第3章　选择与循环

在传统的面向过程程序设计中有3种经典的控制结构，即顺序结构、选择结构和循环结构。即使是在面向对象程序设计语言中以及事件驱动或消息驱动应用开发中，也无法脱离这3种基本的程序结构。可以说，不管使用哪种程序设计语言，在实际开发中，为了实现特定的业务逻辑或算法，都不可避免地要用到大量的选择结构和循环结构，并且经常需要将选择结构和循环结构嵌套使用。本章首先介绍条件表达式和Python中选择结构与循环结构的语法，然后通过几个示例来理解其用法。

3.1　条件表达式

在选择结构和循环结构中，都要使用条件表达式来确定下一步的执行流程。在Python中，单个常量、变量或者任意合法表达式都可以作为条件表达式。

在选择和循环结构中，条件表达式的值只要不是False、0(或0.0、0j等)、空值None、空列表、空元组、空集合、空字典、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价。

关于表达式和运算符的详细内容在1.4.5节中已有介绍，此处不再赘述，只简单介绍一下条件表达式中比较特殊的几个运算符。首先是关系运算符，与很多语言不同的是，在Python中的关系运算符可以连续使用，例如：

```
>>> print(1<2<3)
True
>>> print(1<2>3)
False
>>> print(1<3>2)
True
```

比较特殊的运算符还有逻辑运算符and和or，这两个运算符具有短路求值或惰性求值的特点，简单地说，就是只计算必须计算的表达式的值。在设计条件表达式时，在表示复杂条件时如果能够巧妙利用逻辑运算符and和or的短路求值或惰性求值特性，可以大幅度提高程序的运行效率，减少不必要的计算与判断。

以and为例，对于表达式“表达式1 and 表达式2”而言，如果“表达式1”的值为False或其他等价值时，不论“表达式2”的值是什么，整个表达式的值都是False，此时“表达式2”的值无论是什么都不影响整个表达式的值，因此将不会被计算，从而减少不必要的计算和判断。逻辑或运算符or也具有类似的特点，读者可以自行分析。

在设计条件表达式时，如果能够大概预测不同条件失败的概率，并将多个条件根据and和or运算的短路求值特性组织先后顺序，可以大幅度提高程序运行效率。

在Python中，条件表达式中不允许使用赋值运算符“=”，避免了某些语言中误将关系

运算符“==”写作赋值运算符“=”带来的麻烦,例如下面的代码,在条件表达式中使用赋值运算符“=”将抛出异常,提示语法错误。

```
>>> if a = 3:
SyntaxError: invalid syntax
```

3.2 选 择 结 构

选择结构通过判断某些特定条件是否满足来决定下一步的执行流程,是非常重要的控制结构。常见的有单分支选择结构、双分支选择结构、多分支选择结构、嵌套的分支结构,形式比较灵活多变,具体使用哪一种最终还是取决于所要实现的业务逻辑。后面章节中讲到的循环结构和异常处理结构中也可以带有 else 子句,可以看作是选择结构的一种变形。

3.2.1 单分支选择结构

单分支选择结构是最简单的一种形式,其语法如下所示,其中表达式后面的冒号“:”是不可缺少的,表示一个语句块的开始,后面几种其他形式的选择结构和循环结构中的冒号也是必须有的。

```
if 表达式:
    语句块
```

当表达式值为 True 或其他等价值时,表示条件满足,语句块将被执行,否则该语句块将不被执行。

```
x = input('Input two numbers:')                    #2 个数字之间使用空格分隔
a, b = map(int, x.split())
if a > b:
     a, b = b, a                                   #交换两个变量的值
print(a, b)
```

3.2.2 双分支选择结构

双分支选择结构的语法为

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

当表达式值为 True 或其他等价值时,执行语句块 1,否则执行语句块 2。下面的代码演示了双分支选择结构的用法:

```
>>> chTest = ['1', '2', '3', '4', '5']
>>> if chTest:                                     #前面的 3 个大于号可以理解为不占位置
    print(chTest)
else:                                              #else 虽然顶格,但逻辑上和 if 是对齐的
```

```
    print('Empty')
['1', '2', '3', '4', '5']
```

Python 还支持如下形式的表达式：

```
value1 if condition else value2
```

当条件表达式 condition 的值与 True 等价时，表达式的值为 value1，否则表达式的值为 value2。另外，在 value1 和 value2 中还可以使用复杂表达式，包括函数调用。下面的代码演示了上面的表达式的用法，从代码中可以看出，这个结构的表达式也具有惰性求值的特点。

```
>>> x = math.sqrt(9) if 5 > 3 else random.randint(1, 100) #此时还没有导入 math 模块
NameError: name 'math' is not defined
>>> import math
#此时还没有导入 random 模块，但由于条件表达式 5 > 3 的值为 True，所以可以正常运行
>>> x = math.sqrt(9) if 5 > 3 else random.randint(1,100)
#此时还没有导入 random 模块，由于条件表达式 2 > 3 的值为 False，需要计算第二个表达式的值，因此出错
>>> x = math.sqrt(9) if 2 > 3 else random.randint(1, 100)
NameError: name 'random' is not defined
>>> import random
>>> x = math.sqrt(9) if 2 > 3 else random.randint(1, 100)
```

3.2.3 嵌套的选择结构

嵌套的选择结构为用户提供了更多的选择，可以实现复杂的业务逻辑，一种语法形式为

```
if 表达式 1:
    语句块 1
elif 表达式 2:
    语句块 2
elif 表达式 3:
    语句块 3
    ⋮
else:
    语句块 n
```

其中，关键字 elif 是 else if 的缩写。下面的代码演示了利用该语法将成绩从百分制变换到等级制的实现方法。

```
def func(score):
    if score > 100:
        return 'wrong score.must<= 100.'
    elif score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
```

```
    elif score >=70:
        return 'C'
    elif score >=60:
        return 'D'
    elif score >=0:
        return 'E'
    else:
        return 'wrong score.must> 0.'
```

另一种嵌套选择结构的语法形式如下:

```
if 表达式 1:
    语句块 1
    if 表达式 2:
        语句块 2
    else:
        语句块 3
else:
    if 表达式 4:
        语句块 4
```

使用该结构时,一定要严格控制好不同级别代码块的缩进量,因为这决定了不同代码块的从属关系以及业务逻辑是否被正确地实现、是否能够被 Python 正确理解和执行。例如,百分制转等级制的示例,作为一种编程技巧,还可以尝试下面的写法:

```
def func(score):
    degree = 'DCBAAE'
    if score > 100 or score < 0:
        return 'wrong score.must between 0 and 100.'
    else:
        index = (score- 60)//10
        if index >=0:
            return degree[index]
        else:
            return degree[-1]
```

3.2.4 选择结构应用案例

例 3-1 面试资格确认。

```
age = 24
subject = "计算机"
college = "非重点"
if (age>25 and subject=="电子信息工程") or (college=="重点" and subject=="电子信息工程" )\
    or (age<=28 and subject=="计算机"):
    print("恭喜,您已获得我公司的面试机会!")
else:
```

```
print("抱歉,您未达到面试要求")
```

例 3-2 用户输入若干个分数,求所有分数的平均分。每输入一个分数后询问是否继续输入下一个分数,回答 yes 就继续输入下一个分数,回答 no 就停止输入分数。

```
numbers = []                                    #使用列表存放临时数据
while True:                                     #循环结构,参考 3.3 节
    x = input('请输入一个成绩:')
    try:                                        #异常处理结构,参考第 8 章
        numbers.append(float(x))
    except:
        print('不是合法成绩')
    while True:
        flag = input('继续输入吗? (yes/no)').lower()
        if flag not in ('yes', 'no'):           #限定用户输入内容必须为 yes 或 no
            print('只能输入 yes 或 no')
        else:
            break
    if flag == 'no':
        break

print(sum(numbers)/len(numbers))
```

例 3-3

例 3-3 编写程序,判断某个日期是该年第几天。

```
import time

def demo(year, month, day):                     #函数定义,参考第 5 章
    day_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]      #每个月的天数
    if year%400==0 or (year%4==0 and year%100!=0):                     #判断是否为闰年
        day_month[1] = 29                                              #闰年 2 月为 29 天
    if month == 1:
        return day
    else:
        return sum(day_month[:month-1]) + day

date = time.localtime()                                                #获取当前的日期时间
year, month, day = date[:3]
print(demo(year, month, day))
```

标准库 datetime 提供了 timedelta 对象,可以很方便地计算指定年、月、日、时、分、秒之前或之后的日期、时间,还提供了返回结果中包含“今天是今年第几天”“今天是本周第几天”等答案的 timetuple()函数等。

```
>>> import datetime
>>> Today = datetime.date.today()
>>> Today
```

```
datetime.date(2019, 9, 2)
>>> Today- datetime.date(Today.year, 1, 1) + datetime.timedelta(days=1)
datetime.timedelta(245)
>>> Today.timetuple().tm_yday                          #今天是今年的第几天
245
>>> Today.replace(year=2018)                           #替换日期中的年
datetime.date(2018, 9, 2)
>>> Today.replace(month=1)                             #替换日期中的月
datetime.date(2019, 1, 2)
>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2019, 12, 6, 16, 1, 6, 313898)
>>> now.replace(second=30)                             #替换日期时间中的秒
datetime.datetime(2019, 12, 6, 16, 1, 30, 313898)
>>> now + datetime.timedelta(days=5)                   #计算 5 天后的日期时间
datetime.datetime(2019, 12, 11, 16, 1, 6, 313898)
>>> now + datetime.timedelta(weeks=-5)                 #计算 5 周前的日期时间
datetime.datetime(2019, 11, 1, 16, 1, 6, 313898)
```

3.3 循环结构

3.3.1 for 循环与 while 循环

Python 提供了两种基本的循环结构：while 循环和 for 循环。其中，while 循环一般用于循环次数难以提前确定的情况，当然也可以用于循环次数确定的情况；for 循环一般用于循环次数可以提前确定的情况，尤其适用于枚举或遍历序列或迭代对象中元素的场合。循环结构之间可以互相嵌套，也可以与选择结构嵌套使用，用来实现更为复杂的逻辑。

while 循环和 for 循环常见的用法为

```
while 条件表达式:
    循环体
```

和

```
for 变量 in 序列或其他迭代对象:
    循环体
```

另外，while 循环和 for 循环都可以带 else 子句，如果循环因为条件表达式不成立而自然结束(不是因为执行了 break 而结束循环)，则执行 else 结构中的语句；如果循环是因为执行了 break 语句而导致循环提前结束，则不执行 else 中的语句。其语法形式为

```
while 条件表达式:
    循环体
else:
    else 子句代码块
```

和

```
for 取值 in 序列或迭代对象:
    循环体
else:
    else 子句代码块
```

3.3.2 循环结构的优化

为了优化程序以获得更高的效率和运行速度,在编写循环语句时,应尽量减少循环内部不必要的计算,将与循环变量无关的代码尽可能地提取到循环之外。对于使用多重循环嵌套的情况,应尽量减少内层循环中不必要的计算,尽可能地向外提。例如下面的代码,第二段明显比第一段的运行效率要高。

```
import time
digits = (1, 2, 3, 4)

start = time.time()
for i in range(1000):
    result = []
    for i in digits:
        for j in digits:
            for k in digits:
                result.append(i*100+j*10+k)
print(time.time()-start)
print(result)

start = time.time()
for i in range(1000):
    result = []
    for i in digits:
        i = i * 100
        for j in digits:
            j = j * 10
            for k in digits:
                result.append(i+j+k)
print(time.time()-start)
print(result)
```

在循环中应尽量使用局部变量,因为局部变量的查询和访问速度比全局变量略块,可参考 5.5 节的介绍。另外,在使用模块中的方法时,可以通过将其直接导入来减少查询次数和提高运行速度,可参考 1.4.8 节的介绍。

3.4 break 和 continue 语句

break 语句和 continue 语句在 while 循环和 for 循环中都可以使用,并且一般常与选择结构结合使用,以达到在特定条件得到满足时跳出循环的目的。一旦 break 语句被执行,将

使得整个循环提前结束。continue 语句的作用是终止本次循环,并忽略 continue 之后的所有语句,直接回到循环的顶端,提前进入下一次循环。

continue 的用法在后面章节会进行演示,下面的代码用来计算小于 100 的最大素数,请注意 break 语句和 else 子句的用法。

3.4

```
>>> for n in range(100, 1, -1):
    for i in range(2, n):
        if n%i == 0:
            break
    else:
        print(n)
        break
97
```

删除上面代码中最后一个 break 语句,可以用来输出 100 以内的所有素数。

3.5 案例精选

例 3-4 计算 1+2+3+…+100 的值。

```
s = 0
for i in range(1, 101):
    s = s + i
print('1+2+3+…+100=', s)
print('1+2+3+…+100=', sum(range(1, 101)))        #直接使用内置函数来实现题目的要求
```

例 3-5 枚举并输出序列中的元素。

```
a_list = ['a', 'b', 'mpilgrim', 'z', 'example']    #可以使用 enumerate()函数的参数 start 来简化
for i, v in enumerate(a_list):
    print('列表的第', i+1, '个元素是: ', v)
```

对于类似元素遍历的问题,同样也可以使用 while 循环来解决,但是代码要麻烦一些,可读性也较差,例如:

```
>>> a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
>>> i = 0
>>> number = len(a_list)
>>> while i < number:
    print('列表的第', i+1, '个元素是: ', a_list[i])
    i += 1
```

例 3-6 求 1～100 能被 7 整除,但不能同时被 5 整除的所有整数。

```
for i in range(1, 101):
    if i%7 == 0 and i%5 != 0:
        print(i)
```

例 3-7 输出3位“水仙花数”。所谓n位水仙花数是指一个n位的十进制数，其各位数字的n次方和恰好等于该数本身。例如，153是水仙花数，因为 $153=1^3+5^3+3^3$。

例 3-7

```
for i in range(10**2, 10**3):
    bai, shi, ge = map(int, str(i))
    if bai**3 + shi**3 + ge**3 == i:
        print(i)
```

例 3-8 统计考试成绩中优秀、良、中、及格、不及格的人数。

方法一：

```
scores = [89, 70, 49, 87, 92, 84, 73, 71, 78, 81, 90, 37,
          77, 82, 81, 79, 80, 82, 75, 90, 54, 80, 70, 68, 61]

groups = {'优秀':0, '良':0, '中':0, '及格':0, '不及格':0}
for score in scores:
    if score >= 90:
        groups['优秀'] = groups['优秀'] + 1
    elif score >= 80:
        groups['良'] = groups['良'] + 1
    elif score >= 70:
        groups['中'] = groups['中'] + 1
    elif score >= 60:
        groups['及格'] = groups['及格'] + 1
    else:
        groups['不及格'] = groups['不及格'] + 1
print(groups)
```

方法二：

```
from itertools import group by
scores = [89, 70, 49, 87, 92, 84, 73, 71, 78, 81, 90, 37,
          77, 82, 81, 79, 80, 82, 75, 90, 54, 80, 70, 68, 61]

def classify(score):
    if score >= 90:
        return '优秀'
    elif score >= 80:
        return '良'
    elif score >= 70:
        return '中'
    elif score >= 60:
        return '及格'
    else:
        return '不及格'

groups = {category:len(tuple(score)) for category, score in groupby(sorted(scores), classify)}
```

```
print(groups)
```

方法三：

```
from collections import Counter
from pandas import cut                                    #需要先安装扩展库 pandas 才能使用

scores = [89, 70, 49, 87, 92, 84, 73, 71, 78, 81, 90, 37,
          77, 82, 81, 79, 80, 82, 75, 90, 54, 80, 70, 68, 61]
groups = Counter(cut(scores,[0, 60, 70, 80, 90, 101],
                     labels=['不及格', '及格', '中', '良', '优秀'],
                     right=False))
print(groups)
```

例 3-9 打印九九乘法表。

```
for i in range(1, 10):
    for j in range(1, i+1):
        print('{0}*{1}={2}'.format(i, j, i*j).ljust(6), end=' ')
    print()
```

例 3-10 求 200 以内能被 17 整除的最大正整数。

```
for i in range(200, 0, -1):
    if i%17 == 0:
        print(i)
        break
```

例 3-11 判断一个正整数是否为素数。

```
import math

n = input('Input an integer:')
n = int(n)
m = math.ceil(math.sqrt(n)+1)
for i in range(2, m):
    if n% i ==0 and i<n:
        print('No')
        break
else:
    print('Yes')
```

math 是用于数学计算的标准库，除了用于平方根函数 sqrt()和向上取整函数 ceil()，还提供了最大公约数函数 gcd()，sin()、asin()等三角函数与反三角函数，弧度与角度转换函数 degrees()、radians()，误差函数 erf()、剩余误差函数 erfc()、伽马函数 gamma()，对数函数 log()、log2()、log10()，阶乘函数 factorial()，常数 pi 和 e，等等。

例 3-12 鸡兔同笼问题。假设共有鸡、兔 30 只，脚 90 只，求鸡、兔各有多少只。

```
for ji in range(0, 31):
```

```
    if 2*ji + (30-ji)*4 == 90:
        print('ji:', ji, ' tu:', 30-ji)
        break
```

例 3-12

例 3-13 编写程序，输出由 1、2、3、4 这 4 个数字组成的每位数都不相同的所有三位数。

```
digits = (1, 2, 3, 4)
for i in digits:
    for j in digits:
        for k in digits:
            if i!=j and j!=k and i!=k:
                print(i*100+ j*10+k)
```

例 3-13

从代码优化的角度来讲，上面这段代码并不是很好，其中有些判断完全可以在外层循环来做，从而提高运行效率，可以扫描二维码观看优化方法以及更多实现。

例 3-14 有一箱苹果，4 个 4 个地数最后余下 1 个，5 个 5 个地数最后余下 2 个，9 个 9 个地数最后余下 7 个。编写程序计算这箱苹果至少有多少个。

先确定除以 9 余 7 的最小整数，对这个数字重复加 9，如果得到的数字除以 5 余 2 就停止；然后对得到的数字重复加 45，如果得到的数字除以 4 余 1 就停止。这时得到的数字就是题目的答案。

```
from itertools import count

for num in count(16, 9):
    if num%5 == 2:
        break
for result in count(num, 45):
    if result%4 == 1:
        break
print(result)
```

例 3-15 编写程序，计算组合数 $C(n,i)$，即从 n 个元素中任选 i 个，有多少种选法。

根据组合数定义，需要计算 3 个数的阶乘，在很多编程语言中都很难直接使用整型变量表示大数的阶乘结果，虽然 Python 并不存在这个问题，但是计算大数的阶乘仍需要相当多的时间。本例提供另一种计算方法：以 Cni(8,3)为例，按定义式展开为 Cni(8,3)＝8!/3!/(8－3)!＝(8×7×6×5×4×3×2×1)/(3×2×1)/(5×4×3×2×1)，对于(5,8]区间的数，分子上出现一次而分母上没出现；(3,5]区间的数在分子、分母上各出现一次；[1,3]区间的数分子上出现一次而分母上出现两次。

```
def Cni1(n, i):
    if not (isinstance(n, int) and isinstance(i, int) and n>=i):
        print('n and i must be integers and n must be larger than or equal to i.')
        return
    result = 1
    Min, Max = sorted((i, n-i))
```

```
    for i in range(n, 0, -1):
        if i > Max:
            result *= i
        elif i <= Min:
            result //= i
    return result
print(Cni1(6, 2))
```

当然,也可以使用 math 库中的阶乘函数直接按组合数定义实现。

```
>>> def Cni2(n, i):
    import math                              #Python 3.8 中提供了 math.comb()函数,请自行查阅
    return int(math.factorial(n)/math.factorial(i)/math.factorial(n-i))
>>> Cni2(6, 2)
15
```

还可以直接使用 Python 标准库 itertools 提供的组合函数。

```
>>> import itertools
>>> len(tuple(itertools.combinations(range(60), 2)))
1770
```

计算组合数时如果数值 n 和 i 较大,建议使用前面定义的 Cni1()函数,不建议使用 combinations()和 Cni2()函数,因为这会增加大量的额外操作甚至导致死机。除了组合函数 combinations(),itertools 还提供了排列函数 permutations()、用于循环遍历可迭代对象元素的函数 cycle()、根据一个序列的值对另一个序列进行过滤的函数 compress()、根据函数返回值对序列进行分组的函数 groupby()、计算排列数的函数 perm()(Python 3.8 以上可用)。

```
>>> import itertools
>>> x = 'Private Key'
>>> y = itertools.cycle(x)                  #循环遍历序列中的元素
>>> for i in range(20):
    print(next(y), end=',')
P, r, i, v, a, t, e, , K, e, y, P, r, i, v, a, t, e, , K,
>>> for i in range(5):
    print(next(y), end=',')
e, y, P, r, i,
>>> x = range(1, 20)
>>> y = (1, 0)*9 + (1,)
>>> y
(1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
>>> list(itertools.compress(x, y))          #根据一个序列的值对另一个序列进行过滤
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> def group(v):                           #用来确定分组的自定义函数
    if v > 10:
        return 'greater than 10'
```

```
    elif v < 5:
        return 'less than 5'
    else:
        return 'between 5 and 10'
>>> x = range(20)                                    #待分组的数据,要求已按序排列
>>> y = itertools.groupby(x, group)                  #根据函数返回值对序列元素进行分组
>>> for k, v in y:
    print(k, ':', list(v))
less than 5 : [0, 1, 2, 3, 4]
between 5 and 10 : [5, 6, 7, 8, 9, 10]
greater than 10 : [11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> list(itertools.permutations([1, 2, 3, 4], 3))    #从4个元素中任选3个的所有排列
输出结果(略)
>>> x = itertools.permutations([1, 2, 3, 4], 4)      #4个元素全排列
>>> next(x)                                          #获取下一个排列
(1, 2, 3, 4)
>>> next(x)
(1, 2, 4, 3)
>>> next(x)
(1, 3, 2, 4)
```

例 3-16 编写程序,计算理财产品收益。此处假设利息和本金一起滚动。

```
def licai(base, rate, days):
    #初始投资金额
    result = base
    #整除,用来计算一年可以滚动多少期
    times = 365//days
    for i in range(times):
        result = result + result*rate/365*days
    return result
#14天理财,利率为0.0385,投资10万元
print(licai(100000, 0.0385, 14))
```

例 3-17 计算前 n 个自然数的阶乘之和 $1!+2!+3!+\cdots+n!$ 的值。

例 3-17

```
def factorialBefore(n):
    result, t = 1, 1
    for i in range(2, n+1):
        t *= i
        result += t
    return result
print(factorialBefore(6))
```

例 3-18 验证 6174 猜想。1955 年,卡普耶卡(D. R. Kaprekar)对 4 位数字进行了研究,发现一个规律:对任意各位数字不相同的 4 位数,使用各位数字能组成的最大数减去能组成的最小数,对得到的差重复这个操作,最终会得到 6174 这个数字,并且这个操作最多不

会超过 7 次。下面的代码可以结束并且没有任何输出,说明 6174 猜想是正确的。

例 3-18

```
from string import digits
from itertools import combinations

for item in combinations(digits, 4):
    times = 0
    while True:
        #当前选择的 4 个数字能够组成的最大数和最小数
        big = int(''.join(sorted(item, reverse=True)))
        little = int(''.join(sorted(item)))
        difference = big - little
        times = times + 1
        #如果最大数和最小数相减得到 6174 就退出
        #否则就对得到的差重复这个操作
        #最多 7 次,总能得到 6174
        if difference == 6174:
            if times > 7:
                print(times)
            break
        else:
            item = str(difference)
```

例 3-19 检测序列中的元素是否满足严格升序关系。

例 3-19

```
def lessThan1(seq):
    for index, value in enumerate(seq[:-1]):
        if value >= seq[index+1]:
            return False
    return True

def lessThan2(seq):
    func = lambda x, y: x < y
    return all(map(func, seq[:-1], seq[1:]))

tests = ('abcdeff', [1, 2, 3, 5, 4], (3, 5, 7, 9))
for test in tests:
    print(lessThan1(test), lessThan2(test))
```

本 章 小 结

(1) 几乎所有合法的 Python 表达式都可以作为选择结构和循环结构中的条件表达式。

(2) Python 的关系运算符可以连续使用,例如,3< 4< 5> 2 的值为 True。

(3) 数字 0、0.0、0j、逻辑假 False、空列表[]、空集合或空字典{}、空元组()、空字符串"、

空值 None 以及任意与这些值等价的值作为条件表达式时均被认为条件不成立，否则认为条件表达式成立。

(4) 逻辑运算符 and 和 or 具有短路求值或惰性求值的特点，即只计算必须计算的表达式的值。

(5) 选择结构和循环结构往往会互相嵌套使用来实现复杂的业务逻辑。

(6) 关键字 elif 表示 else if 的意思。

(7) 应优先考虑使用 for 循环，尤其是列表、元组、字典或其他 Python 序列元素遍历的场合。

(8) 编写循环语句时，应尽量减少内循环中的无关计算，对循环进行必要的优化。

(9) for 循环和 while 循环都可以带有 else 子句，如果循环因为条件表达式不满足而自然结束时，执行 else 子句中的代码；如果循环是因为执行了 break 语句而结束，则不执行 else 子句中的代码。

(10) break 语句用来提前结束其所在循环，continue 语句用来提前结束本次循环并进入下一次循环。

(11) 除非 break 和 continue 语句可以让代码变得更简单或更清晰，否则不要轻易使用。

习　题

1. Python 提供了两种基本的循环结构：_____和_____。

2. 分析逻辑运算符 or 的短路求值特性。

3. 编写程序，运行后用户输入 4 位整数作为年份，判断其是否为闰年。如果年份能被 400 整除，则为闰年；如果年份能被 4 整除但不能被 100 整除也为闰年。

4. 编写程序，生成一个包含 50 个随机整数的列表，然后删除其中所有奇数(提示：从后向前删)。

5. 编写程序，生成一个包含 20 个随机整数的列表，然后对其中偶数下标的元素进行降序排列，奇数下标的元素不变(提示：使用切片)。

6. 编写程序，用户从键盘输入小于 1000 的整数，对其进行因式分解。例如，10＝2×5，60＝2×2×3×5。

7. 编写程序，至少使用两种不同的方法计算 100 以内所有奇数的和。

8. 编写程序，输出所有由 1、2、3、4 这四个数字组成的素数，并且在每个素数中每个数字只使用一次。

9. 编写程序，实现分段函数计算，如表 3-1 所示。

表 3-1　分段函数计算

x	y
$x<0$	0
$0\leqslant x<5$	x

续表

$5 \leqslant x < 10$	$3x-5$
$10 \leqslant x < 20$	$0.5x-2$
$20 \leqslant x$	0

10. 查阅资料编写程序,模拟蒙蒂霍尔悖论游戏。

11. 查阅资料编写程序,模拟尼姆游戏。

12. 查阅资料编写程序,输出小于 100 的所有丑数。

第 4 章　字符串与正则表达式

最早的字符串编码是美国标准信息交换码 ASCII，仅对 10 个数字、26 个大写英文字母、26 个小写英文字母及一些其他符号进行了编码。ASCII 采用 1 字节来对字符进行编码，因此最多只能表示 256 个符号。

随着信息技术的发展和信息交换的需要，各国的文字都需要进行编码，于是分别设计了不同的编码格式，并且编码格式之间有较大的区别，其中我国常用的编码有 UTF-8、GB2312、GBK、CP936 等。采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的内容可能会不同，在理解其内容时必须了解编码规则并进行正确的解码。其中，UTF-8 编码是国际通用的编码，以 1 字节表示英语字符（兼容 ASCII），以 3 字节表示常见汉字，对全世界所有国家需要用到的字符进行了编码。

GB2312 是我国制定的中文编码标准，使用 1 字节表示英语，2 字节表示中文；GBK 是 GB2312 的扩充，而 CP936 是微软公司在 GBK 基础上开发的编码方式。GB2312、GBK 和 CP936 都是使用 2 字节表示中文。

Python 3.x 完全支持中文，无论是一个数字、英文字母，还是一个汉字，都按一个字符对待和处理。在 Python 3.x 中可以使用中文作为变量名。

```
>>> s = '中国山东烟台'
>>> len(s)
6
>>> s = 'SDIBT'
>>> len(s)
5
>>> s = '中国山东烟台 SDIBT'
>>> len(s)
11
>>> 姓名 = '张三'
>>> 年龄 = 40
>>> print(姓名)
张三
>>> print(年龄)
40
```

4.1　字　符　串

在 Python 中，字符串使用单引号、双引号、三单引号或三双引号作为界定符，并且不同的界定符之间可以互相嵌套。除了支持序列通用方法（包括比较、计算长度、元素访问、切片等操作）以外，字符串类型还支持一些特有的操作方法，例如，格式化、字符串查找、字符串替换等。由于字符串属于不可变序列，因此不能对字符串对象进行元素增加、修改与删除等操作。字符串对象提供的 replace()、translate()以及其他类似方法并不是对原字符串直接修改替换，而是返回一个修改替换后的结果字符串。

4.1.1　字符串格式化

字符串格式化用来把整数、实数、列表等对象转化为特定格式的字符串。Python 中字符串格式化的格式如图 4-1 所示，%符号之前的部分为格式字符串，之后的部分为需要进行

格式化的内容。

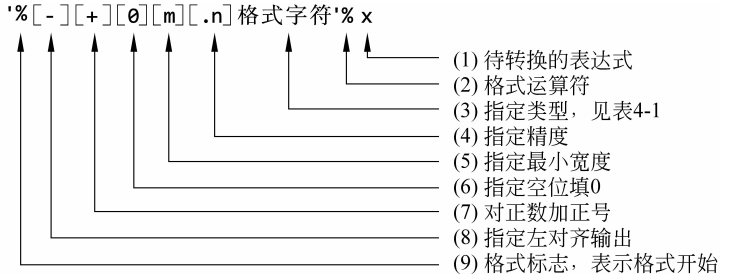

图 4-1　字符串格式化

Python 支持大量的格式字符,常见的格式字符如表 4-1 所示。

表 4-1　格式字符

格式字符	说　　明	格式字符	说　　明
%s	字符串（采用 str()的显示）	%e	指数（基底写为 e）
%r	字符串（采用 repr()的显示）	%E	指数（基底写为 E）
%c	单个字符	%f、%F	浮点数
%d	十进制整数	%g	指数(e)或浮点数（根据显示长度）
%i	十进制整数	%G	指数(E)或浮点数（根据显示长度）
%o	八进制整数	%%	字符"%"
%x	十六进制整数		

下面的代码简单演示了字符串格式化的用法：

```
>>> x = 1235
>>> "%o" %x
"2323"
>>> "%x" %x
"4d3"
>>> "%e" %x
"1.235000e+03"
>>> "%s" %65              #类似于 str()
"65"
>>> '%d, %c' %(65, 65)    #如果要求格式化的对象多于一个,要放在元组中
'65,A'
>>> "%d" %"555"           #试图将字符串转换为整数进行输出,抛出异常
TypeError: %d format: a number is required, not str
>>> '%s' %[1, 2, 3]
'[1, 2, 3]'
>>> str((1, 2, 3))        #可以使用 str()函数将任意类型数据转换为字符串
'(1, 2, 3)'
>>> str([1, 2, 3])
```

```
'[1, 2, 3]'
```

目前 Python 社区更推荐使用 format()方法进行格式化，该方法更加灵活，不仅可以使用位置进行格式化，还支持使用与位置无关的参数名字来进行格式化，并且支持序列解包格式化字符串，为程序员提供了非常大的方便。例如：

```
>>> print("The number {0:,} in hex is: {0:#x}, the number {1} in oct is {1:#o}".format(5555, 55))
The number 5555 in hex is: 0x15b3, the number 55 in oct is 0o67
>>> print("The number {1:,} in hex is: {1:#x}, the number {0} in oct is {0:#o}".format(5555, 55))
The number 55 in hex is: 0x37, the number 5555 in oct is 0o12663
>>> print("my name is {name}, my age is {age}, and my QQ is {qq}".format(name="Dong Fuguo", qq=
"306467355", age=37))
my name is Dong Fuguo, my age is 37, and my QQ is 306467355
>>> position = (5, 8, 13)
>>> print("X:{0[0]};Y:{0[1]};Z:{0[2]}".format(position))
X:5;Y:8;Z:13
>>> weather = [("Monday", "rainy"), ("Tuesday", "sunny"),
               ("Wednesday", "sunny"), ("Thursday", "rainy"),
               ("Friday", "Cloudy")]
>>> formatter = "Weather of '{0[0]}' is '{0[1]}'".format
>>> for item in map(formatter, weather):
    print(item)
Weather of 'Monday' is 'rainy'
Weather of 'Tuesday' is 'sunny'
Weather of 'Wednesday' is 'sunny'
Weather of 'Thursday' is 'rainy'
Weather of 'Friday' is 'Cloudy'
```

4.1.1

关于内置函数 map()的介绍可以参考 5.8 节的介绍。上面最后一段代码也可以改为下面的写法：

```
>>> for item in weather:
    print(formatter(item))
>>> print('{0:.4f}'.format(10/3))              #指定小数位数
3.3333
>>> print('{0:.2%}'.format(1/3))               #格式化为百分数
33.33%
>>> print('{0:>10.2%}'.format(1/3))            #符号>表示右对齐,<表示左对齐,^表示居中对齐
    33.33%
```

Python 3.6.x 之后的版本支持在数字常量的中间位置使用单个下画线作为分隔符来提高可读性，相应地，字符串格式化方法 format()也提供了对下画线的支持。

```
>>> print('{0:_},{0:#_x}'.format(10000000))
10_000_000,0x98_9680
```

从 Python 3.6.x 开始支持一种新的字符串格式化方式，官方叫作 Formatted String Literals，其含义与字符串对象的 format()方法类似，但形式更加简洁。其中花括号里面的

变量名表示占位符,在进行格式化时,使用前面定义的同名变量的值对格式化字符串中的占位符进行替换。在 Python 3.8 之后的版本中,支持 print('{width:}')的用法,可自行测试。

```
>>> width = 8
>>> height = 6
>>> print(f'Rectangle of {width}*{height}\nArea:{width*height}')
Rectangle of 8*6
Area:48
```

4.1.2 字符串常用方法

可以使用 dir("")查看字符串操作所有方法列表,使用内置函数 help()查看每个方法的帮助。字符串也是 Python 序列的一种,除了本节介绍的字符串处理方法,很多 Python 内置函数也支持对字符串的操作,例如用来计算序列长度的 len()函数,求最大值的 max()函数,见 1.4.6 节。

1. find()、rfind()、index()、rindex()、count()

find()和 rfind()方法分别用来查找一个字符串在当前字符串指定范围(默认是整个字符串)中首次和最后一次出现的位置,如果不存在则返回-1;index()和 rindex()方法用来返回一个字符串在当前字符串指定范围中首次和最后一次出现的位置,如果不存在则抛出异常;count()方法用来返回一个字符串在当前字符串中出现的次数。

```
>>> s = "apple,peach,banana,peach,pear"
>>> s.find("peach")            #返回第一次出现的位置
6
>>> s.find("peach", 7)         #从指定位置开始查找
19
>>> s.find("peach", 7, 20)     #在指定范围中查找
-1
>>> s.rfind('p')               #从字符串尾部向前查找
25
>>> s.index('p')               #返回首次出现的位置
1
>>> s.index('pe')
6
>>> s.index('pear')
25
>>> s.index('ppp')             #指定子字符串不存在时抛出异常
ValueError: substring not found
>>> s.count('p')               #统计子字符串出现的次数
5
```

2. split()、rsplit()、partition()、rpartition()

split()和 rsplit()方法分别用来以指定字符为分隔符,从字符串左端和右端开始将其分隔成多个字符串,并返回包含分隔结果的列表;

split 方法