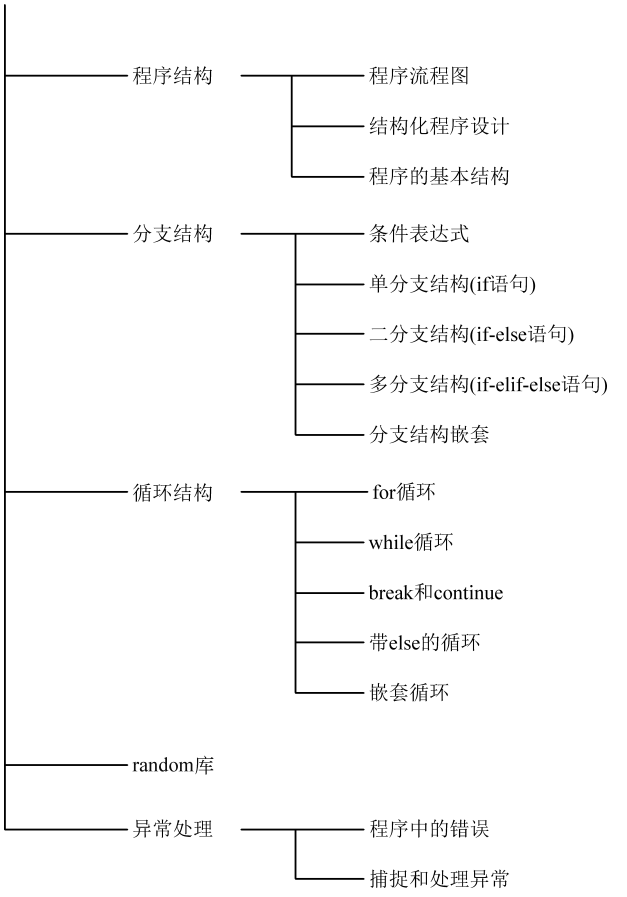
第3章 程序控制结构

学习目标

- 能运用三种基本控制结构分析问题，并绘制程序流程图。
- 能运用分支结构解决选择结构的问题。
- 能运用循环结构解决循环结构的问题。
- 能调用随机函数库解决实际问题。
- 能运用异常处理功能控制程序中的异常情况。

本章主要内容



各例题知识要点

例 3.1　输出三个数中的最大数(单分支结构 if)

例 3.2　判断整数的奇偶(二分支结构 if-else 语句,紧凑型二分支结构)

例 3.3　判断“打鱼”还是“晒网”(二分支结构 if-else 语句)

例 3.4　数学分段函数的计算(多分支结构,math 库)

例 3.5　英制单位英寸与公制单位厘米换算(多分支结构)

例 3.6　计算应付货款(分支嵌套)

例 3.7　计算 $1+2+\cdots+n$ 的值(for 循环)

例 3.8　编程计算 n 的阶乘(for 循环)

例 3.9　计算 $1+1/2+1/3+1/4+\cdots+1/n$(for 循环)

例 3.10　输出所有 3 位水仙花数(for 循环)

例 3.11　求字符串中大写字母的个数(for 循环,字符串遍历)

例 3.12　利用循环将键盘输入的字符串进行反转(字符串遍历)

例 3.13　计算 $1+2+\cdots+n$ 的值(while 循环)

例 3.14　对输入的整数求和(while 循环)

例 3.15　计算 $1+2+3+\cdots+n\geqslant 100$ 时的最小 n(循环,break)

例 3.16　使用循环,求两个数的最大公约数(循环,break)

例 3.17　依次输出字符串"hello,world"中","之外的字符(循环,continue)

例 3.18　判断一个数是否为素数(带 else 的循环)

例 3.19　使用循环,判断字符串是否是回文(带 else 的循环)

例 3.20　百元买百鸡问题(循环嵌套)

例 3.21　输出 10～99 的所有素数(循环嵌套)

例 3.22　掷骰子游戏(循环,random 库)

例 3.23　利用蒙特卡罗算法计算圆周率(循环,random 库)

例 3.24　生成由大写字母、小写字母和数字组成的 6 位随机验证码(循环,random 库)

例 3.25　输入一个整数,输出该数的平方值,如果用户输入的不是整数,则输出提示信息“输入不是整数”(异常处理)

例 3.26　编程实现一个除法计算器(异常处理)

3.1　程 序 结 构

3.1.1　程序流程图

1. 程序与算法

程序(Program)是指一组指示计算机或其他具有信息处理能力的装置,执行动作或

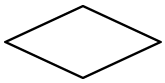
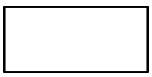
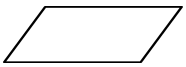
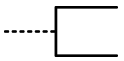
做出判断的指令。计算机程序通常是用高级语言编写源程序，程序包含数据结构、算法、存储方式等，经过语言翻译程序（解释程序和编译程序）转换成机器接受的指令。计算机程序是算法的一种实现，计算机按照程序逐步执行算法，实现对问题的求解。

算法（Algorithm）是为了求解问题而给出的有限的指令序列，每条指令表示一个或多个操作。常用的描述算法的方法有自然语言、流程图、程序设计语言和伪代码等。

2. 程序流程图

程序流程图又称程序框图，是人们对解决问题的方法、思路或算法的一种描述，它利用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式。程序流程图的基本元素包括 7 种，如表 3-1 所示。

表 3-1　程序流程图的基本元素

符号名称	图　形	功　能
起止框		表示程序逻辑的开始或结束
判断框		表示一个判断条件，并根据判断结果选择不同的执行路径
处理框		表示一组处理过程，对应于顺序执行的程序逻辑
输入输出框		表示程序中的数据输入或结果输出
注释框		表示程序的注释
流向线		表示程序的控制流，以带箭头直线或曲线表达程序的执行路径
连接点		表示多个流程图的连接，常用于将多个较小流程图组织成较大流程图

3. 程序设计过程

（1）问题分析：将要解决的问题进行分析、描述程序功能。

（2）设计算法：根据所需的功能，理清思路，设计出解决问题的方法和完成功能的具体步骤，其中每一步都应当是简单的、确定的。这一步也被称为“逻辑编程”。

（3）编写程序：根据前一步设计的算法编写程序。

（4）运行与调试程序：通过编译调试和运行程序，尽可能地排除错误，获得正确的编码和正确的结果。

3.1.2 结构化程序设计

结构化程序设计(Structured Programming)是由荷兰计算机科学家 E. W. Dijkstra 于 1965 年提出,是软件发展的一个重要的里程碑。结构化程序设计是按照一定的原则与原理,组织和编写正确且易读的程序的软件技术。

(1) 结构化程序设计采用自顶向下、逐步求精的程序设计方法。对于小规模程序设计,它与逐步精化的设计策略相联系;对于大规模程序设计,它则与模块化程序设计策略相结合,即将一个大规模的问题划分为几个模块,每一个模块完成一定的功能。

(2) 逐步细化。对复杂问题,应设计一些子目标作为过渡,逐步细化。

(3) 模块化设计。一个复杂问题,肯定是由若干稍简单的问题构成。模块化是把程序要解决的总目标分解为子目标,再进一步分解为具体的小目标,每一个小目标被称为一个模块。模块化提高了编程工作的效率,降低了软件开发成本。

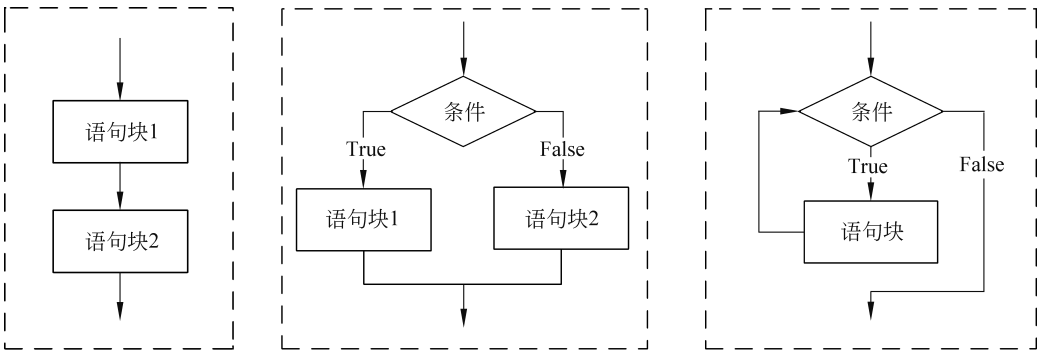
(4) 借助于体现结构化程序设计思想的所谓结构化程序设计语言来书写结构化程序,并采用一定的书写格式以提高程序结构的清晰性,增进程序的易读性。

(5) 程序中的语句执行,任何简单或者复杂的算法都使用顺序、分支、循环三种基本控制结构构造程序,每种结构只有一个入口和一个出口,这是结构化设计的一个原则。

3.1.3 程序的基本结构

计算机的一个程序由若干语句组成,这些语句用来完成特定的任务。程序中的语句可以由顺序结构、分支结构和循环结构三种基本结构组成。

顺序结构:是指程序按照语句顺序自上而下,依次执行的一种运行方式,如图 3-1(a)所示,其中,语句块 1 和语句块 2 表示一个或一组顺序执行的语句。



(a) 顺序结构　　(b) 分支结构　　(c) 循环结构

图 3-1　三种基本结构执行流程

选择结构:根据判断条件的结果而选择不同向前路径的运行方式。如图 3-1(b)所示,其中,语句块 1 是判断表达式返回结果为 True 时执行的语句,语句块 2 是判断表达式

返回结果为 False 时执行的语句。

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。它由循环体中的条件判断继续执行某个功能还是退出循环。如图 3-1(c)所示，其中被反复执行的语句称作“循环体”，判断循环是否继续执行还是退出的条件称作“循环条件”。在 Python 中根据循环体触发条件不同，循环结构又包括条件循环和遍历循环结构。

3.2 分支结构

3.2.1 条件表达式

在分支结构中，程序的执行要根据条件表达式的返回结果来确定要继续执行的语句。Python 中条件表达式可以是单纯的布尔值或变量，也可以是表达式。表达式中可以使用任何能够产生 True 或 False 的语句，形成判断条件最常见的方式是采用关系运算符和逻辑运算符，Python 语言中提供的关系运算符有 6 个，如表 3-2 所示，逻辑运算符 3 个，如表 3-3 所示。

表 3-2 关系运算符

运 算 符	对应数学符号	含 义	举 例	结 果
<	<	小于	'ab'<'ac'	True
<=	≤	小于或等于	125<=130	True
>	>	大于	'e'>'g'	False
>=	≥	大于或等于	'e'>='H'	True
==	=	等于	'f'=='F'	False
!=	≠	不等于	'e'!='E'	True

表 3-3 逻辑运算符

运算符	含 义	使 用	描 述	运 算 规 则
and	逻辑与	x and y	两个条件 x 和 y 的逻辑与	x 和 y 两个均为 True 时，结果为 True
or	逻辑或	x or y	两个条件 x 和 y 的逻辑或	x 和 y 至少有一个为 True 时，结果为 True
not	逻辑非	not x	条件 x 的逻辑非	x 为 True(False)时，结果为 False(True)

在关系运算符中，除了数字可以进行关系运算，字符或字符串也可以用于关系运算。字符串比较本质上是字符串对应的 Unicode 编码的比较，因此，字符串的比较按照字典顺序进行。例如，英文大写字符对应的 Unicode 编码比小写字符小。

3.2.2 单分支结构(if 语句)

Python 中 if 语句用来形成单分支结构,语法格式如下。

```
if 条件表达式 :
    语句块
```

说明:

(1) if 是保留字。

(2) 条件表达式可以是算术表达式、关系表达式、逻辑表达式等任意合法的表达式,条件表达式的返回结果为逻辑值:真(True)或者假(False)。

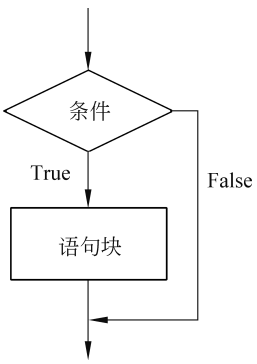


图 3-2 单分支结构控制流程图

(3) 冒号(:)必不可少。

(4) 语句块比 if 语句多缩进若干个字符(通常是 4 个空格或一个 Tab),由相同缩进量的单个或多个语句组成。

单分支结构(if 语句)控制流程图如图 3-2 所示,首先判断 if 后的条件表达式,如果结果为 True(真)说明条件满足,则程序执行语句块中的语句序列,否则不执行语句块,也就是跳过该语句块继续执行后面的语句。

if 语句中语句块是否执行依赖于条件判断的结果,但无论条件返回的结果是 True(真)或 False(假),控制都会转到 if 语句后与该语句同级别的下一条语句。

【例 3.1】 从键盘上输入三个数,输出三个数中的最大数。

【分析】 本例使用单分支结构。定义变量 max 用于存放最大值,暂且假设第 1 个是最大数,将其放到 max 中;把第 2 个数与 max(即第 1 个数)进行比较,如果第 2 个数比 max 大,则将第 2 个数作为最大数放到 max 中,否则什么也不做。此时 max 中存放的是前两个数中的较大数;把第 3 个数与 max(前两个数中的较大数)再进行比较,如果第 3 个数比 max 大,则将第 3 个数作为最大数放入 max,否则什么也不做。此时 max 中存放的是三个数中的最大数。

上述程序流程图如图 3-3 所示。

程序代码:

```
a,b,c=eval(input("请输入三个数: "))
max=a                              #将 a 赋给变量 max
if  b>max:                         #判断 b 是否大于 max
   max=b                           #将 b 赋给变量 max
if  c>max:                         #判断 c 是否大于 max
   max=c                           #将 c 赋给变量 max
print("最大数是{}".format(max))
```

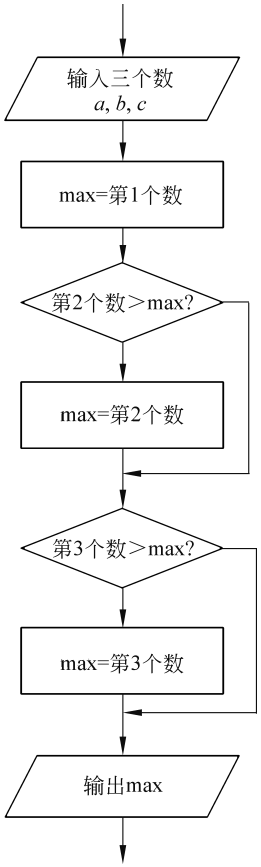


图 3-3　求最大数流程图

程序运行结果：

```
请输入三个数:6,5.2,8.2
最大数是 8.2
```

程序说明：

(1) 使用 input()函数默认输出的是字符串，需要通过 eval()函数进行转换。

(2) a,b,c 三个变量在同一行输入，变量之间用英文半角逗号分隔。

(3) 语句 print("最大数是{}".format(max))中的 format()方法用来定义输出格式。

(4) 本例中使用了两个 if 单分支语句。

3.2.3　二分支结构(if-else 语句)

Python 中 if-else 语句用来形成二分支结构，语法格式如下。

```
if  条件表达式 :
    语句块 1
```

```
else:
    语句块 2
```

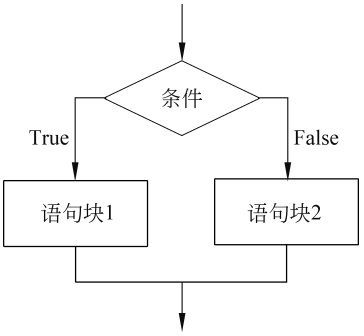


图 3-4　二分支结构控制流程图

说明：

(1) if 与 else 都是保留字，且必须对齐。

(2) else 后必须加冒号(:)。

(3) 语句块 1 与语句块 2 具有相同的缩进量。

二分支结构(if-else 语句)控制流程图如图 3-4 所示，首先判断 if 后的条件表达式，如果结果为 True(真)，说明条件满足，则执行语句块 1 中的语句序列；如果结果为 False(假)，则执行语句块 2 中的语句序列。

【例 3.2】 输入一个整数，判断该数是奇数还是偶数。

【分析】 本例使用 if-else 语句。判断输入数是否能被 2 整除，如果能被 2 整除则为偶数，否则为奇数。

程序代码：

```
n=eval(input())
if n%2==0:
    print("该数是偶数")
else:
    print("该数是奇数")
```

程序说明：通过求余运算%的结果为 0 判断整除。

思考：判断一个数是否是 3 或 5 的倍数，判断条件应如何写？

二分支结构还有一种更简洁的表达方式，紧凑形式的二分支结构，适用于返回特定值的简单表达式。紧凑形式的二分支结构语法格式如下。

```
<表达式 1> if 条件表达式 else <表达式 2>
```

其中，<表达式 1/2>一般是数字类型或字符串类型的一个值。当条件表达式结果为 True 时，返回<表达式 1>，否则返回<表达式 2>。

使用紧凑型二分支结构，改写例 3.2，程序代码如下。

```
n =eval(input())
print("该数是{}".format("偶数" if n%2==0 else "奇数"))
```

【例 3.3】 中国有个典故“三天打鱼，两天晒网”。出自《红楼梦》第九回：“因此也假说来上学，不过三日打鱼，两日晒网，白送些束修礼物与贾代儒。”比喻“做事时断时续，没有恒心，不能坚持”。假如某人从 2020 年 1 月 1 日起开始“三天打鱼两天晒网”，问这个人

在以后的某一天是在打鱼还是晒网？

请编写程序实现：从键盘输入年月日(用逗号分隔)，判断并输出是“打鱼”或者“晒网”。

【分析】 本例涉及 datetime 库和分支结构。首先用 date()函数获得起始日期与当前日期；然后计算两个日期相差的天数；最后用天数除以 5(一个工作周期是 5 天)得到余数，如果余数小于 3(前三天)，则表示“打鱼”，否则表示“晒网”。

程序说明：

```
from datetime import *                          #导入 datetime 库
y,m,d=eval(input("请依次输入年月日："))
start_date = date(2020,1,1)                     #获得起始日期
local_date= date(y,m,d)                         #获得当前日期
t=(local_date-start_date).days                  #计算两个日期相差的天数
x = t %5                                        #计算总天数除以 5 的余数
if x<3 :
   print("打鱼")
else:
   print("晒网")
```

程序运行结果：

```
请依次输入年月日：2021,11,12
打鱼
```

程序说明：

(1) 本例需要导入 datetime 库，调用 datetime 库中的 date()函数构造日期，date()函数有三个参数 year、month 和 day，date(year，month，day)返回日期 year-month-day。

(2) (local_date － start_date).days 返回两个日期相差的天数。

(3) 本例中使用了二分支 if-else 语句。

【扩展】 “三天打鱼，两天晒网”启示我们做事要有持之以恒的毅力。但是要办成一件事，一定要事先进行筹划、安排，这样才能稳步把事情做好。所谓“工欲善其事，必先利其器”就是强调工作方式方法的重要性，只有真正拥有“器”，才能最终做好“事”。勤于学习，善于学习，才能真正保持“器”的“锋利”，实现“事”的“完美”。

计算从某日开始至当前日期，采用“三天打鱼两天晒网”工作模式，计算“打鱼”的总天数和“晒网”的总天数。

3.2.4 多分支结构(if-elif-else 语句)

Python 中 if-elif-else 语句用来形成多分支结构，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
…
[else:
    语句块 n]
```

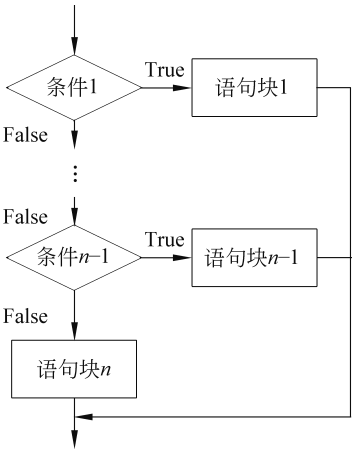


图 3-5　多分支结构控制流程图

说明：

（1）if、elif 与 else 都是保留字。

（2）elif 条件表达式后必须加冒号（:）。

（3）else 书写在最后，也可以省略不写，当省略 else 语句时，说明所有条件表达式不成立时不执行任何语句。

（4）语句块 1、语句块 2 与语句块 n 具有相同的缩进量。

多分支结构是二分支结构的扩展，这种形式通常用于设置同一个判断条件的多条执行路径。多分支结构（if-elif-else）控制流程图如图 3-5 所示，首先判断 if 后的条件表达式 1，如果结果为 True（真）说明条件满足，则程序执行语句块 1 中的语句序列；否则依次判断每个 elif 后的条件，当结果为 True 时，执行该条件下的语句块，同时跳过整个 if-elif-else 结构，执行后面的语句；如果前面的条件表达式均为 False，则执行 else 语句。

【例 3.4】 编写程序，实现如下所示数学分段函数的计算。输入 x，输出“y＝函数值”，结果保留两位小数。

$$f(x)=\begin{cases}|x+2|, & x\leqslant -1\\ \dfrac{1}{2}x^2, & -1<x<2\\ \sin 3x, & x\geqslant 2\end{cases}$$

【分析】 本例涉及 math 库和多分支结构的使用。当不同条件对应不同的输出时，适合使用多分支结构解决；三角函数的计算则需要使用数学库 math。

程序代码：

```
import math
x=eval(input("请输入 x:"))
if x<=-1:
    y=abs(x+2)
elif x<2:
    y=1/2 * x**2
else:
    y=math.sin(3 * x)
print("y={:.2f}".format(y))
```

程序运行结果：

```
请输入 x: 2
y=-0.28
```

程序说明：

(1) 当不满足 $x \leqslant -1$ 时，判断 $-1 < x < 2$ 条件是否满足的条件为 elif $x < 2$，不必再包含 $-1 < x$，因为 elif 中已经包括 $x > -1$ 的情况。当然，如果要按照分段函数公式，可以把条件写完整。如果写完整条件的话，可以使用三个并列的单分支 if 语句，改写如下。

```
import math
x=eval(input("请输入 x:"))
if x<=-1:
    y=abs(x+2)
if -1<x<2:
    y=1/2*x**2
if x>=2
    y=math.sin(3*x)
print("y={:.2f}".format(y))
```

(2) 计算正弦值需调用 math 库，本例中计算绝对值，直接用的 abs()内置函数，如果要使用 math 库中的函数求绝对值，应使用 math.fabs(x+2)。

【例 3.5】 英制单位英寸与公制单位厘米换算。

英寸是英国及英联邦国家使用的一种长度单位，我们国家只在个别场合使用，如相片尺寸等；厘米是国际单位制中长度单位的基本单位；英寸和厘米之间的换算关系为：

1 英寸＝2.54 厘米(1inch＝2.54cm)

编写程序实现英寸与厘米之间的单位换算。如果输入的单位是“英寸”或者 in，则将其转换为厘米输出。如果输入的单位是“厘米”或者 cm 或者“CM”则将其转换为英寸输出；如果输入的单位错误，则输出“单位无效”提示信息。换算后的数值保留两位小数输出。

【分析】 本例涉及字符切片与分支结构。从键盘输入字符串类型的“长度值单位”，由于单位“in”“英寸”“cm”“CM”和“厘米”均为输入字符串的后两位，因此可以用字符切片得到输入的长度单位；用字符切片截取除长度单位以外的字符，得到输入的长度值。

如果截取的长度单位是“in”或“英寸”，则将输入数由英寸转换为厘米输出；否则，如果截取的长度单位是“cm”“CM”或“厘米”，则将输入数由厘米转换为英寸输出；如果截取的长度单位不是“in”“英寸”“cm”“CM”或“厘米”，则输出“单位无效”。

程序代码：

```
length_unit = input("请输入值(单位: 英寸/in/厘米/CM/cm): ")
length=eval(length_unit[0:-2])
unit=length_unit[-2:]
if unit== 'in' or unit == '英寸':
```

```
    print('{}英寸={:.2f}厘米' .format(length,length * 2.54))
elif unit== 'cm' or unit== '厘米' or unit== 'CM':
    print('{}厘米={:.2f}英寸' .format(length,length/2.54))
else:
    print('单位无效')
```

程序说明：

(1) 字符切片 length_unit[0:-2]用来截取 length_unit 字符串中的倒数第 2 个字符之前的所有字符，不包含索引号为-2 的字符。

(2) 字符切片 length_unit[-2:]用来截取 length_unit 字符串中的最后两个字符，包含索引号为-2 的字符。

(3) 本例中使用了多分支 if-elif-else 语句。

程序运行结果 1：

```
请输入值(单位：英寸/in/厘米/CM/cm)：25厘米
25厘米=9.84英寸
```

程序运行结果 2：

```
请输入值(单位：英寸/in/厘米/CM/cm)：30in
30英寸=76.20厘米
```

程序运行结果 3：

```
请输入值(单位：英寸/in/厘米/CM/cm)：10m
单位无效
```

思考：尝试编程实现摄氏度和华氏度之间的温度转换。

3.2.5 分支结构嵌套

分支结构嵌套是指一个分支结构的内部包含另一个分支结构。

Python 中分支结构嵌套的语法格式如下。

```
if  条件表达式 1：
    语句块 1
    if  条件表达式 2_1：
        语句块 2_1
    elif  条件表达式 2_2:
        语句块 2_2
        …
    else:
        语句块 2_n
```

内层 if 语句

```
elif  条件表达式 2:
    语句块 2
…
else:
    语句块 n
```

说明：

(1) 外层 if 语句与内层 if 语句可以是单分支、二分支或多分支结构。

(2) 任何一个语句块中都可以包含更内层的 if 语句。

(3) 同一层的缩进量相同。

【例 3.6】 计算应付货款。从键盘输入订货量和价格。根据订货量大小给以不同的折扣，计算应付货款。订货量在 500 以下，折扣为 3%；订货量在 500 及以上，1000 以下，折扣为 5%；订货量在 1000 及以上，2000 以下，折扣为 8%；订货量在 2000 及以上，折扣为 10%。需要考虑订货量和价格小于 0 的情况，当订货量或价格小于或等于 0 时，输出“输入错误”。

【分析】 首先验证数据的有效性，如果订货量或价格小于或等于 0，则输出错误提示信息。否则根据订货量确定折扣情况。如果订货量在(0,500)，则折扣为 3%；如果订货量在[500,1000)，则折扣为 5%；如果订货量在[1000,2000)，则折扣为 8%；如果订货量在 2000 以上，则折扣为 10%。

程序代码：

```
quantity,price = eval(input("请依次输入订货量和价格: "))
if quantity>0 and price>0:              #判断订货量和价格是否均大于 0
   if quantity < 500:
        d=0.03
   elif quantity <1000:
        d=0.05
   elif quantity <2000:
        d=0.08
   else:
        d=0.1
   pays = quantity * price * (1-d)      #计算应付货款
   print("应付货款为:{:.2f}元".format(pays))
else:
   print("输入错误")
```

程序运行结果 1：

```
请依次输入订货量和价格: 1000,0
输入错误
```

程序运行结果 2：

```
请依次输入订货量和价格：1500,15
应付货款为：20700.00元
```

程序说明：

(1) 本题采用分支嵌套结构，外层使用 if-else 二分支结构，判断订货量和价格的取值是否合法；内层使用 if-elif-else 多分支结构，判断订货量所在范围并确定相应折扣。

(2) 应付货款＝订货量×价格×(1－折扣)。

3.3 循环结构

当需要重复执行某些代码时，可以使用循环语句。例如，需要在屏幕上打印 10 行“HelloWorld”，如果复制、粘贴 10 遍也可以实现该功能，但是这样不符合编程的风格，代码需要简洁高效，所以需要程序能够自动重复执行需要重复的代码，来达到输出 10 行的目的。这里自动重复执行的语句就是循环语句。

Python 的循环语句主要包括 for 和 while 循环。for 循环主要用于已知循环次数的情况；while 循环用于不知道循环次数的情况。

3.3.1 for 循环

for 循环又称为遍历循环，由保留字 for 和 in 组成。语法形式如下。

```
for<循环变量> in <遍历结构>
    <语句块>
```

遍历循环语句从遍历结构中逐一提取元素，放在循环变量中。每提取一次元素，就执行一次语句块，直至遍历完所有元素后结束。从遍历结构中提取几个元素，则语句块就重复执行几次。循环语句中的语句块也叫循环体，循环体就是多次重复执行的部分。循环体可以是一条语句，也可以是多条语句。

按照遍历结构的类型，遍历循环可以分为计数遍历、字符串遍历、列表遍历、文件遍历等几种形式。其中，列表遍历将在第 4 章介绍，文件遍历将在第 6 章介绍。

计数遍历的形式如下。

```
for i in range(m,n,d):
    <语句块>
```

计数遍历由 range()函数产生数字序列进行循环。range(m,n,d)产生的整数数列为 m,$m+d$,$m+2d$,…,$m+xd$，即 m 为起始值、$m+xd$ 为最接近 n 的终止值但不超过 n，d 为步长。

range 可以只有一个参数 n，即产生从 0 到 $n-1$ 的连续整数序列。遍历循环语句从

数字序列 0,1,…,$n-1$ 中逐一提取元素,放在循环变量中。每提取一次元素,就执行一次语句块,直至遍历完所有元素后结束。例如以下程序,循环变量 i 从 0 循环到 4。

```
for i in range(5):
    print(i)
```

程序运行结果:

```
0
1
2
3
4
```

如果 range 包含两个参数(m,n),产生从 m 到 $n-1$ 的连续整数序列。遍历循环语句从数字序列 m,$m+1$,…,$n-1$ 中逐一提取元素,放在循环变量中。每提取一次元素,就执行一次语句块,直至遍历完所有元素后结束。例如以下程序,循环变量 i 从 1 循环到 4。

```
for i in range(1,5):
    print(i)
```

程序运行结果:

```
1
2
3
4
```

如果 range 包含三个参数,第三个参数 d 表示步长,当 d 为正数时,则数列递增。例如通过以下程序,可以输出 1～10 的所有奇数。

```
for i in range(1,11,2):
    print(i)
```

程序运行结果:

```
1
3
5
7
9
```

当 d 为负数，则该数列递减，第一个参数要大于第二个参数。例如通过如下程序，可以从大到小输出 0～10 的偶数。

```
for i inrange(10,0,- 2):
    print(i)
```

程序运行结果：

```
10
8
6
4
2
```

【例 3.7】 使用 for 循环，编程计算 1＋2＋…＋n 的值，n 从键盘输入。

【分析】 利用 range()产生 1～n 的数字序列(初值为 1，终值为 n＋1，步长为 1(可省略))，再利用累加求和的算法计算和值。

程序代码：

```
n=eval(input())
s=0
for i in range(1,n+1):
     s=s+i
print("1~{}的和是{}".format(n,s))
```

程序运行结果：

```
100
1~100 的和是 5050
```

程序说明：

(1) range 产生 1～n 的整数数列，i 依次遍历每个数值。使用累加算法 $s=s+i$，每循环一次，变量 s 在原有的基础上加 i，再赋给 s，从而实现了从 1 到 n 的求和。

(2) 在求和前，需要给 s 赋初值为 0。循环结束后输出结果，print()要与 for 对齐。

思考：尝试编程实现计算 1～n 所有奇数和或偶数和，计算 1～n 所有被 5 整除的数之和。

【例 3.8】 编程计算 n 的阶乘。

【分析】 使用 range 产生 1～n 的数列，通过累乘计算出阶乘。

程序代码：

```
n=eval(input())
f=1
for i in range(1,n+ 1):
     f=f * i
print("{}!={}".format(n,f))
```

程序运行结果：

```
10
10!=3628800
```

程序说明：

(1) 注意求阶乘，存放乘积的变量应赋初值为 1，累乘算法为：$f=f*i$。

(2) 求阶乘可以直接调用 math 库下的函数完成，例如，math.factorial(10)可以求得 10!。

【例 3.9】 计算 $1+1/2+1/3+1/4+\cdots+1/n$，n 从键盘输入，结果保留两位小数。

【分析】 利用 range 函数产生 1～n 的数列作为分母，分子始终是 1，利用累加求和算法求和 $s=s+1/i$。

程序代码：

```
n=eval(input())
s=0
for i in range(1,n+1):
    s=s+1/i
print("1~{}的倒数和是{:.2f}".format(n,s))
```

程序运行结果：

```
100
1~100 的倒数和是 5.19
```

思考：如果计算 $1-1/2+1/3-1/4+\cdots+1/n$，应该如何控制每一项的正负号？

【例 3.10】 输出所有 3 位水仙花数。水仙花数是指 3 位数的各位数字的 3 次方之和等于该数本身的数，即 $abc=a^3+b^3+c^3$。

【分析】 利用 range 产生 100～999 的数列，将每个数的各位数字分离出来，根据水仙花数的特点进行判断。

程序代码：

```
for i in range(100,1000):
    n=i%10
    m=i//10%10
    d=i//100
    if n**3+m**3+d**3==i:
        print(i,end=",")
```

程序运行结果：

```
153,370,371,407
```

程序说明：i 遍历所有的 3 位数 100～999。依次求出 i 的每位数字。

$n=i\%10$：得到 n 是 i 的个位数字。

$m=i//10\%10$：得到 m 是 i 的十位数字。

$d=i//100$：得到 d 是 i 的百位数字。

思考：尝试编程输出所有的四叶玫瑰数(四叶玫瑰数是指四位数各位上的数字的 4 次方之和等于本身的数，即 $abcd=a^4+b^4+c^4+d^4$)。

【例 3.11】 求字符串中大写字母的个数。

【分析】 利用 for 循环遍历字符串，利用字符比较大小或者 isupper()函数判断大写字母，利用累加求和进行计数。

程序代码：

```
n=0
str="Hello World, Hello Python!"
for s in str:
    if "A"<=s<="Z":                    #等价于 if s.isupper()==True:
        n=n+1
print("字符串中大写字母有",n,"个")
```

程序运行结果：

```
字符串中大写字母有 4 个
```

程序说明：

(1) for 循环可以遍历可迭代对象，像前面介绍的 range 函数产生的等差数列，除此之外，Python 中的可迭代对象还有很多，如字符串等。

(2) 本例题利用 for 循环来遍历 str 字符串对象，变量 s 在 for 循环中遍历 str 字符串的每个字符。在循环体中需要判断 s 是否为大写字母，可以像比较数字一样用"A"＜s＜"Z"不等式来比较 s 与“A”和“Z”之间的关系，以此来判断 s 是否为大写字母。

(3) 计算机中任何数据都有自己的值，这个值叫 ASCII 码，A 的 ASCII 码是 65，B 的 ASCII 码是 66，以此类推，Z 的 ASCII 码是 90。所以可以用不等式来表示字符之间的关系。

(4) 除了使用不等式来判断字符的关系，还可以使用 Python 提供的相关函数来判断字符的大小写情况。

s.isupper()函数：判断 s 字符是否为大写字母，如果 s 是大写字母，则 s.isupper()的值为 True；否则为 False。

s.islower()函数：判断 s 字符是否是小写字母，如果 s 是小写字母，则 s.islower()的值为 True；否则为 False。

s.isdigit()函数：判断 s 字符是否是数字字符，如果 s 是数字字符，则 s.isdigit()的值为 True，否则为 False。

(5) 当 s 是大写字母时，执行计数器 $n=n+1$，计数器就是一个累加求和的过程，只

是每次累加的是1,所以 $n=n+1$ 实现了计数器的功能。

【例 3.12】 利用循环将键盘输入的字符串进行反转。

【分析】 不能利用字符串反转函数进行反转,要求利用循环进行字符串反转。首先定义一个新的字符串,然后在 for 循环遍历字符串时,将遍历的每个字符累加放在新字符串中。

程序代码:

```
s = input()
str= ""
for  i  in  s :
    str = i+str
print("原字符串是: ",s)
print("反转后字符串是: ",str)
```

程序运行结果:

```
python
原字符串是: python
反转后字符串是: nohtyp
```

程序说明:该例题通过 i 遍历 s 字符串对象,使用累加求和的思想,str=i+str 每次将新的字符 i 连接在新字符串 str 的前面,从而实现了将 s 字符串反转的功能。

3.3.2 while 循环

for 循环是通过遍历可迭代对象来执行的循环操作。然而在某些情况下,明确需要使用循环结构,但是又没有可迭代对象或者对于循环开始值和结束值不是很明朗的时候,这时就需要采用 while 循环来达到循环的目的。

无限循环也称条件循环,是由条件控制的循环运行方式。无限循环语句用保留字 while 实现,具体形式如下。

```
while <条件> :
    <语句块>
```

当满足条件时,反复执行语句块,直到条件不满足时结束循环。在 while 循环中,需要用户根据题目要求给定条件表达式,并且在循环体中,必须有能使 while 循环趋向于结束的语句,即使条件表达式趋于不成立的语句。

【例 3.13】 利用 while 循环,编程计算 $1+2+\cdots+n$ 的值,n 从键盘输入。

【分析】 该例可以利用 for 循环实现,也可以用 while 循环来实现。用 while 循环时,需要给定条件表达式,循环体内必须有使条件表达式趋于不成立的语句。

程序代码:

```
n=eval(input())
s=0
i=1
while i<=n:
    s=s+i
    i=i+1
print("1~{}的和是{}".format(n,s))
```

程序运行结果：

```
100
1~100 的和是 5050
```

程序说明：使用 while 循环求和时，需要在循环体内有控制循环变量增 1 的语句。通过本例与例 3.7 的对比，体会 for 循环和 while 循环的差异。

思考：尝试利用 while 循环求 1～100 的偶数和。

【例 3.14】 将用户输入的整数进行累加求和，当和大于 100 时，停止输入，求当前的和以及用户输入的次数。

【分析】 本例中循环次数未知，因此采用 while 循环语句。

程序代码：

```
sum=0
n=0
while sum<=100:
    d=int(input())
    n=n+1
    sum=sum+d
print("和={}, 输入了{}次".format(sum,n))
```

程序运行结果：

```
10
25
30
45
和=110, 输入了 4 次
```

程序说明：

(1)“当和大于 100 时，停止输入”，也就是和大于 100 时，停止循环，所以条件表达式应该是“和＜＝100”值为真时，进入循环。

(2) 用户输入、累加求和均放在循环体中。这里的累加求和就是使条件表达式趋于结束的语句。

(3) 累加求和的两个变量 sum 和 n，需要提前赋初值为 0。