

第 3 章

Python 控制语句

对于 Python 程序中的执行语句,默认是按照书写顺序依次执行的,这时称这样的语句是顺序结构的。但是,仅有顺序结构还是不够的,因为有时需要根据特定的情况,有选择地执行某些语句,这时就需要一种选择结构的语句。另外,有时还可以在给定条件下重复执行某些语句,这时称这些语句是循环结构的。有了这三种基本的结构,就能够构建任意复杂的程序了。

3.1 选择结构



视频讲解

三种基本程序结构中的选择结构,可用 if 语句、if...else 语句和 if...elif...else 语句实现。

3.1.1 if 语句

Python 的 if 语句的功能跟其他语言非常相似,都是用来判定给出的条件是否满足,然后根据判断的结果(即真或假)决定是否执行给出的操作。if 语句是一种单选结构,它选择的是做或不做。它由三部分组成:关键字 if 本身、测试条件真假的表达式(简称为条件表达式)和表达式结果为真(即表达式的值为非 0)时要执行的代码。if 语句的语法形式如下所示:

if 表达式:

语句 1

if 语句的流程图如图 3-1 所示。

if 语句的表达式用于判断条件,可以用>(大于)、<(小于)、==(等于)、>=(大于或等于)、<=(小于或等于)来表示其关系。

现在用一个示例程序来演示一下 if 语句的用法。程序很简单,只要用户输入一个整数,如果这个数字大于 6,那么就输出一行字符串;否则,直接退出程序。代码如下:

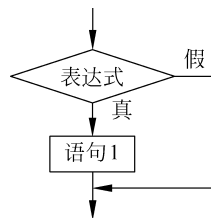


图 3-1 if 语句的流程图

```
# 比较输入的整数是否大于 6
a = input("请输入一个整数: ")    # 取得一个字符串
a = int(a)                        # 将字符串转换为整数
if a > 6:
    print(a, "大于 6")
```

通常,每个程序都会有输入输出,这样可以与用户进行交互。用户输入一些信息,你会对他输入的内容进行一些适当的操作,然后再输出用户想要的结果。Python 可以用 input 进行输入,用 print 进行输出,这些都是简单的控制台输入输出,复杂的有处理文件等。

3.1.2 if...else 语句

上面的 if 语句是一种单选结构,也就是说,如果条件为真(即表达式的值为非零),那么执行指定的操作;否则就会跳过该操作。而 if...else 语句是一种双选结构,在两种备选行动中选择哪一个的问题。if...else 语句由五部分组成:关键字 if、测试条件真假的表达式、表达式结果为真(即表达式的值为非零)时要执行的代码,以及关键字 else 和表达式结果为假(即表达式的值为零)时要执行的代码。if...else 语句的语法形式如下:

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

if...else 语句的流程图如图 3-2 所示。

下面对上面的示例程序进行修改,以演示 if...else 语句的使用方法。程序很简单,只要用户输入一个整数,如果这个数字大于 6,那么就输出一行信息,指出输入的数字大于 6;否则,输出另一行字符串,指出输入的数字小于或等于 6。代码如下:

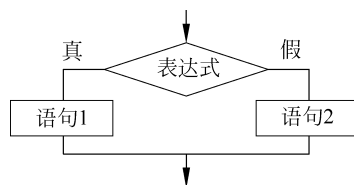


图 3-2 if...else 语句的流程图

```
a = input("请输入一个整数: ") # 取得一个字符串
a = int(a)                     # 将字符串转换为整数
if a > 6:
    print(a, "大于6")
else:
    print(a, "小于或等于6")
```

【例 3-1】 任意输入三个数字,按从小到大的顺序输出。

分析: 先将 x 与 y 比较,把较小者放入 x 中,较大者放入 y 中;再将 x 与 z 比较,把较小者放入 x 中,较大者放入 z 中,此时 x 为三者中的最小者;最后将 y 与 z 比较,把较小者放入 y 中,较大者放入 z 中,此时 x、y、z 已按由小到大的顺序排列。

```
x = input('x = ')           # 输入 x
y = input('y = ')           # 输入 y
z = input('z = ')           # 输入 z
if x > y:
    x, y = y, x              # x, y 互换
if x > z:
    x, z = z, x              # x, z 互换
if y > z:
    y, z = z, y              # y, z 互换
print(x, y, z)
```

假如 x、y、z 分别输入 1、4、3，以上代码输出结果：

```
x=1 ✓ (输入 x 的值, ✓ 表示回车)
y=4 ✓ (输入 y 的值)
z=3 ✓ (输入 z 的值)
1 3 4
```

其中“x, y = y, x”这种语句是同时赋值,将赋值号右侧的表达式依次赋给左侧的变量。例如,“x, y = 1, 4”就相当于“x=1; y=4”的效果,可见 Python 语法多么简洁。

3.1.3 if...elif...else 语句

有时候,需要在多组动作中选择一组执行,这时就会用到多选结构,对于 Python 语言来说就是 if...elif...else 语句。该语句可以利用一系列条件表达式进行检查,并在某个表达式为真的情况下执行相应的代码。需要注意的是,虽然 if...elif...else 语句的备选动作较多,但是有且只有一组动作被执行。该语句的语法形式如下:

```
if 表达式 1:
    语句 1
elif 表达式 2:
    语句 2
    :
elif 表达式 n:
    语句 n
else:
    语句 n+1
```

注意：最后一个 elif 子句之后的 else 子句没有进行条件判断,它实际上处理跟前面所有条件都不匹配的情况,所以 else 子句必须放在最后。

if...elif...else 语句的流程图如图 3-3 所示。

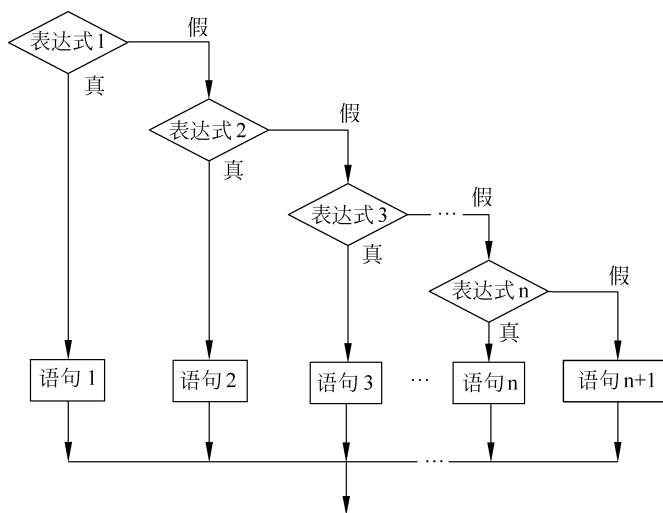


图 3-3 if...elif...else 语句的流程图

下面继续对上面的示例程序进行修改,以演示 if...elif...else 语句的使用方法。我们还是要用户输入一个整数,如果这个数字大于 6,那么就输出一行信息,指出输入的数字大于 6;如果这个数字小于 6,则输出另一行字符串,指出输入的数字小于 6;否则,指出输入的数字等于 6。具体的代码如下:

```
a = input("请输入一个整数: ") # 取得一个字符串
a = int(a)                     # 将字符串转换为整数
if a > 6:
    print ( a, "大于 6")
elif a == 6:
    print ( a, "等于 6")
else:
    print ( a, "小于 6")
```

【例 3-2】 输入学生的成绩 score,按分数输出其等级: $\text{score} \geq 90$ 为优, $90 > \text{score} \geq 80$ 为良, $80 > \text{score} \geq 70$ 为中等, $70 > \text{score} \geq 60$ 为及格, $\text{score} < 60$ 为不及格。

```
score = int(input("请输入成绩")) # int()转换字符串为整型
if score >= 90:
    print("优")
elif score >= 80:
    print("良")
elif score >= 70:
    print("中")
elif score >= 60:
    print("及格")
else:
    print("不及格")
```

说明: 三种选择语句中,条件表达式都是必不可少的组成部分。当条件表达式的值为 0 时,表示条件为假;当条件表达式的值为非 0 时,表示条件为真。那么哪些表达式可以作为条件表达式呢?基本上,最常用的是关系表达式和逻辑表达式。例如:

```
if a == x and b == y:
    print ("a = x, b = y")
```

除此之外,条件表达式还可以是任何数值类型表达式,甚至字符串也可以。例如:

```
if 'a': # 'abc':也可以
    print ("a = x, b = y")
```

另外,C 语言是用花括号 {} 来区分语句体,但是 Python 的语句体是用缩进形式来表示的,如果缩进不正确,则会导致逻辑错误。

3.1.4 pass 语句

Python 提供了一个关键字 `pass`, 类似于空语句, 可以用在类和函数的定义中或者选择结构中。当暂时没有确定如何实现功能, 或者为以后的软件升级预留空间, 又或其他类型功能时, 可以使用该关键字来“占位”。例如下面的代码是合法的:

```
if a < b:
    pass      # 什么操作也不做
else:
    z = a
class A:      # 类的定义
    pass
def demo():   # 函数的定义
    pass
```

3.2 循环结构

程序在一般情况下是按顺序执行的。编程语言提供了各种控制结构, 允许更复杂的执行路径。循环语句允许执行一个语句或语句组多次, Python 提供了 `while` 循环(在 Python 中没有 `do...while` 循环)和 `for` 循环。



视频讲解

3.2.1 while 语句

Python 编程中 `while` 语句用于循环执行程序, 即在某条件下, 循环执行某段程序, 以处理需要重复处理的相同任务。`while` 语句的流程图如图 3-4 所示。

其基本形式为:

while 判断条件:

 执行语句

判断条件可以是任何表达式, 任何非零或非空(`null`)的值均为真。当判断条件为假时, 循环结束。执行语句可以是单个语句或语句块。注意程序中的冒号和缩进。例如:

```
count = 0
while count < 9:
    print ('The count is:', count)
    count = count + 1
print ("Good bye!")
```

以上代码输出结果:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

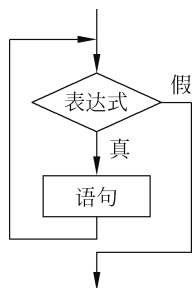


图 3-4 while 语句的流程图

```
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

此外,while 语句中判断条件还可以是个常值,表示循环必定成立。例如:

```
count = 0
while 1:                                # 判断条件是个常值 1
    print ('The count is:', count)
    count = count + 1
print ("Good bye!" )
```

这样就形成无限循环,可以借助后面学习的 break 语句结束循环。

【例 3-3】 输入两个正整数,求它们的最大公约数。

分析: 求最大公约数可以用“辗转相除法”,方法如下:

- (1) 比较两数 m 和 n,并使 m 大于 n。
- (2) 将 m 作为被除数,n 作为除数,相除后余数为 r。
- (3) 循环判断 r,若 r=0,则 n 为最大公约数,结束循环。若 r≠0,执行步骤 $m \leftarrow n, n \leftarrow r$;
将 m 作为被除数,n 作为除数,相除后余数为 r。

```
num1 = int(input("输入第一个数字: ")) # 用户输入两个数字
num2 = int(input("输入第二个数字: "))
m = num1
n = num2
if m < n:                                # m,n 交换值
    t = m
    m = n
    n = t
r = m % n
while r!= 0:
    m = n
    n = r
    r = m % n
print( num1,"和", num2,"的最大公约数为", n)
```

以上代码执行输出结果:

```
输入第一个数字: 36
输入第二个数字: 48
36 和 48 的最大公约数为 12
```

3.2.2 for 语句



视频讲解

for 语句可以遍历任何序列的项目,如一个列表、元组或者一个字符串。

1. for 循环的语法

for 循环的语法格式如下:

for 循环索引值 in 序列

循环体

for 语句的执行过程是:每次循环,都判断循环索引值是否还在序列中,如果在,则取出该值提供给循环体内的语句使用;如果不在,则结束循环。例如:

for 循环把字符串中字符遍历出来。

```
for letter in 'Python':           # 第一个实例
    print( '当前字母 :', letter )
```

以上实例输出结果:

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : h
当前字母 : o
当前字母 : n
```

for 循环把列表中元素遍历出来。

```
fruits = ['banana', 'apple', 'mango']
for fruit in fruits:           # 第二个实例
    print ( '元素 :', fruit)
print( "Good bye!" )
```

会依次打印 fruits 的每一个元素。以上实例输出结果:

```
元素 : banana
元素 : apple
元素 : mango
Good bye!
```

【例 3-4】 计算 1~10 的整数之和,可以用一个 sum 变量做累加。

```
sum = 0
for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    sum = sum + x
print(sum)
```

如果要计算 1~100 的整数之和,从 1 写到 100 有点困难,幸好 Python 提供一个 range() 内置函数,可以生成一个整数序列,再通过 list() 函数可以转换为 list。

例如,range(0, 5) 或 range(5) 生成的序列是从 0 开始到小于 5 的整数,不包括 5。实例如下:

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

range(1, 101) 就可以生成 1~100 的整数序列。计算 1~100 的整数之和代码如下:

```
sum = 0
for x in range(1,101):
    sum = sum + x
print(sum)
```

请自行运行上述代码,看看结果是不是当年高斯同学心算出的 5050。

2. 通过索引循环

对于一个列表,另外一种执行循环的遍历方式是通过索引(元素下标)。实例如下:

```
fruits = ['banana', 'apple', 'mango']
for i in range(len(fruits)):
    print( '当前水果 :', fruits[i] )
print ("Good bye!")
```

以上实例输出结果:

```
当前水果 : banana
当前水果 : apple
当前水果 : mango
Good bye!
```

以上实例使用了内置函数 len() 和 range()。函数 len() 返回列表的长度,即元素的个数,通过索引 i 访问每个元素 fruits[i]。

3.2.3 continue 和 break 语句

continue 语句的作用是终止当前循环,并忽略 continue 之后的语句,然后回到循环的顶端,提前进入下一次循环。

break 语句在 while 循环和 for 循环中都可以使用,一般放在 if 选择结构中,一旦 break 语句被执行,将使得整个循环提前结束。

除非 break 语句让代码更简单或更清晰,否则不要轻易使用。

【例 3-5】 continue 和 break 用法示例。

```
#continue 和 break 用法
i = 1
```

```

while i < 10:
    i += 1
    if i % 2 > 0:           # 非双数时跳过输出
        continue
    print(i)               # 输出双数 2、4、6、8、10
    i = 1
while 1:                  # 循环条件为 1 必定成立
    print(i)              # 输出 1~10
    i += 1
    if i > 10:             # 当 i 大于 10 时跳出循环
        break

```

3.2.4 循环嵌套

Python 语言允许在一个循环体里面嵌入另一个循环。可以在循环体内嵌入其他的循环体,如在 while 循环中可以嵌入 for 循环;也可以在 for 循环中嵌入 while 循环。嵌套层次一般不超过 3 层,以保证可读性。

注意:

(1) 循环嵌套时,外层循环和内层循环间是包含关系,即内层循环必须被完全包含在外层循环中。

(2) 当程序中出现循环嵌套时,程序每执行一次外层循环,其内层循环必须循环所有的次数(即内层循环结束)后,才能进入外层循环的下一循环。

【例 3-6】 打印九九乘法表。

```

for i in range(1,10):
    for j in range(1,i+1):
        print(i,'*',j,'=',i*j,'\t',end="")    # end="" 作用是不换行
    print("")                                   # 仅起换行作用

```

以上代码执行输出结果如图 3-5 所示。

```

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

```

图 3-5 九九乘法表

【例 3-7】 使用嵌套循环输出 2~100 的素数。

素数是除 1 和本身外,不能被其他任何整数整除的整数。判断一个数 m 是否为素数,只要依次用 2, 3, 4, ..., $m-1$ 作为除数去除 m ,如果有一个能被整除, m 就不是素数。

```

m = int(input("请输入一个整数"))
j = 2

```

```

while j <= m-1 :
    if m % j == 0: break # 退出循环
    j = j + 1
if (j > m-1) :
    print (m, "是素数")
else:
    print (m, "不是素数")

```

应用上述代码,对于一个非素数而言,判断过程往往可以很快结束。例如,判断 30 009 时,因为该数能被 3 整除,所以只需判断 $j=2, 3$ 两种情况。而判断一个素数尤其是当该数较大时,例如判断 30 011,则要从 $j=2, 3, 4, \dots$,一直判断到 30 010 都不能被整除,才能得出其为素数的结论。实际上,只要从 2 判断到 $\sqrt{m}$,若 m 不能被其中任何一个数整除,则 m 即为素数。

```

# 找出 100 以内的所有素数
import math                                # 导入 math 数学模块
m = 2
while m < 100 :                            # 外层循环
    j = 2
    while j <= math.sqrt(m) :              # 内层循环, math.sqrt() 是求平方根
        if m % j == 0: break               # 退出内层循环
        j = j + 1
    if (j > math.sqrt(m)) :
        print (m, "是素数")
    m = m + 1
print ("Good bye!")

```

【例 3-8】 使用嵌套循环输出如图 3-6 所示的金字塔图案。

```

      *
     ***
    *****
   *
  *
 *

```

图 3-6 金字塔图案

分析: 观察图形包含 8 行,因此外层循环执行 8 次;每行内容由两部分组成: 空格和星号。假设第 1 行星号在第 10 列,则第 i 行空格的数量为 $10-i$,星号数量为 $2 * i-1$ 。

```

for i in range(1,9):                      # 外层循环
    for j in range(0,10-i):                # 循环输出每行空格
        print (" ", end=" ")

```

```
for j in range(0, 2 * i - 1):           # 循环输出每行星号
    print(" ", end=" ")
print("")                               # 仅起换行作用
```

也可以用如下代码实现：

```
for i in range(1, 9):
    print(" " * (10 - i), "*" * (2 * i - 1))    # 使用重复运算符输出每行空格、星号
```

3.2.5 列表生成式

列表生成式(List Comprehensions)是 Python 内置的一种极其强大的生成 list 列表的表达式。如果要生成一个 list `[1, 2, 3, 4, 5, 6, 7, 8, 9]`，可以用 `range(1, 10)`。

```
>>> L = list(range(1, 10))    # L是[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

如果要生成 `[1 * 1, 2 * 2, 3 * 3, ..., 10 * 10]`，可以使用循环：

```
>>> L = []
>>> for x in range(1, 10):
    L.append(x * x)
>>> L
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

而使用列表生成式，可以用一句代替以上烦琐循环来完成上面的操作：

```
>>> [x * x for x in range(1, 11)]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

列表生成式的书写格式：把要生成的元素 `x * x` 放到前面，后面跟上 for 循环。这样就可以把 list 创建出来。for 循环后面还可以加上 if 判断。例如筛选出偶数的平方：

```
>>> [x * x for x in range(1, 11) if x % 2 == 0]
[4, 16, 36, 64, 100]
```

再如，把一个 list 列表中所有的字符串变成小写形式：

```
>>> L = ['Hello', 'World', 'IBM', 'Apple']
>>> [s.lower() for s in L]
['hello', 'world', 'ibm', 'apple']
```

当然，列表生成式也可以使用两层循环。例如，生成 'ABC' 和 'XYZ' 中字母的全部组合：

```
>>> print( [m + n for m in 'ABC' for n in 'XYZ'] )
['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']
```

再例如生成所有的扑克牌的列表。

```
>>> color = ["草花", "方块", "红桃", "黑桃"]
>>> rank = ["A", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
>>> print ( [m + n for m in color for n in rank])
['草花 A', '草花 1', '草花 2', '草花 3', '草花 4', '草花 5', '草花 6', '草花 7', '草花 8', '草花 9', '草花 10', '草花 J', '草花 Q', '草花 K', '方块 A', '方块 1', '方块 2', '方块 3', '方块 4', '方块 5', '方块 6', '方块 7', '方块 8', '方块 9', '方块 10', '方块 J', '方块 Q', '方块 K', '红桃 A', '红桃 1', '红桃 2', '红桃 3', '红桃 4', '红桃 5', '红桃 6', '红桃 7', '红桃 8', '红桃 9', '红桃 10', '红桃 J', '红桃 Q', '红桃 K', '黑桃 A', '黑桃 1', '黑桃 2', '黑桃 3', '黑桃 4', '黑桃 5', '黑桃 6', '黑桃 7', '黑桃 8', '黑桃 9', '黑桃 10', '黑桃 J', '黑桃 Q', '黑桃 K']
```

for 循环其实可以同时使用两个甚至多个变量,例如字典的 items() 可以同时迭代 key 和 value:

```
>>> d = {'x': 'A', 'y': 'B', 'z': 'C'}          # 字典(dict)
>>> for k, v in d.items():
    print(k, '键 = ', v, endl = ';')
```

输出结果:

```
y 键 = B; x 键 = A; z 键 = C;
```

因此,列表生成式也可以使用两个变量来生成 list:

```
>>> d = {'x': 'A', 'y': 'B', 'z': 'C'}
>>> [ k + '=' + v for k, v in d.items()]
['y=B', 'x=A', 'z=C']
```

3.3 常用算法及应用实例

3.3.1 累加与累乘

累加与累乘是最常见的一类算法,这类算法就是在原有的基础上不断地加上或乘以一个新的数。如求 $1+2+3+\cdots+n$ 、求 n 的阶乘、计算某个数列前 n 项的和,以及计算一个级数的近似值等。

【例 3-9】 求自然对数 e 的近似值,近似公式为:

$$e=1+1/1!+1/2!+1/3!+\cdots+1/n!$$

分析: 这是一个收敛级数,可以通过求其前 n 项和来实现近似计算。通常该类问题会给出一个计算误差,例如,可设定当某项的值小于 10^{-5} 时停止计算。

此题既涉及累加,也包含了累乘。程序如下:

```
i = 1
p = 1
sum_e = 1;
t = 1/p
while t > 0.00001:
    p = p * i;           // 计算 i 的阶乘
    t = 1/ p;
    sum_e = sum_e + t;
    i = i + 1;           // 为计算下一项做准备
print("自然对数 e 的近似值", sum_e);
```

运行结果:

自然对数 e 的近似值 2.7182815255731922

3.3.2 求最大数和最小数

求数据中的最大数和最小数的算法是类似的,可采用“打擂”算法。以求最大数为例,可先用其中第一个数作为最大数,再用其与其他数逐个比较,并将找到的较大的数替换为最大数。

【例 3-10】 求区间[100, 200]内 10 个随机整数中的最大数。

分析: 本题随机产生整数,所以引入 random 模块随机数函数,其中 random.randrange() 可以从指定范围内获取一个随机数。例如,random.randrange(6),从 0~5 中随机挑选一个整数,不包括数字 6; random.randrange(2,6),从 2~5 中随机挑选一个整数,不包括数字 6。

```
import random
x = random.randrange(100,201)           # 产生一个[100, 200]的随机数 x
maxn = x                                # 设定最大数
print(x,end = " ")
for i in range(2, 11):
    x = random.randrange(100,201)       # 再产生一个[100, 200]的随机数 x
    print(x,end = " ")
    if x > maxn :
        maxn = x;                       # 若新产生的随机数大于最大数,则进行替换
print("最大数: ",maxn)
```

运行结果:

185 173 112 159 116 168 111 107 190 188 最大数: 190

当然,在 Python 中求最大数有相应的函数 max(序列)。例如:

```
print ("最大数: ",max([185,173, 112, 159, 116, 168, 111, 107, 190, 188])) # 求序列最大数
```

运行结果:

```
最大数: 190
```

所以上例可以修改如下:

```
import random
a = [] # 列表
for i in range(1, 11):
```

```
    x = random.randrange(100,201) # 产生一个[100, 200]的随机数 x
    print(x,end = " ")
    a.append(x)
print ("最大数: ",max(a))
```

3.3.3 枚举法

枚举法又称为穷举法,此算法将所有可能出现的情况一一进行测试,从中找出符合条件的所有结果。如计算“百钱买百鸡”问题,又如列出满足 $x * y = 100$ 的所有组合等。

【例 3-11】 公鸡每只 5 元,母鸡每只 3 元,小鸡 3 只 1 元,现要求用 100 元钱买 100 只鸡,问公鸡、母鸡和小鸡各买几只?

分析: 设买公鸡 x 只,母鸡 y 只,小鸡 z 只。根据题意可列出以下方程组:

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + z/3 = 100 \end{cases}$$

由于 2 个方程式中有 3 个未知数,属于无法直接求解的不定方程,故可采用“枚举法”进行试根,即逐一测试各种可能的 x 、 y 、 z 组合,并输出符合条件者。

```
for x in range(0, 100):
    for y in range(0, 100):
        z = 100 - x - y
        if z >= 0 and 5 * x + 3 * y + z/3 == 100 :
            print ('公鸡 %d 只,母鸡 %d 只,小鸡 %d 只'%(x, y, z))
```

运行结果:

```
公鸡 0 只,母鸡 25 只,小鸡 75 只
公鸡 4 只,母鸡 18 只,小鸡 78 只
公鸡 8 只,母鸡 11 只,小鸡 81 只
公鸡 12 只,母鸡 4 只,小鸡 84 只
```

【例 3-12】 输出“水仙花数”。所谓水仙花数是指一个 3 位的十进制数,其各位数字的立方和等于该数本身。例如,153 是水仙花数,因为 $153=1^3+5^3+3^3$ 。

```
for i in range(100,1000):
    ge = i % 10
    shi = i // 10 % 10
    bai = i // 100
    if ge**3 + shi**3 + bai**3 == i:
        print(i,end=" ")
```

运行结果:

```
153 370 371 407
```

【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 3 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)
```

3.3.4 递推与迭代

1. 递推

利用递推算法或迭代算法,可以将一个复杂的问题转换为一个简单的过程重复执行。这两种算法的共同特点是,通过前一项的计算结果推出后一项。不同点是,递推算法不存在变量的自我更迭,而迭代算法则在每次循环中用变量的新值取代其原值。

【例 3-14】 输出斐波那契(Fibonacci)数列的前 20 项。该数列的第 1 项和第 2 项为 1,从第 3 项开始,每一项均为其前面 2 项之和,即 1,1,2,3,5,8,...

分析: 设数列中相邻的 3 项分别为变量 f1、f2 和 f3,则有如下递推算法。

- (1) f1 和 f2 的初值为 1。
- (2) 每次执行循环,用 f1 和 f2 产生后项,即 $f3 = f1 + f2$ 。
- (3) 通过递推产生新的 f1 和 f2,即 $f1 = f2, f2 = f3$ 。
- (4) 如果未达到规定的循环次数,则返回步骤(2); 否则停止计算。

```
f1 = 1
f2 = 1
print("1:", f1)
print("2:", f2)
```

```

for i in range(3, 21):
    f3 = f1 + f2      # 递推公式
    print(i, ":", f3)
    f1 = f2
    f2 = f3

```

说明：解决递推问题必须具备两个条件，即初始条件和递推公式。本题的初始条件为 $f_1=1$ 和 $f_2=1$ ，递推公式为 $f_3=f_1+f_2, f_1=f_2, f_2=f_3$ 。

【例 3-15】 有一分数序列：2/1, 3/2, 5/3, 8/5, 13/8, 21/13, …，求出这个数列的前 20 项之和。

分析：根据分子与分母的变化规律，可知后项分母为前项分子，后项分子为前项分子分母之和。

```

number = 20
a = 2
b = 1
s = 0
for n in range(1, number + 1):
    s = s + a/b
    # 以下三句是程序的关键
    t = a
    a = a + b
    b = t
print(s)

```

2. 迭代

迭代法也称辗转法，是一种不断用变量的旧值递推新值的过程。迭代法是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组指令（或一定步骤）进行重复执行，在每次执行这组指令（或这些步骤）时，都从变量的原值推出它的一个新值。

【例 3-16】 迭代法求 a 的平方根。求平方根的公式为： $x_{n+1} = (x_n + a/x_n) / 2$ ，求出的平方根精度是前后项差的绝对值小于 10^{-5} 。

分析：迭代法求 a 的平方根的算法如下：

(1) 设定一个 x 的初值 x_0 （在如下程序中取 $x_0=a/2$ ）。

(2) 用求平方根的公式 $x_1 = (x_0 + a/x_0) / 2$ 求出 x 的下一个值 x_1 ；求出的 x_1 与真正的平方根相比，误差很大。

(3) 判断 $x_1 - x_0$ 的绝对值是否满足大于 10^{-5} ，如果满足，则将 x_1 作为 x_0 ，重新求出新 x_1 ，如此继续下去，直到前后两次求出的 x 值（ x_1 和 x_0 ）的差的绝对值满足小于 10^{-5} 。

```

a = int(input("Input a positive number:"))
x0 = a / 2;
x1 = (x0 + a / x0)
while abs(x1 - x0) > 0.00001 :

```

输入被开方数
任取的初值
x0, x1 分别代表前一项和后一项
abs(x)函数用来求参数 x 的绝对值

```
x0 = x1
x1 = (x0 + a / x0) / 2
print("The square root is: ", x0)
```

运行结果:

```
Input a positive number: 2 ✓
The square root is: 1.4142137800471977
```

3.4 游戏初步——猜单词游戏



视频讲解

【案例 3-1】 游戏初步——猜单词游戏。计算机随机产生一个单词,打乱字母顺序,供玩家去猜。

分析: 游戏中需要随机产生单词以及随机数字,所以引入 random 模块随机数函数,其中 random.choice() 可以从序列中随机选取元素。例如:

```
WORDS = ("python", "jumble", "easy", "difficult", "answer", "continue",
         "phone", "position", "pose", "game")
# 从序列中随机挑出一个单词
word = random.choice(WORDS)
```

word 就是从单词序列中随机挑出的一个单词。

从游戏中随机挑出一个单词 word 后,如何把单词 word 的字母顺序打乱,方法是随机从单词字符串中选择一个位置 position,把 position 位置上的那个字母加入乱序后单词 jumble,同时将原单词 word 中 position 位置上的那个字母删去(通过连接 position 位置前字符串和其后字符串实现)。通过多次循环就可以产生新的乱序后单词 jumble。

```
while word: # word 不是空串循环
    # 根据 word 长度,产生 word 的随机位置
    position = random.randrange(len(word))
    # 将 position 位置上的字母组合到乱序后单词
    jumble += word[position]
    # 通过切片,将 position 位置上的字母从原单词中删除
    word = word[:position] + word[(position + 1):]
print("乱序后单词:", jumble)
```

猜单词游戏程序代码如下:

```
# Word Jumble 猜单词游戏
import random
# 创建单词序列
WORDS = ("python", "jumble", "easy", "difficult", "answer", "continue",
        "phone", "position", "position", "game")
```

```

# start the game
print(
    """
        欢迎参加猜单词游戏
        把字母组合成一个正确的单词.
    """
)
iscontinue = "y"
while iscontinue == "y" or iscontinue == "Y":
    # 从序列中随机挑出一个单词
    word = random.choice(WORDS)
    # 一个用于判断玩家是否猜对的变量
    correct = word
    # 创建乱序后单词
    jumble = ""
    while word: # word 不是空串时循环
        # 根据 word 长度, 产生 word 的随机位置
        position = random.randrange(len(word))
        # 将 position 位置字母组合到乱序后单词
        jumble += word[position]
        # 通过切片, 将 position 位置字母从原单词中删除
        word = word[:position] + word[(position + 1):]
    print("乱序后单词:", jumble)
    guess = input("\n 请你猜: ")
    while guess != correct and guess != "":
        print("对不起, 不正确.")
        guess = input("继续猜: ")
    if guess == correct:
        print("真棒, 你猜对了!\n")
    iscontinue = input("\n\n 是否继续 (Y/N): ")

```

运行结果:

```

        欢迎参加猜单词游戏
        把字母组合成一个正确的单词.
乱序后单词: yaes
请你猜: easy
真棒, 你猜对了!
是否继续 (Y/N): y
乱序后单词: diufctlfi
请你猜: difficult
对不起, 不正确.
继续猜: difficult
真棒, 你猜对了!
是否继续 (Y/N): n
>>>

```

3.5 习 题

1. 输入一个整数 n , 判断其能否同时被 5 和 7 整除, 如能则输出“ xx 能同时被 5 和 7 整除”, 否则输出“ xx 不能同时被 5 和 7 整除”。要求 xx 为输入的具体数据。

2. 输入一个百分制的成绩, 经判断后输出该成绩的对应等级。其中, 90 分以上为 A, 80~89 分为 B, 70~79 分为 C, 60~69 分为 D, 60 分以下为 E。

3. 某百货公司为了促销, 采用购物打折的办法。消费 1000 元以上者, 按九五折优惠; 消费 2000 元以上者, 按九折优惠; 消费 3000 元以上者, 按八五折优惠; 消费 5000 元以上者, 按八折优惠。编写程序, 输入购物款数, 计算并输出优惠价。

4. 编写一个求整数 n 的阶乘($n!$)的程序。

5. 利用循环创建一个包含 10 个奇数的列表, 并计算该列表的和与平均值。

6. 编写程序, 求 $1!+3!+5!+7!+9!$ 。

7. 编写程序, 计算下列公式中 s 的值(n 是运行程序时输入的一个正整数)。

$$s = 1 + (1 + 2) + (1 + 2 + 3) + \cdots + (1 + 2 + 3 + \cdots + n)$$

$$s = 12 + 22 + 32 + \cdots + (10 \times n + 2)$$

$$s = 1 \times 2 - 2 \times 3 + 3 \times 4 - 4 \times 5 + \cdots + (-1)^{(n-1)} \times n \times (n+1)$$

8. 百马百瓦问题: 有 100 匹马驮 100 块瓦, 大马驮 3 块, 小马驮 2 块, 两个马驹驮 1 块。问: 大马、小马和马驹各有多少匹。

9. 有一个数列, 其前三项分别为 1、2、3, 从第四项开始, 每项均为其相邻的前三项之和的 $1/2$, 问: 该数列从第几项开始, 其数值超过 1200。

10. 找出 1~100 的全部同构数。同构数是这样一种数: 它出现在它的平方数的右端。例如, 5 的平方是 25, 5 是 25 中右端的数, 5 就是同构数, 25 也是一个同构数, 它的平方是 625。

11. 猴子吃桃问题: 猴子第一天摘下若干个桃子, 当即吃了一半, 还不过瘾, 又多吃了一个, 第二天早上将剩下的桃子吃掉一半, 又多吃了一个。以后每天早上都吃前一天剩下的一半再加一个。到第 10 天早上想再吃时, 发现只剩下一个桃子。求第一天共摘了多少个桃子。

12. 输入一个字符串, 然后依次显示该字符串的每个字符以及该字符的 ASCII 码。

13. 开发猜数字小游戏。计算机随机生成 100 以内数字, 玩家去猜, 如果猜的数字过大或过小都会给出提示, 直到猜中该数, 显示“恭喜! 你猜对了”, 同时要统计玩家猜的次数。

14. 已知 $abc+cba=1333$, 其中 a, b, c 均为 1 位数, 编写程序求出 a, b, c 分别代表什么数字。