第 3 章　Python 控制结构

学习了 Python 的基本语法和简单的数据类型之后，对于业务逻辑的实现，还需要依赖于 Python 程序控制结构。控制结构就是控制程序执行顺序的结构，Python 程序的基本控制结构主要包括顺序结构、选择结构和循环结构，任何复杂问题的算法都可以由这三种基本结构组合而成。本章从程序和算法、程序的描述方式、程序的基本结构开始，分别讲解 Python 顺序结构、选择结构和循环结构三种基本控制结构的相关知识。

本章学习目标

- 了解算法的概念，程序的三种描述方式。
- 熟悉 Python 程序的三种基本控制结构。
- 掌握数据流程图中基本符号的含义，能够绘制数据流程图。
- 掌握顺序结构的基本语义，能熟练运用顺序结构语句解决实际问题。
- 掌握三种分支结构的基本语义和语法格式，并能熟练运用三种分支结构语句解决实际问题。
- 掌握 for 循环语句和 while 循环语句的基本语义和语法格式，并能熟练运用 for 循环语句和 while 循环语句求解实际问题。
- 掌握 break 语句和 continue 语句的具体作用和用法。
- 掌握 pass 语句的作用。
- 掌握 for 循环语句扩展模式和 while 循环语句扩展模式的具体用法。

3.1　Python 程序的基本结构

3.1.1　程序和算法

1. 程序

程序设计语言是计算机能够识别和理解的语言，是计算机和用户之间进行交互的工具。程序设计语言通常按照特定的语法规则组织计算机指令，使计算机能够自动完成各种运算，并进行相应处理。计算机程序就是指按照一定程序设计语言的语法规则组织起来的一组计算机指令，简称程序。每个计算机程序都用来解决特定的计算问题。

2. 算法

算法属于数学和计算领域的概念，任何完成特定计算功能的一组有序操作都可以称为算法。这组操作可以是单一的计算问题，比如深度优先算法、二叉树算法、最短路径算法等，也可以是多个计算问题的一个组合。算法是一个程序最重要的组成部分，是一个程序的灵魂。

3.1.2　程序的描述方式

到目前为止，程序的描述方式主要有自然语言、流程图和伪代码三种。

1. 自然语言

自然语言描述方式就是指直接使用人类语言描述程序，其中，IPO（Input，Process，Output）描述是自然语言描述方式中的一种。IPO 即输入、处理和输出。输入是一个程序的开始，输出（Output）是程序经过一系列运算之后显示计算结果的方式，处理（Process）是指对用户输入的数据进行计算等操作生成输出结果的程序过程。

在计算机程序中，是否存在没有输入和输出，只有处理过程的程序呢？一起来看看下面这段代码：

```
while(True):
    a=1
```

这是一个无限循环的程序例子，该程序共包含两行语句，其中第一行是 while()条件判断语句，只有当 while 括号内的值为真时程序才顺序执行第二行 a＝1 这条语句，否则跳过该语句。因为 while()语句括号内的值被设定为 True（真值），所以 a＝1 这条循环体语句会一直执行下去，程序无法正常退出。这种无限循环程序尽管没有输入和输出，而且功能也十分有限，但在一些特殊情况下使用时却具有一定的价值。例如，在测试 CPU 或系统性能时，可以通过不间断地执行类似程序来快速消耗 CPU 的计算资源从而辅助完成测试。

IPO 是程序设计的基本方法。下面举一个实例：计算正方形的面积，具体说明如何使用 IPO 方式描述该计算问题。

输入：正方形的边长 a。

处理：计算正方形的面积 s＝ a＊a 或者 $s=a^2$，如果使用后一个公式求解，需要借助幂函数，计算时可以从 Python 库函数中引入。

输出：正方形的面积 s。

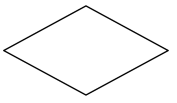
从例子中可以看出，问题的 IPO 描述其实就是对一个计算问题的输入、处理过程和输出的自然语言描述。

2. 流程图

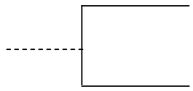
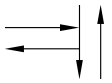
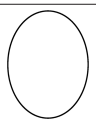
程序流程图又称为程序框图，简称流程图，是程序设计人员用来表达算法的一种工具，也是过程描述和程序分析的一种最基本的方式。程序流程图主要是使用一系列图形、流程线和文字说明来描述程序的基本操作和控制流程。通过流程图，可以直观地体现程序的具体执行顺序。

Python 程序流程图的基本元素包括起止框、判断框、处理框、输入输出框、注释框、流向线和连接点 7 种，具体如表 3-1 所示。

表 3-1　流程图的 7 种基本元素

符　　号	名　　称	表示的含义
	起止框	表示一个程序的开始和结束
	判断框	条件判断框，判断条件是否成立，并根据判断结果选择程序下一步的执行路径

续表

符　　号	名　　称	表示的含义
	处理框	表示赋值等处理过程
	输入输出框	输入框表示数据输入操作，输出框表示数据结果的输出操作
	注释框	对程序的解释说明用注释框来表示
	流向线	流向线通常是用带箭头的直线或曲线来表示程序的执行路径或方向
	连接点	当把一个较大的流程图分割成若干个子流程图的时候，可以用圆形或椭圆形作为连接点，将多个流程图连接成一个整体

为了说明连接点的具体用法，图 3-1 给出了一个流程图示例，图中把一个完整的流程图分解成了两个部分，然后通过连接点 A 将这两个部分连接到一起，再次成为一个程序。

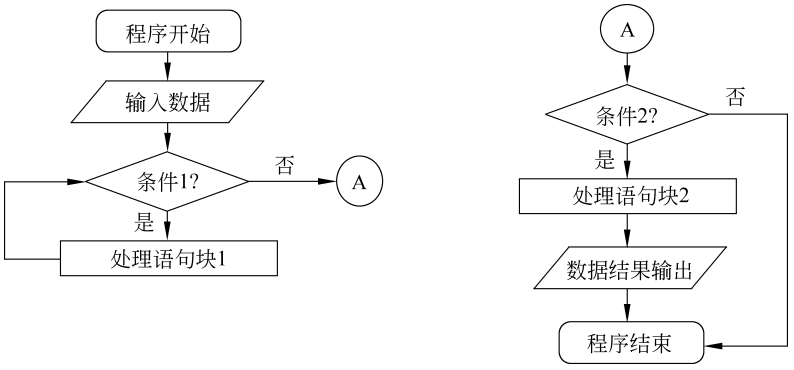


图 3-1　具有连接点的程序流程图

3. 伪代码

伪代码是介于自然语言与编程语言之间的一种算法和程序描述语言。使用伪代码描述程序和算法，不用拘泥于具体编程语言的语法，所以说使用伪代码对程序和整个算法的描述最接近自然语言，但与自然语言描述不同的是，伪代码在描述算法时保持了程序的结构。

因为 Python 语言语法相对简单，所以在本书中不再对伪代码举例说明，而是直接使用 Python 代码。

3.1.3　程序的基本结构

一个完整的程序一般包含一条或多条语句，在实际问题求解时，有时直接按照问题解决的顺序编写相应的程序即可，有时却需要根据判断条件选择程序执行路径或决定程序是否

重复执行。但不管是简单问题还是复杂问题的求解，都可以由顺序结构、选择结构和循环结构这三种基本结构组合而成，所以把这三种基本结构称为 Python 程序的基本控制结构。程序设计的基础就是采用这三种基本结构实现任何单入口、单出口的程序。

1. 顺序结构

顺序结构是一种最简单的控制结构，在顺序结构中，程序按照编写顺序依次执行。顺序结构的流程图如图 3-2 所示，其中语句块 1 和语句块 2 均表示一条或一组顺序执行的语句。

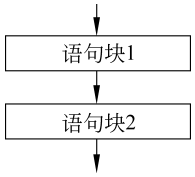
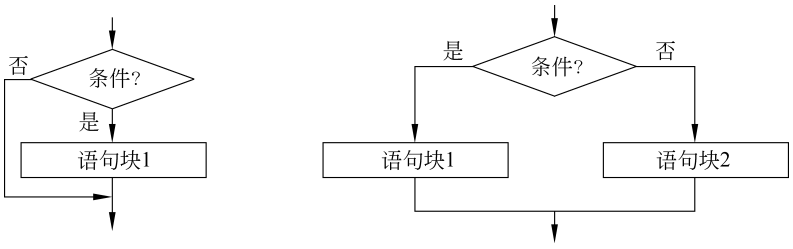


图 3-2　顺序结构的流程图

2. 选择结构

选择结构又称为分支结构。在选择结构中，当程序执行到某条语句时，会根据条件判断结果选择不同的路径进而执行相对应的程序段，如图 3-3 所示。根据分支路径上的完备性，分支结构又可以细分为单分支结构、二分支结构和多分支结构。单分支结构如图 3-3(a)所示，当条件成立时，执行语句块 1，当条件不成立时则跳过语句块 1，直接执行后续语句；二分支结构如图 3-3(b)所示，当条件成立时，执行语句块 1，否则执行语句块 2；多分支结构可以由若干二分支结构或单分支结构组合形成。



(a) 单分支结构　　(b) 二分支结构

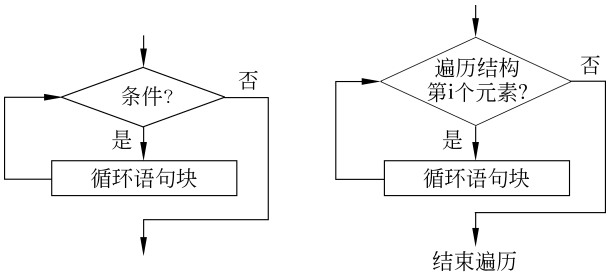
图 3-3　分支结构的流程图

选择结构是程序根据条件判断结果选择不同向前执行路径的一种运行方式。

3. 循环结构

循环结构又称为重复结构。在循环结构中，程序根据条件判断结果选择一条或多条语句重复执行若干遍。循环结构根据循环体触发条件的不同，可以分为条件循环结构和遍历循环结构两种，具体如图 3-4 所示。其中，在图 3-4(a)条件循环结构中，程序顺序执行到条件判断语句时，会根据条件判断结果决定是否执行后面的循环语句块，当条件成立时则执行循环语句块，执行完该语句块之后，继续返回条件语句位置，判断条件是否成立，如果成立，则重复执行循环语句块，然后继续返回判断条件，如果条件不成立，则跳过循环语句块，执行后续语句。在图 3-4(b)遍历循环结构中，程序顺序执行到遍历循环语句关键字时，先取遍历结构第 1 个元素，然后执行循环语句块，接着向前返回继续取遍历结构第 2 个，第 3 个，……，第 i 个元素，重复执行循环体语句，直到所有的遍历元素全部取出并执行循环体语句块后结束循环。

循环结构是程序根据条件判断结果向后反复执行的一种运行方式。



(a) 条件循环结构　　(b) 遍历循环结构

图 3-4　循环结构的流程图

3.1.4　程序基本结构实例

下面给出 3 个微实例，并分别通过自然语言、流程图和直接编写 Python 代码三种不同的描述方式来具体介绍顺序结构、选择结构和循环结构这三种基本程序结构。

【例 3-1】　已知圆的半径 r，计算圆的周长 l 和面积 S。

(1) 计算圆的周长可以使用周长公式：周长＝2 * 圆周率 * 半径 r。

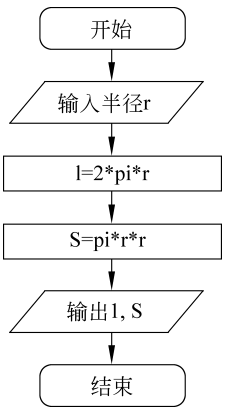


图 3-5　流程图描述计算圆的周长和面积

(2) 计算圆的面积需要使用圆的面积公式：面积＝圆周率 * 半径 r * 半径 r。

(3) 根据题意，该计算问题属于最简单的顺序结构。

下面分别通过 IPO 方式、流程图方式和直接编写 Python 程序代码的方式来具体描述该问题。

① IPO 方式描述。

输入：圆的半径 r。

处理：计算圆的周长 l＝2 * pi * r，圆的面积 S＝pi * r * r，这里的 pi 为圆周率，可以通过 Python 库函数得到。

输出：圆的周长 l、圆的面积 S。

② 流程图方式描述。

流程图方式描述如图 3-5 所示。

③ Python 程序代码方式描述

具体如下：

```
from math import pi                          #从 Python 库 math 中引入 pi
r=eval(input("请输入圆的半径: "))              #输入一个数值赋给 r
l=2 * pi * r                                 #计算圆的周长
S=pi * r * r                                 #计算圆的面积
print("圆的周长为: ",l,"圆的面积为: ",S)       #输出圆的周长和面积
```

【例 3-2】　给定一个整数 N，判断该数的奇偶性并输出。

(1) 判断一个整数 N 的奇偶性就是判断这个整数 N 是否能够被 2 整除，如果能够被 2 整除，就说明 N 是偶数，如果不能就说明 N 是奇数。

(2) 该问题所采用的算法需要包含两种情况,可以使用选择结构中的二分支结构来实现。

下面分别通过 IPO、流程图和直接编写 Python 代码的方式来具体描述该问题。

① IPO 方式描述

输入:整数 N。

处理:判断表达式 N%2==0 的真假。

输出:如果表达式 N%2==0 的结果为真,则输出 N 是偶数,否则输出 N 是奇数。

② 流程图方式描述

流程图方式描述如图 3-6 所示。

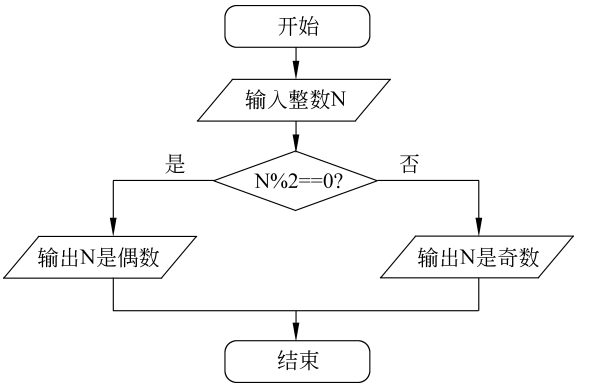


图 3-6　流程图描述判断一个整数的奇偶性

③ Python 程序代码方式描述

具体如下:

```
N=eval(input("请输入一个整数: "))      #输入一个整数赋给 N
if(N%2==0):                            #判断 N 是否能够被 2 整除
    print(N,"是偶数")
else:
    print(N,"是奇数")
```

【例 3-3】 计算 1～N 的累加和。

(1) 假定用变量 s 来存储累加和,通过分析,可以将问题中计算 1 到 N 的累加和的算式表示成 s=((((((((((0+1)+2)+3)+4)+5)+6)+7)+8)+9)+10)+…+N),当 N 的值比较小时,比如 10,用顺序结构即可求解。虽然这种算法简单,但是当 N 的值为 100,甚至是 1000 的时候,这种顺序结构的方法显然太过烦琐,是不可取的。

(2) 通过分析题目可以发现一个规律:题目中的计算只有一种操作,就是累加。设定累加和为 s,在没有累加之前,s 的初始值为 0,需要累加的数是 i,i 的初始值为 1。在算法中,需要重复执行 s 与 i 相加,并把结果写回 s,即 s=s+i,并且每执行完该条语句后,都需要对 i 做加 1 运算,即 i=i+1,这两条语句需要反复执行 N 次,符合循环语句的特点,所以应该借助循环结构来实现。

下面分别通过 IPO、流程图和直接编写 Python 代码的方式来描述该计算问题。

① IPO 方式描述

输入：整数 N。

处理：s＝0,i＝1。

判断：i 是否小于等于 N。

处理：如果 i 的值小于等于 N,则重复执行 s＝s＋i,i＝i＋1。

输出：当 i 的值大于 N 时,输出 s 的值。

② 流程图方式描述

流程图方式描述如图 3-7 所示。

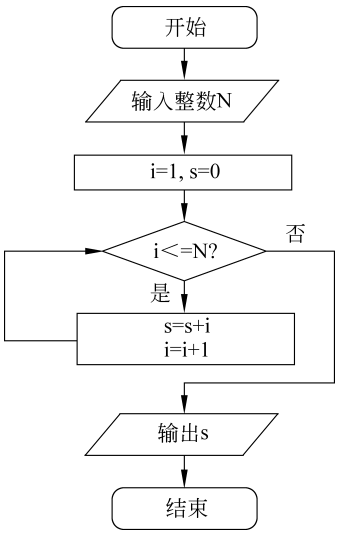


图 3-7　流程图描述求 1～N 的累加和

③ Python 程序代码方式描述

具体如下：

```
N=eval(input("输入 N 的值: "))          #输入 N 的值
i=1                                     #给循环变量 i 赋初值 1
s=0                                     #累加和 s 赋初值 0
while i<=N:                             #给出循环条件判断语句
      s=s+i                             #s 与 i 相加,并把结果写回 s
      i=i+1                             #循环变量进行加 1 运算
print("1～",N,"的累加和是: ",s)          #输出 1～N 的累加和 s
```

由上述三个微实例可以发现,IPO 描述、流程图描述和直接编写 Python 代码描述作为解决实际计算问题的三种主要描述方式,其细致程度逐步递进。IPO 描述重点在于程序结构的划分,明确区分了程序的输入输出关系,在进行处理算法描述时主要采用自然语言。流程图则侧重于描述整个程序的具体流程关系,有助于明确算法的具体操作和执行过程,其细致程度比自然语言更进一步。而 Python 代码的描述最为细致,它体现了问题的最终求解程序。一般而言,在实际应用中,对于功能简单的问题,可以直接采用编写 Python 代码的方式求解,而对于大型复杂的问题,则可以采用 IPO(自然语言)描述或绘制流程图的方式进行描述。

3.2 Python 的顺序结构

顺序结构是最简单的程序结构，也是最常用的程序结构，只要按照解决问题的顺序写出相应的语句并执行就可以对问题求解。

3.2.1 顺序结构语句

在顺序结构中，语句和语句、程序段与程序段之间严格按照程序编写的先后顺序进行执行，因此可以说它是自上而下，完全按照时间顺序执行的。例如由语句 A 和语句 B 构成的顺序结构代码，编写顺序可以是下面的代码 1，也可以是代码 2：

代码 1：

```
语句 A;语句 B
```

代码 2：

```
语句 B;语句 A
```

在上述两种结构中，代码 1 执行时先执行语句 A，然后再执行语句 B；代码 2 在执行时先执行语句 B，然后再执行语句 A。在通常情况下，语句 A 和语句 B 之间执行的先后顺序会对程序的最终结果产生一定的影响。例如下面这段程序代码：

```
x=20            #语句 A
y=x+30          #语句 B
x=y+20          #语句 C
```

在执行语句 A 时，首先 x 被赋值为 20；然后执行语句 B，此时，利用了 x 的值(20)，因此 y 被赋值为 50(20＋30)；继而执行语句 C，再次利用 y 的值(50)，所以执行完语句 C 后，x 的值变为 70(50＋20)。但是，如果将这段程序代码的执行顺序调换一下，比如调换成下面所示的执行顺序：

```
x=20            #语句 A
x=y+20          #语句 B
y =x +30        #语句 C
```

在执行语句 A 时，首先 x 被赋值为 20；然后执行语句 B，此时，因为 y 没有被定义，所以程序终止执行，系统抛出如下异常提示：

```
Traceback (most recent call last):
    File "D: /Program Files/Python38-32/example/b.py", line 2, in <module>
        x=y+20              #语句 B
NameError: name 'y' is not defined
```

但是，在一些特殊情况下，语句 A 与语句 B 之间执行的先后顺序并不会对程序的结果产生任何影响。例如下面这三条程序代码：

```
x=20          #语句 A
y=30          #语句 B
z=50          #语句 C
```

因为语句 A、B、C 之间不存在必然的因果关系，所以不管这三条语句在程序中如何放置，例如“y＝30；z＝50；x＝20”，或者“z＝50；x＝20；y＝30”，系统都不会抛出异常，也不会影响代码的运行结果。

在上述程序代码中，如果把语句 A、B、C 分别换成程序段 A、程序段 B 和程序段 C，先执行完程序段 A 后再执行程序段 B，然后执行程序段 C，则程序段 A、程序段 B 和程序段 C 也是顺序结构；同样，把程序段 A 和程序段 B 看成一段程序（程序段 A1），先执行完程序段 A1 后再执行程序段 C，则程序段 A1 和程序段 C 也是顺序结构。

顺序结构可以独立使用构成一个简单的完整程序。

3.2.2 顺序结构实例

【例 3-4】 交换两个变量的值。

交换两个变量 a 和 b 的值，首先想到的语句是 “a＝b；b＝a”。接下来试着在 Python 环境下输入下面几条语句，看看会出现什么情况。

```
a=20          #定义变量 a
b=30          #定义变量 b
a=b           #把变量 b 的值赋给 a
b=a           #把变量 a 的值赋给 b
print(a,b)    #输出变量 a 和 b 的值
```

通过执行上面的语句，发现输出变量 a 和 b 的值均是 30，并没有实现交换，因为程序第三条语句将变量 b 的值赋给了变量 a 之后，a 的值就也变成了 30，接着第四条语句将变量 a 的值赋给变量 b，也就是把 30 赋给 b。

所以，交换两个变量 a 和 b 的值，不能简单地将变量 a 的值赋给 b，将变量 b 的值赋给 a，为什么呢？打个比方，有一瓶水和一瓶醋，现在想要将两个瓶中的液体进行交换，如果直接将水倒入醋瓶，或将醋倒入水瓶，则只能得到水和醋的混合液，而不能实现水和醋的交换。因此，要想实现交换并分离水和醋，必须借助一个空瓶子作为周转，第一步先将水（或醋）倒入空瓶子中，第二步把醋（或水）倒入腾空的瓶子里，最后再把空瓶子中的水（或醋）倒入另外一个腾空的瓶子中，从而实现水和醋的交换，如图 3-8 所示。同理，要想实现变量 a 和 b 的值的交换也必须借助另外一个临时变量 t，具体算法流程图如图 3-9 所示。

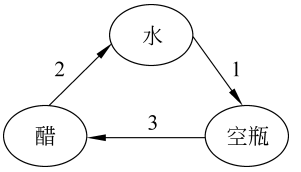


图 3-8　交换水和醋两瓶液体的流程图

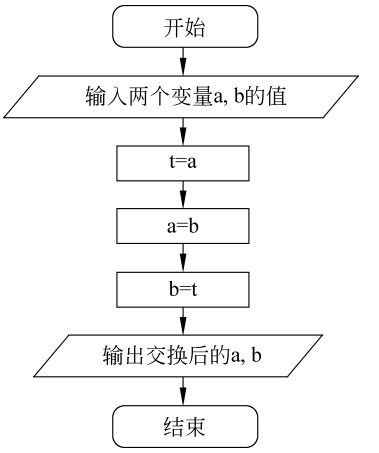


图 3-9　交换两个变量值的流程图

程序代码如下：

```
a=eval(input("请输入变量 a 的值: "))        #输入变量 a 的值
b=eval(input("请输入变量 b 的值: "))        #输入变量 b 的值
t=a                                         #将变量 a 的值存入临时变量 t 中
a=b                                         #将变量 b 的值存入变量 a 中
b=t                                         #将临时变量 t 的值存入变量 b 中
print("交换后 a,b 两个变量的值为: ",a,b)    #输出交换后的变量 a,b 的值
```

代码运行结果：

```
=========
请输入变量 a 的值: 17
请输入变量 b 的值: 11
交换后 a,b 两个变量的值为: 11 17
>>>
```

【例 3-5】 聪明的一休。

从前，有位将军听说一休很聪明，就给他出了一道考题："我昨天请客，客多，碗少，饭碗每人一个，菜碗两人共用一个，汤碗三人共用一个，一共用了 220 个碗。昨天一共来了多少位客人?"一休琢磨了一会儿，微笑着说："一共有 120 位客人就餐。"请编程判断一下一休计算的客人数是否正确。

要计算客人数，可以首先假定客人数为 x，根据题意，可以得到每人使用的碗数为 $\left(1+\frac{1}{2}+\frac{1}{3}\right)$，而所有客人使用碗的总数 s＝220，所以客人数可以利用以下的公式计算得出：

$$x=\frac{220}{1+\frac{1}{2}+\frac{1}{3}}$$

程序代码如下：

```
s=220                    #将碗的总数赋值给变量 s
a=1+1/2.0+1/3.0          #计算每人使用碗的总数
x=s/a                    #计算客人总数
print(x)                 #输出客人总数
```

运行程序，计算得出就餐客人总数为 120，所以一休计算出的客人数是正确的。

【例 3-6】 编写程序，根据下列公式计算存款到期时的本息税前合计，输出时保留两位小数。

$$sum = money * (1 + rate)^{year}$$

根据题意，用变量 sum 来存放本息合计金额，用变量 money 来存放实际存款金额，用变量 year 来表示具体的存期，用变量 rate 来表示年利率。

程序代码如下：

```
from math import pow                          #从 Python 库引入 pow()函数
money=float(input("请输入存款金额 money: "))
rate=float(input("请输入年利率 rate: "))      #float()函数将输入的数据转换成浮点型数据
year=int(input("请输入存期 year: "))          #int()函数将输入的数据转换成整型数据
sum=money*pow((1+rate),year)                  #计算本息合计 sum 的值
print("到期本息合计为{: .2f}".format(sum))    #{.2f}表示输出时保留两位小数
```

代码运行结果：

```
=======
请输入存款金额 money: 50000
请输入年利率 rate: 0.03
请输入存期 year: 3
到期本息合计为 54636.35
>>>
```

3.3 Python 的选择结构

选择结构也就是条件判断结构，又称为分支结构，是 Python 程序控制结构中的一类重要结构，其主要作用是根据判断条件的结果选择程序的执行流程。常见的分支结构有单分支结构、二分支结构、多分支结构及嵌套的分支结构。

3.3.1 if 单分支结构

1. if 语句的语法格式

单分支结构即 if 语句，是 Python 中最简单的分支结构。if 语句由关键字 if、条件表达式和执行语句块三部分组成，其基本语法格式如下：

```
if 条件表达式:
    语句块
```

说明：

(1) if 后面的条件表达式不需要用括号括起来，但后面的冒号“：”不可缺少，它表示一个语句块的开始。

(2) 执行语句块也不需要用括号括起来，但根据 Python 的语法格式，语句块在书写时必须要做相应的缩进，以此来表示语句块与 if 语句之间的包含关系。通常情况下以 4 个空格作为缩进单位。

(3) 语句块是当 if 条件表达式的值为真时执行的一个或多个语句序列，当 if 条件满足的情况下需要执行多条语句时，只要保持多条语句具有相同的缩进即可，也就是说，连续的代码如果缩进相同，则说明这些代码属于同一个代码块，在 Python 程序中属于同一层次，执行时作为一个整体语句。

if 语句在执行时，首先判断条件表达式的值，只有当表达式的值为 True(真)或其他与 True 等价的值时，语句块才会被执行，如果表达式的值为 False(假)，表示条件不满足，则会跳过该语句块中的语句继续向后执行。所以说，if 语句中语句块是否被执行完全依赖于条件表达式的判断结果，但无论什么情况，程序都会转到 if 语句后与 if 语句同级别的下一条语句(如果有)或结束(如果没有)。

if 单分支结构的具体流程图如图 3-10 所示。

下面来看几个具体的单分支结构实例。

2. if 语句实例

【例 3-7】 输入两个字符，判断两个字符的大小，并按由大到小的顺序排列后输出这两个字符。

(1) 假设输入的两个字符分别存放在变量 a，b 中。

(2) 如果由大到小输出的顺序是 a、b，那么当 a＞b 时，直接输出 a，b 即可；当 a＜b 时，则需要首先交换 a 和 b 的值，然后再输出。

该问题对应的数据处理流程图如图 3-11 所示。

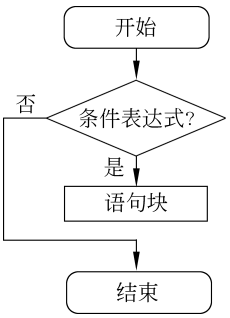


图 3-10　if 单分支结构流程图

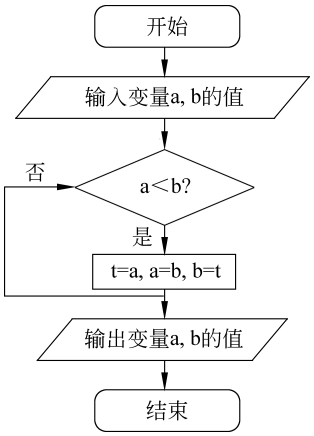


图 3-11　判断两个字符大小的数据处理流程图

程序代码如下：

```
a=input("请输入字符 a 的值: ")                    #输入字符 a 的值
b=input("请输入字符 b 的值: ")                    #输入字符 b 的值
if a<b:                                          #如果 a<b,则交换变量 a 和 b 的值
    t=a                                          #把变量 a 的值赋值给临时变量 t
    a=b                                          #把变量 b 的值赋值给临时变量 a
    b=t                                          #把临时变量 t 的值赋值给变量 b
print("按从大到小的顺序输出 a,b 的值为: ",a,b)      #从大到小输出 a 和 b
```

代码运行结果：

```
==========
请输入字符 a 的值: a
请输入字符 b 的值: t
按从大到小的顺序输出 a,b 的值为: t a
>>>
```

例 3-7 展示了用字符进行条件比较的例子。在 Python 语言中，当字符或字符串作为条件比较时，其比较本质上是字符串对应的 Unicode 编码的比较，因为字符或字符串的比较是按照字典顺序进行的。其中，数字字符对应的 Unicode 编码比英文大写字母小，而英文小写字母对应的 Unicode 编码又比大写字母大。以下再给出一个具体的例子加以说明：

```
>>>'A'>'B'
False
>>>'F'<'f'
True
>>>'9'<'A'
True
>>>'9'>'2'
True
>>>10<2
False
>>>
```

【例 3-8】 从键盘输入一个数，如果这个数为正数，则输出；否则不输出。

定义变量 x 来存放从键盘输入的数。对于变量 x 的值，当 x＞0 时，表示 x 是正数，输出 x 的值，否则不输出。

程序代码如下：

```
x=eval(input("请输入一个数 x 的值: "))          #输入一个数赋给变量 x
    if x>0:                                     #判断 x 是否大于 0
    print("输入的数",x,"是正数")                 #输出正数 x
```

运行程序，当输入一个整数 8 时，if 条件表达式的值为真，程序执行 print 语句，代码运

行结果为：

```
======
请输入一个数 x 的值: 8
输入的数 8 是正数
>>>
```

如果输入的是一个小于或等于 0 的数时，if 条件表达式的值为假，程序不执行任何语句。

【例 3-9】 编写程序求绝对值。输入一个整数 N，输出 N 的绝对值。

(1) 因为正数或 0 的绝对值是它本身，负数的绝对值是它的相反数，所以输出数 N 的绝对值时，首先需要判断数 N 是否为负数。

(2) 判断整数 N 是否为负数，可以用 if 条件表达式 N<0 是否为真来表示。

程序代码如下：

```
N=int(input("请输入整数 N 的值: "))      #输入整数 N 的值
if N<0:                                  #判断整数 N 是否是负数
      N=-N                               #负数的绝对值是它的相反数
print("整数 N 的绝对值是: %d"%N)          #输出整数 N 的绝对值
```

运行程序，当输入整数 8 时，程序中 if 条件表达式的值为 False，所以程序直接执行 print 输出语句，代码运行结果：

```
========
请输入整数 N 的值: 8
整数 N 的绝对值是: 8
>>>
```

当输入整数－12 时，程序中 if 条件表达式的值为 True，所以程序首先执行语句 N＝－N，然后结束 if 分支语句，执行 print 输出语句，代码运行结果：

```
========
请输入整数 N 的值: -12
整数 N 的绝对值是: 12
>>>
```

3.3.2 if-else 二分支结构

1. if-else 语句的语法格式

简单的 if 语句只规定了当条件为 True 或相当于 True 时程序要执行的语句块，但有时候还需要定义当条件为 False 时执行的语句块，即当满足条件时执行一个分支，条件不满足时，执行另外一个分支，这种结构就是二分支结构，Python 使用 if-else 语句来实现二分支结构。

if-else 语句的基本语法格式如下：

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

说明：

(1) if 和 else 属于同一个层次，在书写时缩进要一致，即要左对齐。

(2) if 表达式后面和 else 语句后面的“：”都不可缺少。

(3) 语句块 1 是当 if 条件满足时执行的一个或一组语句序列，语句块 2 是当 if 条件不满足时执行的一个或一组语句序列。语句块 1 和语句块 2 在程序中要保持相同的缩进。

当程序的流程执行到二分支结构时，首先会判断 if 语句后面条件表达式的值，当表达式的值为 True 或其他与 True 等价的值时，选择语句块 1 执行；否则当表达式的值为 False 时，选择语句块 2 执行。也就是说，语句块 1 或语句块 2 只有一个会被执行，然后执行 if-else 语句后面的其他语句(如果有)或者结束程序的执行(如果没有其他语句)。

二分支结构的具体流程图如图 3-12 所示。

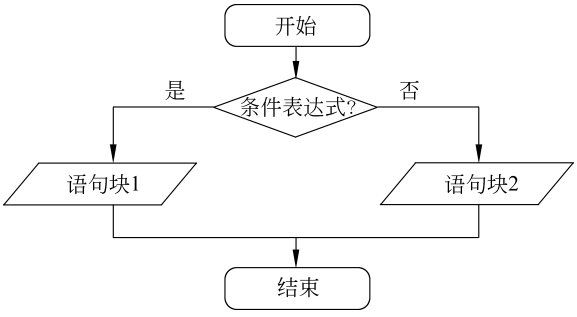


图 3-12　if-else 语句数据处理流程图

下面来看几个具体的二分支结构实例。

2. if-else 结构实例

【例 3-10】 从键盘输入一个用 24 小时制表示的时间，然后把它转换成 12 小时制表示的时间并输出。例如输入 14:20(14 点 20 分)，输出 2:20PM。

(1) 可以假设输入的小时用变量 h 表示，分钟用变量 m 表示。

(2) 根据 24 小时制时间的特点，h 的取值为 1～24，当 h 的值小于等于 12 时，表示时间是上午(AM)，直接输出并说明是上午即可，但当 h 的值大于 12 时，表示时间是下午(PM)，输出的小时则应该为 h－12。

根据分析，该问题的数据处理流程图如图 3-13 所示。

程序代码如下：

```
h=eval(input("请输入 24 小时制小时值: "))      #输入 24 小时制时间小时
m=eval(input("请输入 24 小时制分钟值: "))      #输入 24 小时制时间分钟
if h>12:                                       #判断输入的小时 h 是否大于 12
```

```
    h=h-12                  #大于 12 时,显示处理方法
print("输入的 24 小时制时间是",h+12,": ",m,",转换成 12 小时制时间是 ",h,": ",m,"PM")
else:                       #输入的小时 h 小于等于 12 时
    print("输入的 24 小时制时间是",h,": ",m,",转换成 12 小时制时间是", h,": ",m,"AM")
```

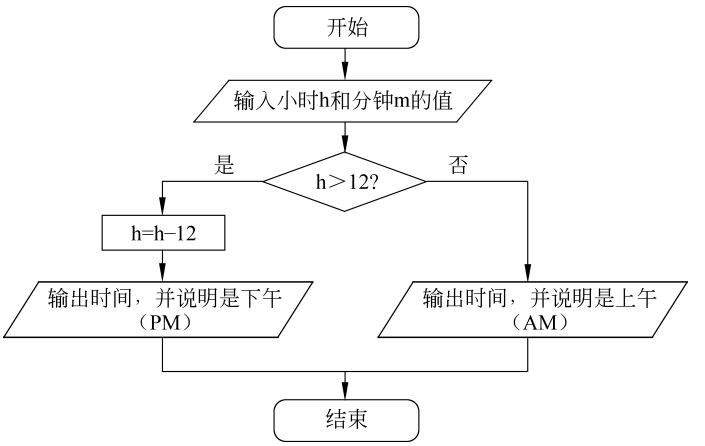


图 3-13　24 与 12 小时制转换数据处理流程图

运行程序,当输入的小时值 h>12 时,满足 if 条件表达式,所以程序执行 if 分支内的语句块,计算 h=h-12,并输出,代码运行结果:

```
======
请输入 24 小时制小时值: 22
请输入 24 小时制分钟值: 50
输入的 24 小时制时间是 22 : 50 ,转换成 12 小时制时间是 10 : 50 PM
>>>
```

当输入的小时值 h≤12 时,if 条件表达式的值为 False,所以程序跳过 if 分支语句,执行 else 分支语句,代码运行结果:

```
======
请输入 24 小时制小时值: 11
请输入 24 小时制分钟值: 23
输入的 24 小时制时间是 11 : 23 ,转换成 12 小时制时间是 11 : 23 AM
>>>
```

【例 3-11】 输入三角形的三边长,求三角形的面积。

(1) 假设将输入的三角形的三条边分别存放在 a,b,c 三个变量中,根据海伦公式,半周长 p=(a+b+c)/2,面积 $s=\sqrt{p*(p-a)*(p-b)*(p-c)}$ 即可求出面积。

(2) 在实际编写程序代码求解时,需要考虑任意输入的三条边是否可以构成三角形。只有当输入的三条边可以构成三角形时,才可以用海伦公式计算求解面积,否则可以输出提示语句。

该问题具体的数据处理流程图如图 3-14 所示。

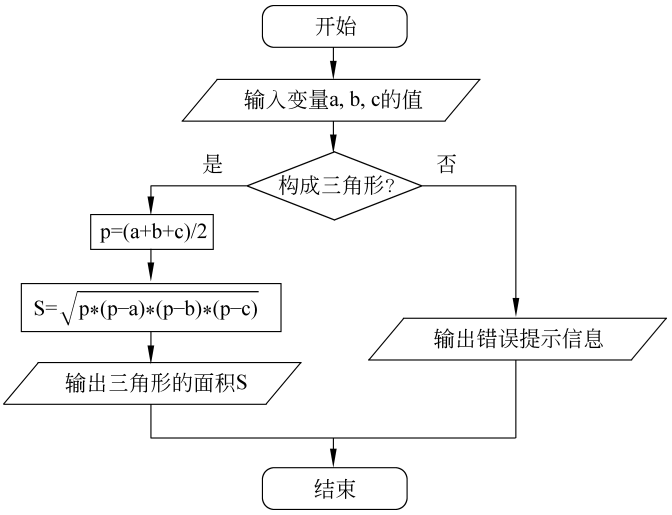


图 3-14　计算三角形面积数据处理流程图

程序代码如下：

```
from math import sqrt                                    #从 math 库中引入 sqrt 计算平方根的函数
print("请输入三角形的三条边 a,b,c 的长度")                  #提示语句
a=float(input("请输入边长 a 的值: "))                    #输入 a 的值,并将输入的字符串转换成浮点数
b=float(input("请输入边长 b 的值: "))                    #输入 b 的值,并将输入的字符串转换成浮点数
c=float(input("请输入边长 c 的值: "))                    #输入 c 的值,并将输入的字符串转换成浮点数
if a+b>c and a+c>b and b+c>a:                            #判断输入的三条边是否可以构成三角形
      p=(a+b+c)/2                                        #求半周长
      s=sqrt(p*(p-a)*(p-b)*(p-c))                        #利用海伦公式计算三角形的面积
      print("三角形的面积是: ",S)                        #输出三角形的面积
else:
      print("输入的三条边无法构成三角形,请重新输入")      #无法构成三角形时给出提示信息
```

运行程序，当输入的三条边分别为 3,4,5 时，if 条件表达式的值为 True，所以程序执行 if 分支语句，计算面积并输出，代码运行结果：

```
========
请输入三角形的三条边 a,b,c 的长度
请输入边长 a 的值: 3
请输入边长 b 的值: 4
请输入边长 c 的值: 5
三角形的面积是: 6.0
>>>
```

当输入的三条边分别为 1,2,3 时，if 条件表达式的值为 False，即输入的三条边无法构成三角形，所以程序执行 else 分支语句，输出提示信息，代码运行结果：

```
========
请输入三角形的三条边 a,b,c 的长度
请输入边长 a 的值: 1
请输入边长 b 的值: 2
请输入边长 c 的值: 3
输入的三条边无法构成三角形,请重新输入
>>>
```

【例 3-12】 输入一个正整数,判断它能否被 3 和 5 整除,如果能,则输出“YES”,否则输出“NO”。

(1) 可以假设输入的整数存放在变量 x 中。

(2) 如果条件表达式“x%3==0 and x%5==0”的值为真,则说明 x 能够被 3 和 5 整除,否则不能。

程序代码如下:

```
x=int(input("请输入一个整数 x: "))    #输入 x 的值,并将输入的字符串转换成整型数据
if x%3==0 and x%5==0:                 #判断 x 能否整除 3,并且整除 5
    print(x,"能够被 3 和 5 整除。")     #x 能整除 3 和 5 时的输出分支
else:
    print(x,"不能被 3 和 5 整除。")     #x 不能整除 3 和 5 时的分支
```

运行程序,当输入整数 15 时,满足能被 3 和 5 整除的条件,即 if 条件表达式的值为 True,所以程序执行 if 分支语句,代码运行结果:

```
========
请输入一个整数 x: 15
15 能够被 3 和 5 整除。
>>>
```

当输入整数 18 时,不满足能被 3 和 5 整除的条件,即 if 条件表达式的值为 False,所以程序执行 else 分支语句,代码运行结果:

```
========
请输入一个整数 x: 18
18 不能被 3 和 5 整除。
>>>
```

3. if-else 语句的简便方式

在 Python 中,二分支 if-else 结构也有一种简洁的表达方式,或称之为三元运算符,并且在三元运算符构成的表达式中还可以再嵌套三元运算符,用来实现与选择结构相似的效果,具体语法格式如下所示:

```
<表达式 1>if <条件>else <表达式 2>
```

其中,表达式 1 或表达式 2 一般为数字类型或字符串类型的一个值,所以该表达式主要

适用于通过判断返回特定值问题的求解。当条件表达式的值为 True 或等价于 True 时，整个表达式的值为表达式 1，否则整个表达式的值为表达式 2。而且，表达式 1 和表达式 2 本身也可以是一个复杂的表达式，比如包含函数调用，甚至可以是三元运算符构成的表达式。在 Python 中，该结构的表达式具有惰性求值的特点。下面以几个具体的使用例子来加以说明：

```
>>>a=8
>>>print(a) if a>=5 else print(0)
8
>>>print(a if a>=5 else 0)                    #运行结果和上条语句一样,但代码含义却不同
8
>>>print(a if a<=5 else 0)
0
>>>b=11 if a>10 else 1                        #赋值运算符的优先级低于三元运算符
>>>b
1
>>>y=math.sqrt(25) if 7>4 else random.randint(1,100)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    y=math.sqrt(25) if 7>4 else random.randint(1,100)
NameError: name 'math' is not defined       #使用 math()函数但没有引入时
>>>import math
y=math.sqrt(25) if 7>4 else random.randint(1,100)
>>>y
5.0                                         #使用 math()函数并引入之后,输出 y 的值
>>>y=math.sqrt(25) if 4>7 else random.randint(1,100)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    y=math.sqrt(25) if 4>7 else random.randint(1,100)
NameError: name 'random' is not defined     #使用 random()函数但没有引入

>>>import random
y=math.sprt(25) if 4>7 else random.randint(1,100)
>>>y
65                                          #使用 random()函数并引入之后输出 y 的值
>>>
```

【例 3-13】 为鼓励居民节约用水，自来水公司采取按居民月用水量分段计费的方式收取水费，居民应交水费 y(元)与居民用水量 x(立方米)之间的对应关系如下所示。编写程序，输入用户的月用水量 x，计算并输出该用户应支付的水费 y。

$$y=f(x)=\begin{cases}\dfrac{4*x}{3}, & 0\leqslant x\leqslant 15\\ 2.5*x-10.5, & x>15\end{cases}$$

根据题意，居民应交水费 y(元)取决于其每月实际用水量 x(立方米)，用水量为 0～15 立方米时使用一种计算方法，用水量大于 15 立方米时，使用另一种计算方法。

程序代码如下：

```
x=float(input("请输入居民的月用水量 x(立方米)"))      #输入 x 的值，并将其转换成浮点数
if x>15:                                              #判断 x 输入的值是否超过了 15
    y=2.5*x-10.5                                      #计算居民每月应交的水费
else:                                                 #x 输入的值满足 0≤x≤15 的条件
    y=4*x/3.0
print("居民月用水量为{},应交水费为{}".format(x,y))   #输出居民的月用水量和应交水费
```

运行程序，当输入用水量为 20 立方米时，if 条件表达式的值为真，所以程序执行 if 分支语句，利用公式 y＝2.5＊x－10.5 计算居民应交水费，代码运行结果：

```
========
请输入居民的月用水量 x(吨)20
居民月用水量为 20.0,应交水费为 39.5
>>>
```

当输入用水量为 12 立方米时，不满足 if 条件表达式的值，所以程序执行 else 分支语句，代码运行结果：

```
========
请输入居民的月用水量 x(立方米)12
居民月用水量为 12.0,应交水费为 16.0
>>>
```

上述问题同时也属于通过判断返回特定值的问题求解，所以可以使用简单表达式来实现，程序代码如下：

```
x=float(input("请输入居民的月用水量 x(立方米)"))
print("居民月用水量为{},应交水费为{}".format(x,2.5*x-10.5 if x>15 else 4*x/3.0)
```

3.3.3 if-elif-else 多分支结构

1. if-elif-else 语句的语法格式

单分支结构和二分支结构最多定义满足两种条件判断问题的解决方法，但是如果在每种条件判断后的执行语句中，又需要根据条件判断结果再次选择程序执行路径时，即出现多重判断，形成多重条件执行结构时，就需要借助多分支结构来解决。

在 Python 中，多分支结构使用 if-elif-else 语句进行描述，其语法格式如下：

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