 字符串的 format()方法

在 Python 中，字符串有一种 format()方法。这个方法会将格式字符串当作一个模板，通过传入的参数对输出项进行格式化。

字符串 format()方法的一般形式为：

```
格式字符串.format(输出项 1,输出项 2,…,输出项 n)
```

其中，格式字符串中可以包括普通字符和格式说明符，普通字符原样输出，格式说明符决定了所对应输出项的格式。

格式字符串使用大括号括起来，一般形式为：

```
{[序号或键]: 格式说明符}
```

其中，可选项序号表示要格式化的输出项的位置，从 0 开始，0 表示输出项 1，1 表示输出项 2，以后以此类推。序号可全部省略。若全部省略表示按输出项的自然顺序输出。可选项键对应要格式化的输出项名字或字典的键值。格式说明符如表 3.1 所示，以冒号

开头。

以下是使用 format()的应用实例。

(1) 使用“{序号}”形式的格式说明符：

```
>>>"{} {}".format("hello", "world")          #不设置指定位置,按默认顺序
'hello world'
>>> "{0}{1}".format("hello", "world")        #设置指定位置
'hello world'
>>> "{1}{0} {1}".format("hello", "world")    #设置指定位置,输出项 2 重复输出
'world hello world'
```

(2) 使用“{序号：格式说明符}”形式的格式说明符：

```
>>> "{0:.2f},{1}".format(3.1415926,100)
'3.14,100'
```

该例中，“{0:.2f}”决定了该格式说明符对应于输出项 1，“.2f”说明输出项 1 的输出格式，即以浮点数形式输出，小数点后保留 2 位小数。

(3) 使用“{键}”形式的格式说明符：

```
>>> "pi={x}".format(x=3.14)
'pi=3.14'
```

(4) 混合使用“{序号}”“{键}”形式的格式说明符：

```
>>> "{0},pi={x}".format("圆周率",x=3.14)
'圆周率,pi=3.14'
```

在 format()方法格式字符串中，除了可以包括序号或键外，还可以包含格式控制标记。格式控制标记用来控制参数输出时的格式。格式控制标记如表 3.3 所示。

表 3.3　format()方法中的格式控制标记

格式控制标记	说　明
＜宽度＞	设定输出数据的宽度
＜对齐＞	设定对齐方式，有左对齐、右对齐和居中对齐三种形式
＜填充＞	设定用于填充的单个字符
,	数字的千位分隔符
＜.精度＞	浮点数小数部分精度或字符串最大输出长度
＜类别＞	输出整数和浮点数类型的格式规则

表 3.3 中的这些字段都是可选的，也可以组合使用。

1. ＜宽度＞

＜宽度＞用来设定输出数据的宽度。如果该输出项对应的 format( )参数长度比＜宽度＞设定值大，则使用参数实际长度；如果该值的实际位数小于指定宽度，则位数将被默认以空格字符填充。例如：

```
>>> "{0:10}".format("Python")
'Python    '                    #输出宽度设定为 10,字符串右边以 5 个空格填充
>>> "{0:10}".format("Python Programming")
'Python Programming'            #输出宽度设定为 10,字符串实际宽度大于 10
```

2. ＜对齐＞

参数在＜宽度＞内输出时的对齐方式，分别使用“＜”“＞”和“^”三个符号表示左对齐、右对齐和居中对齐。例如：

```
>>> "{0:>10}".format("Python")          #输出时右对齐
'    Python'
>>> "{0:<10}".format("Python")          #输出时左对齐
'Python    '
>>>"{0:^10}".format("Python")           #输出时居中对齐
'  Python  '
```

3.＜填充＞

＜填充＞是指＜宽度＞内除了参数外的字符采用的表示方式，默认采用空格，可以通过＜填充＞更换。例如：

```
>>>"{0:*^10}".format("Python")         #输出时居中且使用 * 填充
'**Python**'
>>>"{0:#<10}".format("Python")         #输出时左对齐且使用#填充
'Python####'
```

4. 逗号(,)

逗号(,)用于显示数字的千位分隔符，适用于整数和浮点数。例如：

```
>>>'{0:,}'.format(1234567890)          #用于整数输出
'1,234,567,890'
>>>'{0:,}'.format(1234567.89)          #用于浮点数输出
'1,234,567.89'
```

5. ＜.精度＞

＜.精度＞由小数点(.)开头。对于浮点数，精度表示小数部分输出的有效位数；对于

字符串，精度表示输出的最大长度。例如：

```
>>>'{0:.2f},{1:.5}'.format(1.2345,'programming')
'1.23,progr'
```

6. ＜类型＞

＜类型＞表示输出整数和浮点数类型时的格式规则。

对于整数类型，输出格式包括以下 6 种。

- b：输出整数的二进制方式。
- c：输出整数对应的 Unicode 字符。
- d：输出整数的十进制方式。
- o：输出整数的八进制方式。
- x：输出整数的小写十六进制方式。
- X：输出整数的大写十六进制方式。

例如：

```
>>>'{:b}'.format(10)
'1010'
>>>'{:d}'.format(10)
'10'
>>>'{:x}'.format(95)
'5f'
>>>'{:X}'.format(95)
'5F'
```

对于浮点数类型，输出格式包括以下 4 种。

- e：输出浮点数对应的小写字母 e 的指数形式。
- E：输出浮点数对应的大写字母 E 的指数形式。
- f：输出浮点数的标准浮点形式。
- %：输出浮点数的百分形式。

例如：

```
>>> '{0:e},{0:E},{0:f},{0:%}'.format(123.456789)
'1.234568e+02,1.234568E+02,123.456789,12345.678900%'
>>> '{0:.2e},{0:.2E},{0:.2f},{0:.2%}'.format(123.456789)
'1.23e+02,1.23E+02,123.46,12345.68%'
```

注意：浮点数输出时尽量使用＜.精度＞表示小数部分的宽度，有助于更好控制输出格式。

3.4 顺序结构程序设计举例

到目前为止，介绍的程序都是逐条语句书写的，程序的执行也是按照顺序逐条执行的，这种程序称为顺序程序。

下面是能够实现实际功能的顺序程序设计示例，虽然不难，但对形成清晰的编程思路是有帮助的。

【例 3.4】 从键盘输入一个 3 位整数，分离出它的个位、十位和百位并分别在屏幕输出。

分析：此题要求设计一个从三位整数中分离出个位、十位和百位数的算法。例如，输入的数是 235，则输出分别是 2、3、5。百位数字可采用对 100 整除的方法得到，235//100＝2；个位数字可采用对 10 求余的方法得到，235％10＝5；十位数字可通过将其变化为最高位后再整除的方法得到，(235－2＊100)//10＝3，也可通过将十位数字其变换为最低位再求余的方法得到，(235/10)％10＝3。

根据以上分析，程序应分为三步完成。

(1) 调用 input()函数输入一个三位整数。

(2) 利用上述算法计算该数的个位、十位和百位数。

(3) 输出计算后的结果。

程序如下：

```
x=int(input("请输入一个 3 位整数"))
a=x//100
b=(x-a*100)//10
c=x%10
print("百位=%d,十位=%d,个位=%d"%(a,b,c))
```

程序运行结果：

```
请输入一个 3 位整数 235
百位=2,十位=3,个位=5
```

【例 3.5】 小写字母转盘(如图 3.14 所示)。

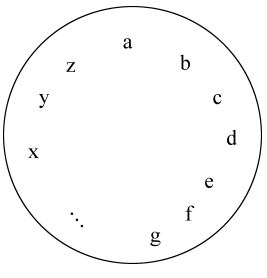

图 3.14 小写字母转盘

用户输入一个小写字母，求出该字母的前驱和后继字符。例如，c 字符的前驱和后继分别是 b 和 d，a 字符的前驱和后继分别是 z 和 b，z 字符的前驱和后继分别是 y 和 a。

分析：首先应该输入一个小写字母存储到字符类型变量(假设为 ch 变量)中，接着再求该字符的前驱和后继。

求一个字母的前驱字母并不是简单地减 1，例如，字母 a 的前驱是 z，不能通过减 1 来实现。在没有学习条件控制之前，可以利用取余操作的特性，即任何一个整数除以 26(26 个字母)

的余数只能为 0～25。可以以 z 为参考点，首先求出输入的字符 ch(假设是 w)与 z 之间的字符偏移数 n='z'－ch='z'－'w'＝3，而(n＋1)％26＝4 则是 ch(字母 w)的前驱字母相对于 z 的偏移数，'z'－(n＋1)％26＝122－4＝118(即字母 v)就是 ch(字母 w)的前驱字母。

可以采用同样的道理求后继字母。

程序如下：

```
ch=input("请输入一个字符：")
pre=ord('z')-(ord('z')-ord(ch)+1)%26          #ord()函数用来得到字符的 ASCII 值
next=ord('a')+(ord(ch)-ord('a')+1)%26
print("%c 的前驱字母是%c,后继字母是%c"%(ch,pre,next))
```

程序运行结果：

```
请输入一个字符：z
z 的前驱字母是 y,后继字母是 a
```

再次运行程序，结果如下：

```
请输入一个字符：a
a 的前驱字母是 z,后继字母是 b
```

习　　题

1. 选择题

(1) 算法是指(　　)。

A. 数学计算公式　　B. 对问题的精确描述

C. 程序的语句序列　　D. 解决问题的精确步骤

(2) 下列叙述正确的是(　　)。

A. 算法的效率只与问题的规模有关，而与数据的存储结构无关

B. 算法的时间复杂度与空间复杂度一定相关

C. 数据的逻辑结构与存储结构是一一对应的

D. 算法的时间复杂度是指执行算法所需要的计算工作量

(3) 以下(　　)是流程图的基本元素。

A. 判断框　　B. 顺序结构　　C. 分支结构　　D. 循环结构

(4) 程序流程图中带有箭头的线段表示的是(　　)。

A.调用关系　　B. 控制流　　C. 图元关系　　D. 数据流

(5) 以下程序段的运行结果是(　　)。

```
s='PYTHON'
print("{0:3}".format(s))
```

A. PYT　　B. PYTH　　C. PYTHON　　D. PYTHON

(6) 利用print()函数进行格式化输出，(　　)用于控制浮点数的小数点后两位输出。

A. {.2}　　B. {:.2}　　C. {.2f}　　D. {:.2f}

2. 什么是算法？算法的基本特征是什么？

3. 编写一个加法和乘法计算器程序。

4. 编写程序，输入三角形的3个边长a、b、c，求三角形的面积area，并画出算法的流程图和N-S结构图。公式为

$$area=\sqrt{S(S-a)(S-b)(S-c)}$$

其中，S=(a+b+c)/2。

5. 编写程序，输入四个数，并求它们的平均值。

6. 从键盘上输入一个大写字母，并将大写字母转换成小写字母后输出。

7. 一年按365天计，以1.0作为每天的能力值基数，每天原地踏步则能力值为1.0，每天努力一点则能力值提高1%；每天更努力则能力值提高2%。一年后，这三种行为收获的成果相差多少呢？