# 第 5 章

# 程序流控制与异常处理

本章主要介绍程序流的逻辑控制，以及程序流发生异常的处理方法。

本章的学习目标：

- 掌握 if、while、for 程序流控制流程；
- 掌握异常的捕获 try…except…finally 的使用。

## 5.1 Python 程序控制流

Python 源程序代码是根据一定的逻辑条件运行的程序流，程序流包括顺序语句、条件语句和循环语句。顺序语句按照源程序代码的先后顺序，从上到下依次运行每一条程序语句。顺序结构是程序的最基本结构(见图 5.1)。

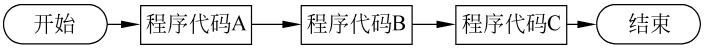


图 5.1 顺序结构

条件语句(if)和循环语句(while、for)是根据一定的逻辑条件(True 或者 False)来选择后续运行的代码块。

**提示**：Python 无 switch 或 case 语句。因为按照 Python 之“禅(Zen)”宗旨，该语句属于多余。

## 5.2 if 条件语句

Python 语言中 if 条件语句是最基本的逻辑判断程序流控制语句。语句的关键词为 if…elif…else，关键字 elif 是 else if 的缩写。其格式为：

```
if  C1:
    程序代码 1
elif C2:
    程序代码 2
else:
    程序代码 3
```

if 条件语句的语法说明：

(1) if 条件后面用冒号“:”表示满足条件后要运行的语句块。

(2) if 条件语句可以有 0～n(任意)个 elif,而不用 else if。

(3) if 条件语句结束可以有 0～1 个 else 语句。

(4) if 语句可以嵌套,使用缩进空格数来区分嵌套语句块。

if 条件语句的程序流程图如图 5.2 所示。

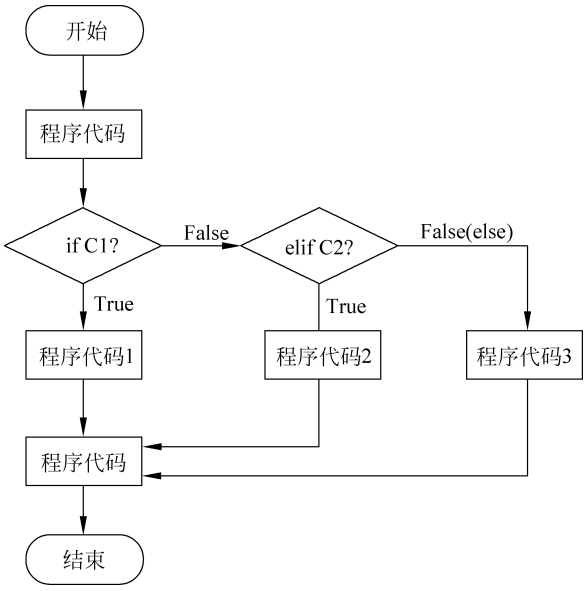


图 5.2　if 条件语句的程序流程图

图 5.2 中的 if 条件语句程序运行流程如下：

(1) 如果 C1 为 True,将运行“程序代码 1”块语句。

(2) 如果 C1 为 False,将判断 C2。

(3) 如果 C2 为 True,将运行“程序代码 2”块语句。

(4) 如果 C2 为 False(else),将运行“程序代码 3”块语句。

**【例 5.1】** 输入 0～100 任意成绩数据,通过 if 条件语句判断成绩的等级。

```
num = int(input('请输入分数: '))
if  0 <= num < 60:      # 判断值是否在 0～60
    print ('挂科!')
elif 60 <= num < 70:    # 判断值是否在 60～70
```

```
    print ('及格')
elif 70 <= num < 80:                    # 判断值是否在 70～80
    print ('中')
elif 80 <= num < 90:                    # 判断值是否在 80～90
    print ('良好')
elif 80 <= num <= 100:                  # 判断值是否在 90～100
    print ('优秀')
else:
    print ('输入有误!')
```

**提示**：程序如果只需要条件赋值，可以用 if 的简写形式。

如果字符串 s 的长度大于 4，把 s 赋值给 b，否则输出'too short'。

```
s = input('请输入一个长度大于 4 的字符串: ')
b = s if len(s)> 4 else 'too short'
```

如果 a 等于 10，输出'等于 10'，否则输出'不等于 10'。

```
a = 23
print('等于 10') if a == 10 else print('不等于 10')
```

## 5.3　Python 循环语句

Python 语言中有两种类型的循环语句，即 while 循环语句和 for 循环语句。

**提示**：Python 无 do…while 循环语句。因为按照 Python 之“禅(Zen)”，它属于多余的。

### 5.3.1　while 循环语句

运行 while 循环语句，程序先判断循环条件，如果条件满足，则进入再循环。

while 循环语句的语法说明：

(1) while 循环语句后面用冒号“:”表示满足条件后要运行的语句块。

(2) while 循环语句可以嵌套，使用缩进空格数来区分嵌套语句块。

(3) while 循环语句可以有 else(很少用)语句。

**【例 5.2】** while 循环语句示例。

```
# 用 while 循环语句,必须有一个控制逻辑条件
count = 0
while count < 3:
    # 循环就是重复运行循环体里面的代码
    print(f'Round.{count} test!')
    count += 1
```

运行结果：

```
Round.0 test!
Round.1 test!
Round.2 test!
```

### 5.3.2 for…in 循环语句

Python 中的 for…in 语句是另一种循环语句，该语句必须有一个可迭代(iterates)的对象才能循环。它会遍历序列(可迭代对象)中的每一个元素。

**提示**：Python 的 for 语句是遍历任何可迭代的对象，如 list、set 等，而不是仅用来控制循环次数或循环条件，这与 C 和 Java 语言中的 for 语句有本质不同。

```
a = ['hello', 'world']
for i in range(len(a)):
    print(i, a[i])
```

运行结果：

```
0 hello
1 world
```

while 和 for 循环语句可以包含 else 子句(虽然使用概率很低)，即结束 while 或 for 循环后运行的语句。

**【例 5.3】** for 循环代码示例。

```
'''
for 迭代对象 in 序列:
    代码块(一行语句或多行代码)
else:
    代码块(一行语句或多行代码)
'''
while 条件:
else:
    代码块(一行语句或多行代码)
'''
```

示例代码：

```
sites = ["Youtub", "Google","Baidu","Taobao"]
for site in sites:
    if site == "sohu":
       break
    print("get " + site)
```

```
else:
    print("没有循环数据!")
print("完成循环!")
```

### 5.3.3　break、continue、pass 语句

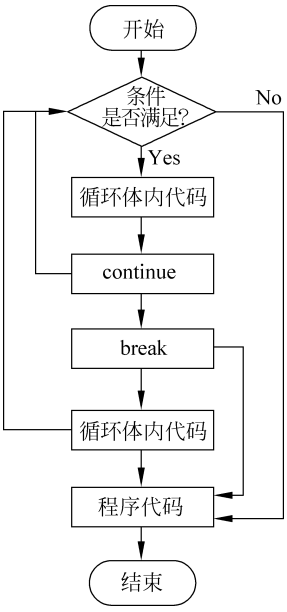


图 5.3　循环与 continue、break 控制语句

break 语句与 Java 和 C 语言的 break 控制语句一样。即程序运行到 break 语句，中断循环、跳出 break 所在的最里层的循环体。

continue 语句与 Java 和 C 语言的 continue 控制语句一样。即程序运行到 continue，不再向下运行，直接再进入下一次循环。

关于 continue 和 break 语句的区别(见图 5.3)，可以举一个通俗的例子场景来表达。

- 使用 continue 语句的场景：甲乙两人计划准备下 5 局象棋，两人在下到第 3 局中途时，乙看到本局败势已定，但乙并不服输，于是放弃第 3 局，而接着比赛下面第 4、5 局，这时使用 continue 语句。
- 使用 break 语句的场景：甲乙两人计划下 5 局象棋，两人在下到第 3 局中途，乙看到败局已定，丧失了信心，直接放弃所有比赛，退出比赛，这时使用 break 语句。

**【例 5.4】** continue 语句代码示例。

```
# continue 语句,跳过循环
a = '2123456'
for letter in a:
    if letter == '2':
        continue
    print(letter)
#运行结果: 1 3 4 5 6
```

break 语句用来终止循环语句，即循环条件没有 False 条件或者序列还没被完全递归完，也会停止运行循环语句。

**【例 5.5】** break 语句代码示例。

```
# break 语句,跳出循环
b = '1234567'
for sam in b:
    if sam == '2':
        break
    print(sam)
#运行结果: 1
```

pass 语句是空语句,不运行任何操作。通常在开发程序的原型或调试程序时,用它来占位,将来用代码替换掉 pass 语句。如:

```
while 条件:
    pass
```

## 5.4 异常

异常是一种事件,它有可能在程序运行过程中发生,会改变程序的正常运行。Python 程序运行时,可能涉及输入输出、数学运算,以及对文件和网络资源的访问等。有时程序运行时,会遇到算术运算除数为 0、无权访问文件、网络连接中断等问题,程序会引发异常。必须在程序中捕获、处理这些异常,才能确保程序正确运行。

**【例 5.6】** 异常代码示例。

```
num = input('please input the number?')
for i in range(int(num)):
    print(i)
```

上述程序在运行时,如果用户输入的是数字字符串,则可以得到正确结果,如果输入的是非数字,则程序会引发异常。

```
please input the number?ok
ValueError: invalid literal for int() with base 10: 'ok'
```

### 5.4.1 异常的处理

异常处理是通过使用 try…except…finally 语句来捕获、处理异常实现的。其语法为:

```
try:
    <语句>          # 运行的代码
except <名字>:
    <语句>          # 如果在 try 代码块中,引发了'名字'异常
except <名字> as <数据>:
    <语句>          # 如果引发了'名字'异常,获得附加的数据
finally:
    <语句>          # 有没有异常,始终运行
```

#### 1. 程序流程

(1) 运行 try 语句后,Python 也在当前程序的上下文中做标记,这样当异常出现时就可以回到这里。如果运行 try 语句没有发生异常,则程序直接跳转到 finally 语句位置。

（2）如果运行 try 后的语句发生异常，程序流就跳回到 try 语句，并运行第一个匹配该异常的 except 子句；若没有该异常匹配的 except 子句，该异常将被递交到上层的 try 语句，或者到程序的最上层（这样将结束程序，并打印默认的出错信息）。

（3）异常处理完毕，控制流就通过整个 try 语句（除非在处理异常时又引发新的异常）。finally：无论系统有无异常代码均运行。

**2. 异常说明**

（1）except 语句可以有多个，Python 会按 except 语句的顺序依次匹配出现的异常，如果异常已经处理就不会再进入后面的 except 语句。

（2）except 语句后面如果不指定异常类型，则默认捕获所有异常。可以通过 logging 或者 sys 模块获取当前异常。

（3）except 语句可选，finally 语句也可选，但是二者必须要有一个，否则 try 语句就没有意义了。

（4）except 语句可以以元组形式同时指定多个异常。

**【例 5.7】** 异常捕获与处理代码示例。

```
while True:
    try:
        num = input('please input the number?')
        for i in range(int(num)):
            print(i)
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again  ")
    finally:
        print('Whatever, I always run!')
```

运行结果：

```
please input the number?ss
Oops!  That was no valid number.  Try again
Whatever, I always run!

please input the number?3
0
1
2
Whatever, I always run!
```

异常的参数：

```
while True:
    try:
        num = input('please input the number?')
```

```
            for i in range(int(num)):
                print(i)
            break
        except ValueError as arg :
             print(arg.args)
        finally:
            print('Whatever, I always run!')
```

一个异常可以带上参数,可作为输出的异常信息参数。

```
please input the number?rr
("invalid literal for int() with base 10: 'rr'",)
Whatever, I always run!
please input the number?2
0
1
Whatever, I always run!
```

变量接收的异常值通常包含在异常的语句中,保持在一个元组中(如上述例子错误("invalid literal for int() with base 10: 'rr'",)参数是一个元组),变量可以接收一个或者多个值。元组通常包含错误字符串、错误数字和错误位置。

### 5.4.2 异常的抛出

Python 程序除了可以捕获异常外,还可以在代码中使用 raise 语句主动抛出异常。raise 语句的一般语法如下:

```
raise [Exception [, args [, traceback]]]
```

其中:

- Exception 是异常的类型(例如,NameError),args 是异常参数的值。
- 参数是可选的;如果没有提供,则异常参数为 None。
- 最后一个参数 traceback 也是可选的(在实践中很少使用),如果存在,则用于异常的追溯对象。

示例:

```
x = input('input x:')
if (int(x)< 5):
    raise  NameError('HiThere')
```

运行结果:

```
    raise  NameError('HiThere')
NameError: HiThere
```

raise 唯一的一个参数指定了要被抛出的异常。它必须是一个异常的实例或者是异

常的类(也就是 Exception 的子类)。

【例 5.8】 raise 主动抛出异常代码示例。

```
try:
    x = input('input x:')
    if (int(x)< 5):
        raise NameError('Hi There')
    print(x)
except NameError as arg:
    print(arg)
```

运行结果:

```
input x:4
HiThere
```

## 5.5 断言的用法

在开发一个程序时,与其让它运行时崩溃,不如在它出现错误条件时就崩溃(返回错误)。这时断言(assert)就显得非常有用。

assert 的语法格式:

```
assert expression
```

它的等价语句为:

```
if not expression:
    raise AssertionError
```

【例 5.9】 assert 代码示例。

```
>>> assert 1 == 1
>>> assert 1 == 0
Traceback (most recent call last):
  File "< pyshell # 1 >", line 1, in < module >
    assert 1 == 0
AssertionError
```

## 5.6 with 语句

with 语句上下文管理器是一个对象,它定义了在运行 with 语句时要建立的运行时上下文(资源),它处理进入和退出所需运行时上下文,并运行代码块。它是对普通的 try…except…finally 使用模式进行封装以方便地重用。

with 语句的语法为:

```
with expression [as target]
```

参数说明：

- expression：一个需要运行的表达式；
- target：一个变量或者元组，存储的是 expression 表达式运行返回的结果，它是可选参数。

相比 try…except…finally 语句，with 语句更加简洁。

**【例 5.10】** 使用 with 语句打开文本文件 c:/a.txt，并显示文件内容。

```
with open('c:/a.txt',encoding = 'utf - 8') as f :
    print(f.read())
```

## 5.7 综合案例

Python 中有些对象是可变的，有些对象是不可变的。可以用函数 hash(object)来判定对象是否可变，如果 hash(object)有返回值的是不可变。hash(object)抛出异常的是可变对象。以下针对 list、tuple、range、str、bytes、bytearry、memoryview、set、frozenset、dict 十类对象进行可变与不可变对象测试。

**【例 5.11】** Python 中可变对象与不可变对象代码测试(通过代码自动判读对象是否可以改变)。

测试对象为：

```
a = [1,2,'hello']
b = set('hello')
c = dict(one = 11, two = 22, three = 33)
d = bytearray('hello',encoding = 'utf - 8')
e = (1,2,'hello')
f = range(1,6,2)
g = 'hello'
h = bytes('hello',encoding = 'utf - 8')
i = memoryview(b'hello')
j = frozenset('hello')
```

测试代码为：

```
test = [a,b,c,d,e,f,g,h,i,j]
for i in test:
    try:
        hash(i)
        print(f'{type(i)} 是不可变对象')
    except:
        print(f'{type(i)} 是可变对象')
```

图 5.4 所示对象种类与下述测试结果相符。

- <class 'list'>的实例是可变对象。
- <class 'set'>的实例是可变对象。
- <class 'dict'>的实例是可变对象。
- <class 'bytearray'>的实例是可变对象。
- <class 'tuple'>的实例是不可变对象。
- <class 'range'>的实例是不可变对象。
- <class 'str'>的实例是不可变对象。
- <class 'bytes'>的实例是不可变对象。
- <class 'memoryview'>的实例是不可变对象。
- <class 'frozenset'>的实例是不可变对象。

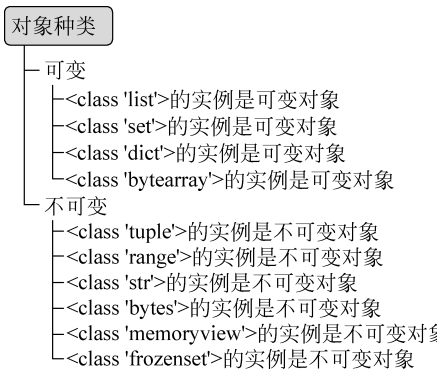


图 5.4　对象种类

## 5.8　本章小结

通过本章的学习，可以掌握 if、while、for 语句对程序流的控制，掌握程序发生异常的处理方法。

## 5.9　习题

扫码观看

(1) 已知上海地铁 1、2 号线(为了简化问题，只考虑这两条线路)两条地铁站线路各站点名称(可以到 http://service.shmetro.com/axlcz01/index.htm 获取)，通过 Python 编程获得“上海南站”到“金科路”的换乘路线。

(2) 模拟双色球投注与兑奖。使用循环随机投注 10 000 注，给定一组中奖号码，判断有多少注号码中奖，计算中奖概率。

(3) 给定一个字典，请按照逆序输出字典的 key 和 value。