第 3 章　流程控制结构

结构化是程序设计应遵循的基本原则，其核心思想是将程序划分为不同的逻辑结构，由这些结构决定程序的执行流程。结构化程序设计有 3 种基本结构，即顺序结构、选择结构和循环结构。通过这 3 种基本结构就可以控制程序的执行流程。顺序结构比较简单，按照执行顺序依次写出语句即可，选择结构和循环结构则需要通过专门的流程控制语句来实现。本章主要讲述 Python 流程控制语句的语法和应用，同时讨论如何在程序中捕获和处理异常。

3.1　选择结构

选择结构是指程序运行时根据特定的条件选择一个分支执行。根据分支的多少，选择结构可以分为单分支选择结构、双分支选择结构和多分支选择结构。根据实际需要，还可以在一个选择结构中嵌入另一个选择结构。使用选择结构可以在程序中对指定条件进行判断，并据此选择不同的代码块来执行。

3.1.1　单分支选择结构

单分支选择结构用于处理单个条件、单个分支的情况。在 Python 中，单分支选择结构可以用 if 语句来实现，其一般语法格式如下。

```
if 表达式:
    语句块
```

其中，表达式表示要测试的条件，其值为布尔值，在该表达式后面必须加上半角冒号。语句块可以是单条语句，也可以是多条语句。语句块必须向右缩进，如果语句块中包含多条语句，则这些语句必须具有相同的缩进量。如果语句块中只有一条语句，可以与 if 语句写在同一行，即在冒号后面直接写出条件成立时要执行的语句，但是一般不建议这样处理。

if 语句的执行流程如下：首先计算表达式的值，如果该值为 True，则执行语句块，然后执行 if 语句的后续语句；如果该值为 False，则跳过语句块，直接执行 if 语句的后续语句。if 语句的执行流程如图 3.1 所示。

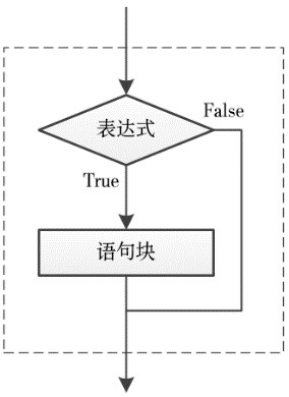

图 3.1　if 语句的执行流程

【例 3.1】从键盘输入两个整数，然后判断它们的奇偶性并输出结果。

【程序代码】

```
num = int(input('请输入一个整数：'))
parity = '奇数'        # 事先设置变量的初始值为“奇数”
```

```
if num % 2 == 0:      # 若 num 能被 2 整除，则修改 parity 变量值
    parity = '偶数'
print(f'{num}是{parity}! ')
```

【运行结果】

```
请输入一个整数：6↵
6 是偶数!
```

再次运行程序，结果如下。

```
请输入一个整数：9↵
9 是奇数!
```

【例 3.2】从键盘输入两个整数，然后按从小到大的顺序输出这两个整数。

【程序代码】

```
num1 = int(input('请输入一个整数：'))
num2 = int(input('请输入另一个整数：'))
if num1 > num2:  # 若 num1 大于 num2，则交换两个变量的值
    num1, num2 = num2, num1
print(f'{num1:d}, {num2:d}')
```

【运行结果】

```
请输入一个整数：200↵
请输入另一个整数：300↵
200, 300
```

再次运行程序，结果如下。

```
请输入一个整数：990↵
请输入另一个整数：600↵
600, 990
```

3.1.2 双分支选择结构

双分支选择结构用于处理单个条件、两个分支的情况。在 Python 中，双分支选择结构可以用 if-else 语句来实现，其一般语法格式如下。

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

其中，表达式表示条件，其值为布尔值，在该表达式后面必须加上半角冒号。语句块 1 和语句块 2 可以是单条语句或多条语句，这些语句块中的语句必须向右缩进，而且语句块中包含的各条语句必须具有相同的缩进量。

if-else 语句的执行流程如下：首先计算表达式的值，如果计算结果为 True，则执行语句块 1，否则执行语句块 2；执行语句块 1 或语句块 2 之后接着执行 if-else 语句的后续语句。if-else 语句的执行流程如图 3.2 所示。

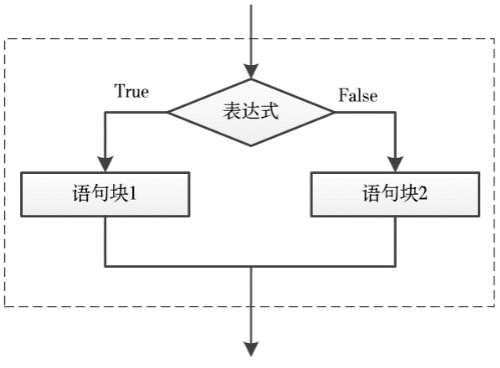

图 3.2 if-else 语句的执行流程

【例 3.3】从键盘输入一个数字，求其绝对值并输出。

【程序代码】

```
x = float(input('请输入一个数字：'))
if x < 0:
    a = -x
else:
    a = x
print('这个数的绝对值为：{0:.3f}'.format(a))
```

【运行结果】

```
请输入一个数字：1.23↵
这个数的绝对值为：1.230
```

再次运行程序，结果如下。

```
请输入一个数字：-358.369667↵
这个数的绝对值为：358.370
```

【例 3.4】从键盘输入两个整数，求出较大的数并输出。

【程序代码】

```
num1 = int(input('请输入一个整数：'))
num2 = int(input('请输入一个整数：'))
if num1 > num2:
    max = num1
else:
    max = num2
print('较大的数为：{0:d}'.format(max))
```

【运行结果】

```
请输入一个整数：123↵
请输入一个整数：456↵
较大的数为：456
```

再次运行程序，结果如下。

```
请输入一个整数：-123↵
请输入一个整数：-456↵
较大的数为：-123
```


3.1.3 多分支选择结构

多分支选择结构用于处理多个条件、多个分支的情况，可以用 if-elif-else 语句来实现，其一般语法格式如下。

```
if 表达式 1:
    语句块 1
elif 表达式 2
    语句块 2
……
elif 表达式 n:
    语句块 n
[else:
    语句块 n+1]
```

其中，表达式 1、表达式 2、…、表达式 n 表示多个条件，它们的值为布尔值，在这些表达式后面要加上半角冒号；语句块 1、语句块 2、…、语句块 n+1 可以是单条语句或多条语句，这些语句必须向右缩进，而且语句块中包含的多条语句必须具有相同的缩进量。

if-elif-else 语句的执行流程如下：首先计算表达式 1 的值，如果表达式 1 的值为 True，则执行语句块 1，否则计算表达式 2 的值；如果表达式 2 的值为 True，则执行语句块 2，否则计算表达式 3 的值，以此类推。如果所有表达式的值均为 False，则执行 else 后面的语句块 n+1。选择执行一个分支之后，程序将接着 if-elif-else 语句的后续语句执行。if-elif-else 语句的执行

流程如图 3.3 所示。

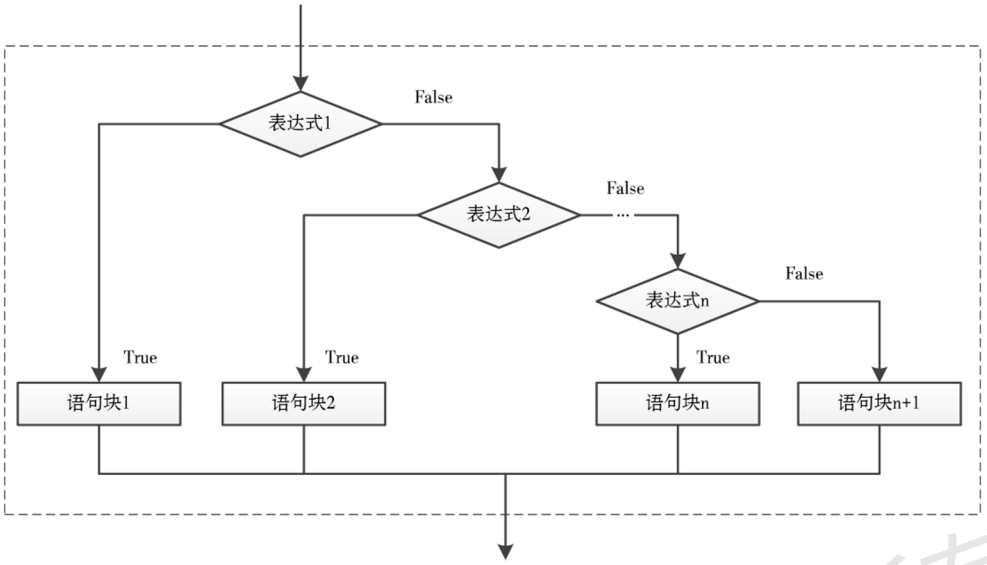

图 3.3　if-elif-else 语句的执行流程

【例 3.5】从键盘输入自变量 x 的值，计算分段函数 y 的值。

$$y=\begin{cases}2x+15 & (x\leqslant 0)\\ 6\sqrt{x}+7x+13 & (0<x<2)\\ 3\sqrt[3]{x}+5x^2-6x & (x\geqslant 2)\end{cases}$$

【程序代码】

```
x = float(input('输入自变量 x 的值：'))
if x <= 0:
    y = 2 * x + 15
elif 0 < x < 2:
    y = 6 * x ** 0.5 + 7 * x + 13
else:
    y = 3 * x ** (1 / 3) + 5 * x * x - 6 * x
print(f'当 {x = :.2f}时，{y = :.2f}')
```

【运行结果】

```
输入自变量 x 的值：-3↵
当 x = -3.00 时，y = 9.00
```

再次运行程序，结果如下。

```
输入自变量 x 的值：1.6↵
当 x = 1.60 时，y = 31.79
```

再次运行程序，结果如下。

```
输入自变量 x 的值：5↵
当 x = 5.00 时，y = 100.13
```

【例 3.6】从键盘输入百分制成绩，计算成绩等级并输出。

【程序代码】

```
score = int(input('请输入百分制成绩：'))
grade = ''
if score >= 90:
    grade = '优秀'
elif score >= 80:
    grade = '良好'
elif score >= 70:
```

```
    grade = '中等'
elif score >= 60:
    grade = '及格'
else:
    grade = '不及格'
print(f'成绩：{score}；等级：{grade}')
```

【运行结果】

```
请输入百分制成绩：86↵
成绩：86；等级：良好！
```

再次运行程序，结果如下。

```
请输入百分制成绩：93↵
成绩：93；等级：优秀！
```

再次运行程序，结果如下。

```
请输入百分制成绩：78↵
成绩：78；等级：中等！
```

【例 3.7】从键盘输入一个算术表达式，计算其值并输出。

【程序代码】

```
print('*****算术运算*****')
x, operator, y = input('请输入 数字 1 运算符 数字 2：').split()
result = 0
if operator == '+':
    result = float(x) + float(y)
    print(f'{x:s} {operator:s} {y:s} = {result:.2f}')
elif operator == '-':
    result = float(x) - float(y)
    print(f'{x:s} {operator:s} {y:s} = {result:.2f}')
elif operator == '*':
    result = float(x) * float(y)
    print(f'{x:s} {operator:s} {y:s} = {result:.2f}')
elif operator == '/':
    result = float(x) / float(y)
    print(f'{x:s} {operator:s} {y:s} = {result:.2f}')
else:
    print('您输入的运算符不支持！')
```

【代码说明】

程序中变量 x 和 y 表示两个数字，变量 operator 表示算术运算符。输入函数 input()的返回值是一个字符串，通过对该字符串调用 split()方法，使用空格将字符串分割成不同的部分，并分别为变量 x、y 和 operator 赋值。

【运算结果】

```
请输入 数字 1 运算符 数字 2：222 + 333↵
222 + 333 = 555.00
```

再次运行程序，结果如下。

```
请输入 数字 1 运算符 数字 2：999 ÷ 3↵
您输入的运算符不支持！
```

3.1.4 条件运算符

Python 中有一个条件运算符，这是一个三目运算符，它对指定条件进行判断并据此返回不同的值，语法格式如下。

```
表达式 1 if 条件 else 表达式 2
```

条件运算符执行时对条件进行测试，如果条件为 True，则返回表达式 1 的值，否则返回表达式 2 的值。

条件运算符有 3 个运算对象，并与这些运算对象共同组成条件表达式，该表达式出现在可以使用表达式的任何位置，如赋值运算符的右边。需要注意的是，使用条件运算符时，不要在 if 语句和 else 语句后面使用冒号。

【例 3.8】从键盘输入 3 个整数，求出其中的最大数并输出。

【程序代码】

```
a = int(input('请输入第一个数：'))
b = int(input('请输入第二个数：'))
c = int(input('请输入第三个数：'))
max = a if a > b else b
max = max if max > c else c
print(f'最大数为：{max:d}')
```

【运行结果】

```
请输入第一个数：333↵
请输入第二个数：999↵
请输入第三个数：666↵
最大数为：999
```

3.1.5 选择结构的嵌套

当使用选择结构控制程序执行流程时，如果有多个条件并且条件之间存在递进关系，则可以在一个选择结构中嵌入另一个选择结构，由此形成选择结构的嵌套。

在内层的选择结构中还可以继续嵌入选择结构，嵌套的深度是没有限制的。

使用嵌套的选择结构时，将根据代码的缩进量来确定代码的层次关系。

选择结构的嵌套主要有以下两种形式。

- 在 if 语句中嵌入 if-else 语句，一般语法格式如下。

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

在这种嵌套结构中，else 与第二个 if 语句配对。

- 在 if-else 语句中嵌入 if 语句，一般语法格式如下。

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

在这种嵌套结构中，else 与第一个 if 语句配对。

【例 3.9】编写一个模拟登录程序。从键盘输入用户名和密码，然后对输入的用户名进行验证，如果用户名正确，再对输入的密码进行验证。如果用户名和密码都与预设值匹配，则登录成功，否则登录失败。

【程序代码】

```
# 设置用户名和密码
USERNAME = 'admin'
PASSWORD = 'zhimakaimen'

# 从键盘输入用户名
username = input('请输入用户名：')
# 验证用户名
if username == USERNAME:
    # 从键盘输入密码
```

```
        password = input('请输入密码：')
        # 验证密码
        if password == PASSWORD:
            print('登录成功！')
            print(f'欢迎{username}进入系统！')
        else:
            print('密码错误，登录失败！')
    else:
        print(f'用户名\"{username}\"不存在，登录失败！')
```

【运行结果】

```
请输入用户名：kk↵
用户名"kk"不存在，登录失败！
```

再次运行程序，结果如下。

```
请输入用户名：admin↵
请输入密码：123456↵
密码错误，登录失败！
```

再次运行程序，结果如下。

```
请输入用户名：admin↵
请输入密码：zhimakaimen↵
登录成功！
欢迎 admin 进入系统！
```

3.2 循环结构

循环结构是控制某个语句块重复执行的程序结构，其特点是在指定条件（循环条件）成立时重复执行某个语句块（循环体）。在 Python 中，可以通过 while 语句和 for 语句来实现循环结构，也可以通过 break 语句和 continue 语句对循环结构的执行过程进行控制，此外，还可以在一个循环结构中使用另一个循环结构，从而形成循环结构的嵌套。

3.2.1 while 语句

while 语句在指定条件成立时重复执行一个语句块，其一般语法格式如下。

```
while 表达式:
    语句块 1
[else:
    语句块 2]
```

其中，表达式表示循环条件，通常是关系表达式或逻辑表达式，也可以是能够转换布尔值的任何表达式；在表达式后面必须添加半角冒号。语句块 1 是将要重复执行的单条语句或多条语句，称为循环体。else 子句是可选的，语句块 2 是循环正常结束后将执行的单条语句或多条语句。

语句块 1 和语句块 2 中包含的语句必须向右缩进。如果语句块中包含多条语句，则这些语句必须具有相同的缩进量。如果循环体只包含单条语句，也可以将这条语句与 while 关键字写在同一行，但一般不建议这样做。

while 语句的执行流程如下：首先计算表达式的值，当计算结果为 True 时，则重复执行循环体内的语句，然后再次计算表达式的值，以此类推。如果表达式的值为 False，则结束循环并执行语句块 2（如果有的话）。while 语句的执行流程如图 3.4 所示。

在 while 语句中，如果循环条件的值恒为 False，则循环体就会一次也不执行。与此相反，如果循环条件的值恒为 True，则循环体将会无限地执行下去，这种情况称为无限循环。

为了适时结束循环过程，需要在循环体内包含能够修改循环条件的值的语句，使该值在某个时刻变为 False，从而结束循环。

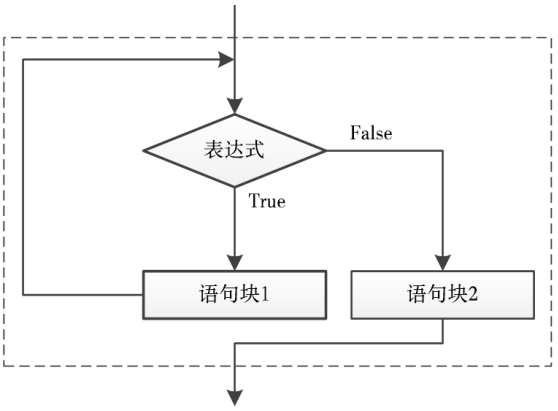

图 3.4　while 语句的执行流程

使用 while 语句时，循环体也可以什么事情都不做。在这种情况下，应在循环体中放置一条 pass 语句作为占位符，以保持程序结构的完整性。pass 语句也称为空语句，语法格式如下。

```
pass
```

【例 3.10】计算前 100 个自然数之和。

【程序代码】

```
i = 1
sum = 0
while i <= 100:
    sum = sum + i;
    i = i + 1;
else:
    print(f'循环结束时 {i = }')
print(f'1 + 2 + 3 + ... + 100 = {sum}')
```

【运行结果】

```
循环结束时 i = 101
1 + 2 + 3 + ... + 100 = 5050
```

【例 3.11】从键盘输入两个正整数，计算它们的最大公约数和最小公倍数。

【程序代码】

```
x = m = int(input('请输入一个正整数：'))
y = n = int(input('请输入另一个正整数：'))
p = m * n
while m % n != 0:
    m, n = n, m % n
print(f'{x}和{y}的最大公约数为：{n}')
print(f'{x}和{y}的最小公倍数为：{p // n}')
```

【运行结果】

```
请输入一个正整数：39↵
请输入另一个正整数：52↵
39 和 52 的最大公约数为：13
39 和 52 的最小公倍数为：156
```

【例 3.12】编写一个打字练习程序，输入 quit 或 exit 退出。

【程序代码】

```
print('*****打字程序*****')
print('输入 quit 或 exit 退出')
line = input('>>>')
while line != 'quit' and line != 'exit':
    line = input('>>>')
else:
```

```
    print('谢谢使用！')
```

【运行结果】

```
*****打字程序*****
输入 quit 或 exit 退出
>>>This is a book about Python programming.↵
>>>生命苦短，我用 Python!↵
>>>quit↵
谢谢使用!
```

3.2.2 for 语句

for 语句用于遍历序列（如字符串、元组或列表）或其他可迭代对象的元素，一般语法格式如下。

```
for 变量 in 序列对象:
    语句块 1
[else:
    语句块 2]
```

其中，变量也称为循环变量，不需要事先进行初始化。序列对象表示要遍历的字符串、列表或元组等。语句块 1 表示循环体，可以包含单条语句或多条语句。当循环体只包含单条语句时，也可以将这条语句与 for 语句写在同一行，但一般不建议这样做。else 子句是可选的，语句块 2 包含循环正常结束时将执行的单条语句或多条语句。

语句块 1 和语句块 2 中包含的语句必须向右缩进。如果语句块中包含多条语句，则这些语句必须具有相同的缩进量。

for 语句的执行流程如下：将序列对象中包含的元素依次赋给循环变量，并针对当前元素执行一次循环体，直至序列中的每个元素都已用过。当序列中的元素用尽时，将执行语句块 2（如果有的话）并结束循环。for 语句的执行流程如图 3.5 所示。

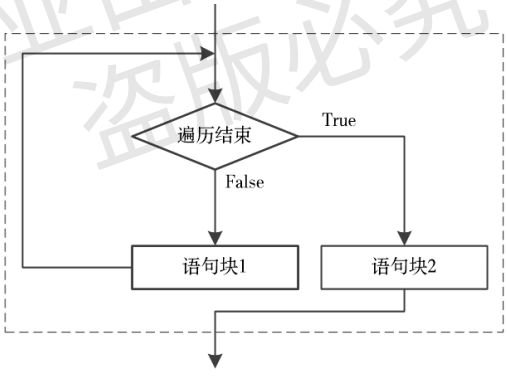

图 3.5　for 语句的执行流程

【例 3.13】4 个人中有一个人打碎了花瓶。A 说不是我，B 说是 C，C 说是 D，D 说 C 撒谎，已知有 3 个人说了真话，请根据以上对话判断是谁打碎了花瓶。

【程序分析】

打碎花瓶的人必是 4 个人中的一个，这个人的编号用变量 who 来表示。将 4 个人的编号组成字符串“ABCD”，并使用 for 语句来遍历该字符串中的每个编号，对每个人的说法进行判断，如果某人说的是真话，则关系表达式的布尔值为 True，可以转换为整数 1。在循环体中，用 if 语句判断是否满足“三人说真话”的条件，如果满足该条件，则输出结果。

【程序代码】

```
for who in 'ABCD':
    if (who != 'A') + (who == 'C') + (who == 'D') + (who != 'D') == 3:
```

```
            print(f'答案：是{who}打碎了花瓶。')
```

【运行结果】

```
答案：是C打碎了花瓶。
```

在实际应用中，经常将 for 语句与 range 对象结合起来使用，用于循环指定的次数，语法格式如下。

```
for x in range(start, stop step):
    语句块
```

其中，x 为循环变量；range(start, stop step)是 Python 中的内置函数，亦即 range 类的构造函数，它返回一个 range 对象，由此生成一个以 step 为步长、从 start 开始（包括）到 stop 终止（不包括）的整数序列。参数 start、stop 和 step 必须是整数，如果省略 step 参数，则步长默认为 1。如果省略 start 参数，则起始值默认为 0。具体示例如下。

```
>>> for i in range(10):
        print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> for i in range(1, 11):
        print(i, end=' ')
1 2 3 4 5 6 7 8 9 10
>>> for i in range(0, 30, 5):
        print(i, end=' ')
0 5 10 15 20 25
```

step 步长可以是正数，也可以是负数。对于正步长，范围 r 中的元素由公式 r[i] = start + step * i 确定，循环条件为 i> = 0 且 r [i] <stop。对于负步长，范围 r 中的元素仍由公式 r [i] =start+step* i 确定，但循环条件为 i> = 0 且 r [i]>stop。如果 step 为零，则引发 ValueError。

下面给出一个使用负步长的例子。

```
>>> for i in range(-1, -11, -1):
        print(i, end=' ')
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

【例 3.14】打印所有水仙花数。所谓的水仙花数是指这样的三位数，其每个数位上的数字的立方和等于它本身，如 153 就是一个水仙花数，因为 $1^3 + 5^3 + 3^3 = 153$。

【程序分析】

要判断一个整数是不是水仙花数，需要从这个整数中拆分出其百位数、十位数和个位数，然后用 if 语句判断由此得到的这些数的立方和是否等于该整数本身。由于水仙花数是三位数，所以使用 for 语句遍历整数序列 range(100, 1000)即可。

【程序代码】

```
print('水仙花数如下：')
for i in range(100, 1000):
    a = i // 100          # 百位上的数
    b = i % 100 // 10     # 十位上的数
    c = i % 10            # 个位上的数
    if a ** 3 + b ** 3 + c ** 3 == i:
        print(i, end='  ')
```

【运行结果】

```
水仙花数如下：
153  370  371  407
```

3.2.3 循环控制语句

循环语句在循环条件成立时会重复执行循环体，一旦循环条件不再满足便会执行 else 子句（如果存在）并结束循环，这是循环语句的正常执行流程。根据实际需要，也可以使用 Python 提供的如下两个循环控制语句来改变循环语句的执行流程。

1．break 语句

break 语句用来终止当前循环的执行，其语法格式如下。

```
break
```

break 语句只能嵌套在 while 语句和 for 语句中。它通常与 if 语句一起使用，可以用来跳出当前所在的循环结构，即使循环条件表达式的值仍然为 True，或者序列中还有未用过的元素，也会立即停止执行循环语句，即跳出循环体，跨过可选的 else 子句（如果有的话），转而执行循环语句的后续语句。

【例 3.15】编写一个菜单程序，用于模拟信息管理系统的运行。

【程序分析】

信息管理系统运行时，显示出系统功能菜单，其中列出所提供的各项功能，并提供一个出口，用户可以从中选择要执行的功能或退出系统。为了使用户能够连续执行多种操作，可以使用一个无限循环来模拟信息管理系统的运行。无限循环使用 while 语句来实现，将循环条件设置为常量 1 或 True，在循环体内放置一个多分支选择语句，当选择退出系统时，执行 break 语句，结束 while 语句，从而退出系统。

【程序代码】

```
import time
menu = '''
*****信息管理系统*****
1.录入信息  2.查询信息
3.打印信息  4.退出系统
'''
while 1:
    print(menu)
    choice = int(input('请选择：'))
    if choice == 1:
        print('您选择了录入功能\n......')
        time.sleep(6)  # 延迟 6 秒钟
        print('录入完毕！')
    elif choice == 2:
        print('您选择了查询功能\n......')
        time.sleep(6)
        print('查询完毕！')
    elif choice == 3:
        print('您选择了打印功能\n......')
        time.sleep(6)
        print('打印完毕！')
    elif choice == 4:
        print('谢谢使用！')
        break  # 退出循环
```

【运行结果】

```
*****信息管理系统*****
1.录入信息  2.查询信息
3.打印信息  4.退出系统

请选择：1
您选择了录入功能
......
录入完毕！

*****信息管理系统*****
1.录入信息  2.查询信息
3.打印信息  4.退出系统
```

```
请选择：2
您选择了查询功能
......
查询完毕!

*****信息管理系统*****
1.录入信息  2.查询信息
3.打印信息  4.退出系统

请选择：4
谢谢使用!
```

2．continue 语句

continue 语句用于跳出本次循环，其语法格式如下。

```
continue
```

与 break 语句一样，continue 语句也只能嵌套在 while 语句和 for 语句中，通常也是与 if 语句一起使用的，但两者的作用有所不同：continue 语句用来跳过当前循环中的剩余语句，然后继续进行下一轮循环；break 语句则用于结束整个循环，即跳出循环体，跨过可选的 else 子句（如果有的话），然后执行循环语句的后续语句。

【例 3.16】已知 x 是一个两位数，且满足公式 $809x = 800x + 9x$，其中 $809x$ 为四位数，$8x$ 为两位数，$9x$ 为 3 位数。求 x 的值。

【程序分析】

由于 x 是两位数，所以可以使用 for 语句遍历整数序列 range(10, 100)，针对该序列中的每个整数进行测试。对于测试的整数而言，如果它与 8 的乘积大于 99，或者与 8 的乘积小于 100，则使用 continue 语句跳过剩余步骤。在剩余步骤中，如果当前整数满足【例 3.16】中的公式，则输出结果并结束循环。

【程序代码】

```
for x in range(10, 100):
    if 8 * x > 99 or 9 * x < 100:
        continue
    if 809 * x == 800 * x + 9 * x:
        print(f'{x = }')
        break
```

【运行结果】

```
x = 12
```

3.2.4　循环结构的嵌套

在一个循环结构中可以嵌入另一个循环结构，由此形成嵌套的循环结构，也称为多重循环结构，如二重循环和三重循环。多重循环结构由外层循环和内层循环组成，当外层循环进入下一轮循环时，内层循环将重新初始化并开始执行。

如果在多重循环结构中使用 break 语句和 continue 语句，则这些语句的作用仅限于其所在层的循环。使用多重循环结构时，嵌套的深度不限，但是需要特别注意代码的缩进问题，内层循环与外层循环之间不能交叉。

【例 3.17】输出乘法口诀表。

【程序分析】

输出乘法口诀表可以通过一个二重 for 语句来实现，外层循环需要执行 9 次，每执行一次输出一行。内层循环执行的次数由行号决定，行号是多少内层循环就执行多少次，每执行一次输出一

个等式；同一个内层循环输出的所有等式位于同一行。

【程序代码】

```
print('乘法口诀表')
for i in range(1, 10):
    for j in range(1, i + 1):
        print(f'{j}×{i}＝{i * j}\t', end='')
    print()
```

【运行结果】

```
乘法口诀表
1×1＝1
1×2＝2  2×2＝4
1×3＝3  2×3＝6  3×3＝9
1×4＝4  2×4＝8  3×4＝12  4×4＝16
1×5＝5  2×5＝10  3×5＝15  4×5＝20  5×5＝25
1×6＝6  2×6＝12  3×6＝18  4×6＝24  5×6＝30  6×6＝36
1×7＝7  2×7＝14  3×7＝21  4×7＝28  5×7＝35  6×7＝42  7×7＝49
1×8＝8  2×8＝16  3×8＝24  4×8＝32  5×8＝40  6×8＝48  7×8＝56  8×8＝64
1×9＝9  2×9＝18  3×9＝27  4×9＝36  5×9＝45  6×9＝54  7×9＝63  8×9＝72  9×9＝81
```

3.3 异常处理

异常是指程序运行期间出现错误或意外情况。在一般情况下，如果 Python 无法正常处理程序就会发生一个异常。引发异常有各种各样的原因，如命名错误、语法错误及数据类型错误等。Python 语言提供了一套完整的异常处理方法，可以用来对各种可预见的错误进行处理。下面首先介绍 Python 提供的标准异常，然后讨论如何捕获和处理异常，最后讲述如何抛出异常。

3.3.1 标准异常

在 Python 中，异常是以对象的形式实现的。BaseException 类是所有异常类的基类，其子类是 Exception。除了 SystemExit、KeyboardInterrupt 和 GeneratorExit 这 3 个系统级异常，所有内置异常类和用户自定义异常类都是 Exception 的子基类。常见的标准异常如表 3.1 所示。

表 3.1 常见的标准异常

异常名称	描　述	异常名称	描　述
BaseException	所有异常的基类	WindowsError	系统调用失败
SystemExit	解释器请求退出	ImportError	导入模块/对象失败
KeyboardInterrupt	用户中断执行（通常是按“Ctrl+C”键）	LookupError	无效数据查询的基类
Exception	常规错误的基类	IndexError	序列中没有此索引
StopIteration	迭代器没有更多的值	KeyError	映射中没有这个键
GeneratorExit	生成器发生异常来通知退出	MemoryError	内存溢出错误
StandardError	所有的内建标准异常的基类	NameError	未声明/初始化对象（没有属性）
ArithmeticError	所有数值计算错误的基类	UnboundLocalError	访问未初始化的本地变量
FloatingPointError	浮点计算错误	ReferenceError	弱引用试图访问已经垃圾回收的对象
OverflowError	数值运算超出最大限制	RuntimeError	一般的运行时错误
ZeroDivisionError	除（或取模）零（所有数据类型）	NotImplementedError	尚未实现的方法
AssertionError	断言语句失败	SyntaxError	Python 语法错误
AttributeError	对象没有这个属性	IndentationError	缩进错误

续表

异常名称	描　　述	异常名称	描　　述
EOFError	没有内建输入，到达 EOF 标记	TabError	制表符和空格混用
EnvironmentError	操作系统错误的基类	SystemError	一般的解释器系统错误
IOError	输入/输出操作失败	TypeError	对类型无效的操作
OSError	操作系统错误	ValueError	传入无效的参数

下面列举一些常见的异常。

在 Python Shell（IDLE）中试图显示一个变量的值时，该变量却没有定义，结果会引发 NameError，具体示例如下。

```
>>> username
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    username
NameError: name 'username' is not defined
```

在进入算术运算时，使用了无效数据类型，具体示例如下。

```
>>> 'book' / 3
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    'book' / 3
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

在进行除法或求余运算中，如果除数为 0，将引发 ZeroDivisionError，具体示例如下。

```
>>> 1 / 0
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    1 / 0
ZeroDivisionError: division by zero
```

当输入 if 语句时，如果在表达式后面未输入冒号便按“Enter”键，将会引发 SystemError，具体示例如下。

```
>>> if x != 0
SyntaxError: invalid syntax
```

在访问列表元素时，如果索引越界，将引发 IndexError，具体示例如下。

```
>>> list1 = [1, 2, 3, 4, 5]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    list1[9]
IndexError: list index out of range
```

在访问字典元素时，如果所使用的关键字不存在，将引发 KeyError，具体示例如下。

```
>>> student = {'name': '张三', 'gender': '男', 'age': 20}
>>> student['email']
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    student['email']
KeyError: 'email'
```

3.3.2　捕获和处理异常

在 Python 中，异常处理可以通过 try-except 语句来实现。这条语句主要由 try 子句和 except 子句两部分组成，可以用来检测 try 语句块中的错误，从而使 except 子句捕获异常信息并加以处理。如果不想在异常发生时结束程序运行，只需要在 try 子句中捕获它即可。按照异常处理分支的数目，try-except 语句可以分为单分支异常处理和多分支异常处理。

1．单分支异常处理

在单分支异常处理中，try-except 语句的语法格式如下。

```
try:
    语句块 0        # 有可能引发异常的操作
except:
    语句块 1        # 发生异常时执行的操作
[else:
    语句块 2]       # 未发生异常时执行的操作
```

其中，try 子句指定一组包含可能会引发异常的语句（语句块 0）；except 子句指定一组发生异常时执行的语句（语句块 1）；可选的 else 子句指定一组未发生异常时执行的语句（语句块 2）。所有语句块可以是单条语句或多条语句。使用单条语句时，该语句可以与 try 子句、except 子句或 else 子句位于同一行。如果使用多条语句，则这些语句必须另起一行，而且具有相同的缩进量。

单分支异常处理语句未指定异常类型，对所有异常不加区分进行统一处理，其执行流程如下：首先执行 try 子句中的语句块，如果未发生异常，则执行 else 子句中的语句块；如果 try 后面的某条语句在执行时出现错误，则停止执行 try 子句中的语句块，而是转向 except 子句中的异常处理语句块。单分支异常处理语句的执行流程如图 3.6 所示。

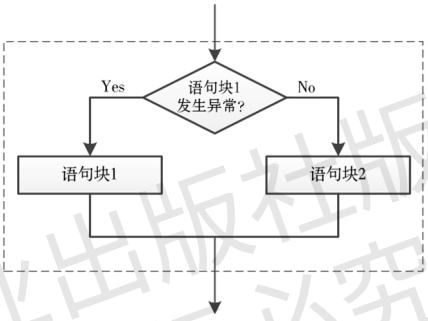

图 3.6　单分支异常处理语句的执行流程

【例 3.18】从键盘上输入两个数字，然后进行除法运算，要求添加异常处理功能。

【程序分析】

本例中程序的功能是做除法运算，即从键盘上输入两个数字，然后进行除法运算。在输入数字和进行除法运算时可能会出现各种错误，编程时可以将输入数字和进行除法运算的代码放在 try 子句中，而将异常处理的代码放在 except 子句中，对各种类型的错误不加区分，在这里进行统一处理。

【程序代码】

```
try:
    x, y = eval(input('请输入两个数字: '))
    z = x / y
    print(f'{x = :.2f}, {y = :.2f}')
    print(f'z = x / y = {z:.2f}')
except:
    print('输入错误! ')
```

【程序说明】

在本例中，使用 input()函数输入字符串内容之后，通过 Python 内置函数 eval()对所输入的内容进行解析，执行所生成的字符串表达式并返回其值。当使用 input()函数输入多项内容时，应以逗号来分隔这些内容，eval()函数会根据所输入的内容确定其数据类型，无须再用类型转换函数 int()或 float()进行数据类型转换。如果输入了无效数据，则会发生异常，此时会提示“输入错误!”。

【运行结果】

```
请输入两个数字：369, 17↵
x = 369.00, y = 17.00
```

```
z = x / y = 21.71
```

再次运行程序，结果如下。

```
请输入两个数字：33, 0↵
输入错误!
```

再次运行程序，结果如下。

```
请输入两个数字：29, 'python'↵
输入错误!
```

2．多分支异常处理

在多分支异常处理中，try-except 语句的语法格式如下。

```
try:
    语句块 0
except 异常类 1 [as 标识 1]:
    语句块 1
except 异常类 2 [as 标识 2]:
    语句块 2
……
except 异常类 n [as 标识 n]:
    语句块 n
except:
    语句块 n+1
[else:
    语句块 n+2]
```

其中，try 子句指定一组可能会引发异常的语句（语句块 0）；带表达式的各个 except 子句分别指定一组发生特定类型的异常时执行的语句（语句块 1～语句块 n）；as 标识为可选项，用于定义异常类实例，以获取异常的描述信息；不带表达式的 except 子句必须位于最后，指定一组发生任何类型异常时执行的语句（语句块 n+1），用于提供默认的异常处理操作；else 子句指定一组未发生异常时执行的语句（语句块 n+2）。各个语句块都可以包含单条语句或多条语句，使用单条语句时，该语句可以与 try 子句、except 子句或 else 子句位于同一行；使用多条语句时，这些语句必须具有相同的缩进量。

如果要在一个 except 子句中捕获多个异常类型，则应使用元组来表示，格式如下。

```
except (异常类 1, 异常类 2) as 标识:
```

多分支异常处理语句可以针对不同的异常类型进行不同的处理，其执行流程如下：首先执行 try 子句中的语句块 0，如果未发生任何异常，则不会执行任何 except 子句，而是执行 else 子句中的语句块 n+2（如果存在的话）。如果 try 子句中的某条语句引发了异常，则停止执行语句块 0 中的剩余语句，而是依次对各个带表达式的 except 子句中的异常类进行检查，试图找到所匹配的异常类型。如果找到了，则执行相应的异常处理语句块；如果未找到，则执行最后一个 except 子句中用于默认异常处理的语句块 n+1。多分支异常处理语句的执行流程如图 3.7 所示。

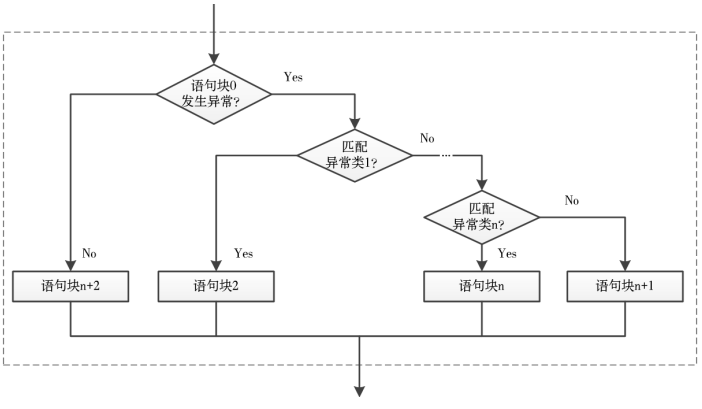

图 3.7　多分支异常处理语句的执行流程

【例 3.19】从键盘上输入两个数字，然后进行除法运算，要求对错误进行分类处理。

【程序分析】

本例中程序的功能仍然是做除法运算，编程时可以将实现输入数字和进行除法运算的代码放在 try 子句中。使用 input()函数动态输入数字时可能会出现各种各样的错误，为了根据不同的错误类型分别进行不同的处理，需要将异常处理的代码放在不同的 except 子句中，并指定不同的异常类型。

【程序代码】

```
try:
    x, y = eval(input('请输入两个数字：'))
    z = x / y
    print(f'{x = :.2f}, {y = :.2f}')
    print(f'z = x / y = {z:.2f}')
except NameError as ne:
    print('输入错误：变量未初始化。')
    print(f'错误描述：{ne}。')
except TypeError as te:
    print('输入错误：数据类型错误。')
    print(f'错误描述：{te}。')
except ZeroDivisionError as zde:
    print('输入错误：使用零作为除数。')
    print(f'错误描述：{zde}')
except:
    print('输入错误！')
else:
    print('除法运算已完成。')
```

【运行结果】

```
请输入两个数字：332, 57↵
x = 332.00, y = 57.00
z = x / y = 5.82
除法运算已完成。
```

再次运行程序，结果如下。

```
请输入两个数字：2020, 0
输入错误：使用零作为除数。
错误描述：division by zero。
```

再次运行程序，结果如下。

```
请输入两个数字：x, y
输入错误：变量未初始化。
错误描述：name 'a' is not defined。
```

再次运行程序，结果如下。

```
请输入两个数字：1000, '555'
输入错误：数据类型错误。
错误描述：unsupported operand type(s) for /: 'int' and 'str'。
```

3．执行清理任务

在 try-except 语句捕获和处理异常之后，如果要执行某种清理任务，可以通过添加一个 finally 子句来实现，该子句用于指定无论是否发生异常都会执行的代码。异常处理语句的完整格式如下。

```
try:
    语句块 0
except:
    语句块 1
else:
    语句块 2
finally:
```

```
    语句块 3
```

无论在 try 子句中是否发生了异常，finally 子句总是在离开 try-except-else-finally 语句之前执行。如果在 try 子句中发生异常并且没有被 except 子句处理，或者在 except 子句或 else 子句中又发生了异常，则在 finally 子句执行之后这个异常将被重新引发。

【例 3.20】除法运算中的异常处理，用于演示 finally 子句的应用。

【程序分析】

本例仍然是做整数除法运算，即从键盘上输入两个数字，然后进行除法运算。与【例 3.19】不同的是，本例使用通用基类 Exception 来处理所有错误，此外还添加了一个 finally 子句。

【程序代码】

```
try:
    x, y = eval(input('请输入两个数字：'))
    z = x / y
    print(f'{x = :.2f}, {y = :.2f}')
    print(f'z = x / y = {z:.2f}')
except Exception as e:
    print('输入错误！')
    print(f'错误描述：{e}')
else:
    print('除法运算正常结束。')
finally:
    print('谢谢使用！')
```

【运行结果】

```
请输入两个数字：818, 19↵
x = 818.00, y = 19.00
z = x / y = 43.05
除法运算正常结束。
谢谢使用！
```

再次运行程序，结果如下。

```
请输入两个数字：999, 5-3-2↵
输入错误！
错误描述：division by zero
谢谢使用！
```

再次运行程序，结果如下。

```
请输入两个数字：'被除数', '除数'↵
输入错误！
错误描述：unsupported operand type(s) for /: 'str' and 'str'
谢谢使用！
```

3.3.3 抛出异常

在 Python 中，程序运行期间出现错误就会引发异常，这种异常是由 Python 解释器自动引发的。在程序设计过程中，有时候需要主动抛出异常，这可以使用 raise 语句和 assert 语句来实现。

1．raise 语句

raise 语句用于显式地引发异常，该语句有以下几种用法。

- 使用不带参数的 raise 语句重新引发刚刚发生的异常，语法格式如下。

```
raise
```

请看下面的例子。

```
>>> try:
    x = 1 / 0
except:
    print('出错啦！')
```

```
    raise
出错啦!
Traceback (most recent call last):
  File "<pyshell#5>", line 2, in <module>
    x = 1 / 0
ZeroDivisionError: division by zero
```

在上述例子中，try 子句进行除法运算时引发了 ZeroDivisionError，程序跳转到 except 子句中执行打印语句，然后使用 raise 语句再次引发了刚刚发生的异常，导致程序出现错误而终止运行。

使用不带表达式的 raise 语句时，如果当前范围内没有任何异常处于活动状态，则会引发 RuntimeError，表明这是一个错误。

- 在 raise 语句中使用异常类名称创建该类的实例对象并引发异常，语法格式如下。

```
raise 异常类([描述信息])
```

其中，描述信息参数是可选的，用于对异常类指定描述信息。

请看下面的例子。

```
>>> raise Exception('主动抛出异常')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    raise Exception('主动抛出异常')
Exception: 主动抛出异常
```

- 使用 raise-from 语句在一个异常中抛出另一个异常，语法格式如下。

```
raise 异常类或实例 from 异常类或实例
```

请看下面的例子。

```
>>> try:
    x = 1 / 0
except Exception as ex:
    raise RuntimeError("出错啦。") from ex
```

在上述例子中,try 子句中除以零引发 ZeroDivisionError,程序跳转到 except 子句中执行。except 子句能够捕获所有异常，并使用 raise-from 语句抛出 ZeroDivisionError 后再抛出 RuntimeError，运行结果如下。

```
Traceback (most recent call last):
  File "<pyshell#16>", line 2, in <module>
    x = 1 / 0
ZeroDivisionError: division by zero
The above exception was the direct cause of the following exception:
Traceback (most recent call last):
  File "<pyshell#16>", line 4, in <module>
    raise RuntimeError("出错啦。") from ex
RuntimeError: 出错啦。
```

2．assert 语句

assert 语句用于声明断言，即期望用户满足指定的约束条件，语法格式如下。

```
assert 逻辑表达式，字符串表达式
```

其中，逻辑表达式指定一个约束条件，如果该表达式的值为 False，则会抛出 AssertionError，否则什么事情也不做；字符串表达式指定 AssertionError 的描述信息。

从逻辑上分析，assert 语句与下面的语句等效。

```
if 逻辑表达式:
    raise AssertionError(字符串表达式)
```

显然，assert 语句可以视为条件式的 raise 语句，其主要作用是帮助调试程序，以保证程序正常运行。请看下面的例子。

```
>>> x = 3
>>> assert x == 1, 'x 的值必须等于 1。'
```

```
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    assert x == 1, 'x 的值必须等于 1。'
AssertionError: x 的值必须等于 1。
```

【例 3.21】从键盘输入三角形的三条边长，计算三角形的面积。

【程序分析】

已知三角形的三条边长为 a、b、c，则三角形面积计算公式为 $S=\sqrt{p(p-a)(p-b)(p-c)}$，其中 $p=(a+b+c)/2$。使用 assert 语句设置构成三角形的条件，即任何两条边之和大于第三条边。如果不满足该条件，则抛出 AssertionError。

【程序代码】

```
try:
    a, b, c = eval(input('请输入三角形的三条边长：'))
    assert a + b > c and b + c > a and c + a > b, '无效输入，不能构成三角形！'
    p = (a + b + c) / 2
    area = (p * (p - a) * (p - b) * (p - c))**0.5
    print(f'三角形面积为{area:.2f}')
except Exception as ex:
    print(f'输入有误：{ex}')
```

【运行结果】

```
请输入三角形的三条边长：3, 4, 5↵
三角形面积为 6.00
```

再次运行程序，结果如下。

```
请输入三角形的三条边长：1, 2, 3↵
无效输入，不能构成三角形!
```

3.4 典型案例

作为本章知识的综合应用，下面给出两个典型案例，其中一个根据输入的出生日期计算生肖和星座，另一个则用于编写计算机猜数游戏。

3.4.1 计算生肖和星座

【例 3.22】从键盘输入出生日期，据此计算生肖和星座。

【程序分析】

生肖和星座都可以根据出生日期来计算，为此首先要调用 Python 内置函数 input()来输入出生日期，然后通过调用 time.strptime()函数将日期字符串解析为时间元组，并从元组中分别取出年、月、日的数值。

十二生肖包括鼠、牛、虎、兔、龙、蛇、马、羊、猴、鸡、狗、猪，可以根据年份除以 12 所得的余数来判断：0（猴）；1（鸡）；2（狗）；3（猪）；4（鼠）；5（牛）；6（虎）；7（兔）；8（龙）；9（蛇）；10（马）；11（羊），可以使用多分支 if-elif-else 语句测试该余数来实现。

十二星座包括水瓶座、双鱼座、白羊座、金牛座、双子座、巨蟹座、狮子座、处女座、天秤座、天蝎座、射手座、摩羯座，可以根据出生月份和日子来判断：1 月 21 日—2 月 19 日（水瓶座）；2 月 20 日—3 月 20 日（双鱼座）；3 月 21 日—4 月 20 日（白羊座）；4 月 21 日—5 月 21 日（金牛座）；5 月 22 日—6 月 21 日（双子座）；6 月 22 日—7 月 22 日（巨蟹座）；7 月 23 日—8 月 23 日（狮子座）；8 月 24 日—9 月 23 日（处女座）；9 月 24 日—10 月 23 日（天秤座）；10 月 24 日—11 月 22 日（天蝎座）；11 月 23 日—12 月 21 日（射手座）；12 月 22 日—1 月 20 日（摩羯座）。为了计算星座，可将月份值扩大 100 倍加上日子数构成一个整数，并使用多分支 if-elif-else 语句对该整数进行测试。例如，对于 1 月 21 日—2 月 19 日，条件表达式应表示为“121 <= xz <= 219”；对于

12 月 22 日—1 月 20 日，条件表达式则应使用 or 运算符来组合两个条件，即表示为“1222 <= xz or xz <= 120”。

【程序代码】

```
import time          # 导入 time 模块

zodiac_sign = ''     # 生肖
constellation = ''   # 星座
print('***计算生肖和星座***')
birthdate = input('请输入出生日期：')
date = time.strptime(birthdate, '%Y-%m-%d')
year = date.tm_year
month = date.tm_mon
day = date.tm_mday
# 计算生肖
remainder = year % 12
if remainder == 0:
    zodiac_sign = '猴'
elif remainder == 1:
    zodiac_sign = '鸡'
elif remainder == 2:
    zodiac_sign = '狗'
elif remainder == 3:
    zodiac_sign = '猪'
elif remainder == 4:
    zodiac_sign = '鼠'
elif remainder == 5:
    zodiac_sign = '牛'
elif remainder == 6:
    zodiac_sign = '虎'
elif remainder == 7:
    zodiac_sign = '兔'
elif remainder == 8:
    zodiac_sign = '龙'
elif remainder == 9:
    zodiac_sign = '蛇'
elif remainder == 10:
    zodiac_sign = '马'
elif remainder == 11:
    zodiac_sign = '羊'
# 计算星座
xz = month * 100 + day
if 121 <= xz <= 219:
    constellation = '水瓶座'
elif 220 <= xz <= 320:
    constellation = '双鱼座'
elif 321 <= xz <= 420:
    constellation = '白羊座'
elif 421 <= xz <= 521:
    constellation = '金牛座'
elif 522 <= xz <= 621:
    constellation = '双子座'
elif 622 <= xz <= 722:
    constellation = '巨蟹座'
elif 723 <= xz <= 823:
    constellation = '狮子座'
elif 824 <= xz <= 923:
```

```
    constellation = '处女座'
elif 924 <= xz <= 1023:
    constellation = '天秤座'
elif 1024 <= xz <= 1122:
    constellation = '天蝎座'
elif 1123 <= xz <= 1221:
    constellation = '射手座'
elif xz <= 120 or xz >= 1222:
    constellation = '摩羯座'
print('计算结果如下：')
print(f'生肖：{zodiac_sign}；星座：{constellation}')
```

【运行结果】

```
***计算生肖和星座***
请输入出生日期：1999-9-9↵
计算结果如下：
生肖：兔；星座：处女座
```

3.4.2 猜数游戏

【例 3.23】编写一个猜数游戏，生成一个 1～100 的随机整数作为秘密数字，允许有 6 次尝试机会，即通过键盘输入猜测的结果，并提示猜测结果是高还是低，最后输出游戏结果。

【程序分析】

编写这个猜数游戏时，首先需要导入 random 模块，并通过调用 random.randint(1, 100)函数生成一个随机数。6 次猜数尝试通过 while 语句来实现，循环条件是猜测的数字与秘密数字不相等并且猜测次数小于或等于 6。为这个 while 语句添加 else 子句，当循环结束后执行 else 子句，输出游戏结果。

【程序代码】

```
import random                          # 导入 random 模块

secret = random.randint(1, 100)  # 生成 1～100 的随机数
guess = 0                              # 初始化猜测的数字
tries = 0                              # 初始化尝试的次数
print('有一个从 1 到 100 的秘密整数。')
print('这个整数到底是什么呢？你一共有 6 次机会。')

while guess != secret and tries <= 6:
    guess = int(input('猜一猜：'))
    if guess < secret:
        print('不对，太小了！')
    elif guess > secret:
        print('糟糕，太大了！')
    tries = tries + 1
else:
    if guess == secret:
        print('恭喜你猜对了！')
    else:
        print('很遗憾，你没能猜出来。')
        print(f'告诉你吧，这个秘密数字是{secret}。')
        print('祝你下次好运！')
```

【运行结果】

```
有一个从 1 到 100 的秘密整数。
这个整数到底是什么呢？你一共有 6 次机会。
猜一猜：50↵
不对，太小了！
```

```
猜一猜：80↵
糟糕，太大了！
猜一猜：65
恭喜你猜对了！
```

再次运行程序，结果如下。

```
有一个从 1 到 100 的秘密整数。
这个整数到底是什么呢？你一共有 6 次机会。
猜一猜：33↵
糟糕，太大了！
猜一猜：26↵
糟糕，太大了！
猜一猜：19↵
糟糕，太大了！
猜一猜：9↵
不对，太小了！
猜一猜：10↵
不对，太小了！
猜一猜：12↵
不对，太小了！
很遗憾，你没能猜出来。
告诉你吧，这个秘密数字是 16。
祝你下次好运！
```


习 题 3

一、选择题

1. 在下列各项中，（　　）不属于流程控制结构。

   A. 顺序结构　　B. 网状结构

   C. 循环结构　　D. 选择结构

2. 在下列各项中，（　　）用于实现多分支选择。

   A. 在 if-else 的 if 中加 if　　B. 在 if-else 的 else 中加 if

   C. if-elif-else　　D. if-else

3. 在下列各项中，（　　）可以用来判断整数 n 是否为整数。

   A. n % 2 == 0　　B. n % 2 != 1

   C. n // 2 == 0　　D. n // 2 != 0

4. 下列关于 break 语句和 continue 语句的叙述中，不正确的是（　　）

   A. 在多重循环语句中，break 语句的作用仅限于其所在层的循环

   B. continue 语句执行后，继续执行循环语句的后续语句

   C. continue 语句与 break 语句类似，只能用在循环语句中

   D. break 语句结束循环，继续执行循环语句的后续语句

5. 下列语句执行后，变量 n 的值为（　　）。

```
n = 0
for i in range(1, 100, 3):
    n += 1
```

   A. 32　　B. 33

   C. 34　　D. 35

6. 在下列语句中，正确的是（　　）。

   A. max = x > y ? x: y　　B. min = x if x < y else y

C. if (x > y) print(x)　　D. while (x < 10) print(x)

7. 以 0 作为除数时将会引发（　　）。

A. ZeroDivisionError　　B. AttributeError

C. IndexError　　D. NameError

二、判断题

1. 若 x=30，y=90，则条件表达式 x if x>y else y 的值为 30。（　　）
2. while 语句至少会执行一次。（　　）
3. for 语句用于遍历任何有序序列对象中的所有元素。（　　）
4. 循环语句可以嵌套使用。（　　）
5. break 语句和 continue 语句均可用于循环语句中，两者的作用完全相同。（　　）
6. 执行 try-except 语句时，首先执行 try 子句中的语句块，如果未发生异常，则不检查任何 except 子句，而转向执行 else 子句中的语句块。（　　）
7. assert 语句可以看作条件式的 raise 语句。（　　）

三、编程题

1. 从键盘输入一个年份，判断这一年是不是闰年。
2. 从键盘输入 3 条线段的长度，判断能否构成三角形，若能则计算三角形的面积。
3. 求解爱因斯坦的阶梯问题：有一个长阶梯，若每步上 2 阶，最后剩 1 阶；若每步上 3 阶，最后剩 2 阶；若每步上 5 阶，最后剩 4 阶；若每步上 6 阶，最后剩 5 阶；只有每步上 7 阶，最后刚好一阶也不剩。计算该阶梯至少有多少阶。
4. 利用下列公式计算圆周率π的值。

$$\frac{\pi}{4}=\left(\frac{1}{2}+\frac{1}{3}\right)-\frac{1}{3}\left(\frac{1}{2^3}+\frac{1}{3^3}\right)+\frac{1}{5}\left(\frac{1}{2^5}+\frac{1}{3^5}\right)-\cdots$$

5. 计算棋盘上的麦粒数：国际象棋棋盘由 64 个黑白相间的方格组成，假如在第 1 个方格放 1 颗麦粒，在第 2 个方格放 2 颗麦粒，在第 3 个方格放 4 颗麦粒，以后每个方格放的麦粒数都比前一个方格增加一倍，请问在第 64 个方格放的麦粒数是多少？这样摆满棋盘上的 64 个方格一共需要多少颗麦粒?
6. 从键盘输入 a、b、c 的值，在实数范围内求解一元二次方程 $ax^2+bx+c=0$。要求用 try-except 语句来处理二次项 a 的值不是数字或等于 0 等异常情况。