第4章
条件、循环和其他语句

CS, ZJU

2018年12月

Overview

- 条件语句
- 循环语句
- 搜索和排序
- 异常处理
- 嵌套循环和二维列表

4.1 条件语句

- 条件语句的三种格式：

基本的条件语句	有分支的条件语句	连缀的if-elif-else

if 条件：
语句块1
	if 条件：
	语句块1
	else:
	语句块2
		if 条件1：
		语句块1
		elif 条件2:
		语句块2
		…
		elif 条件n:
		语句块n
		else:
		语句块 n+1

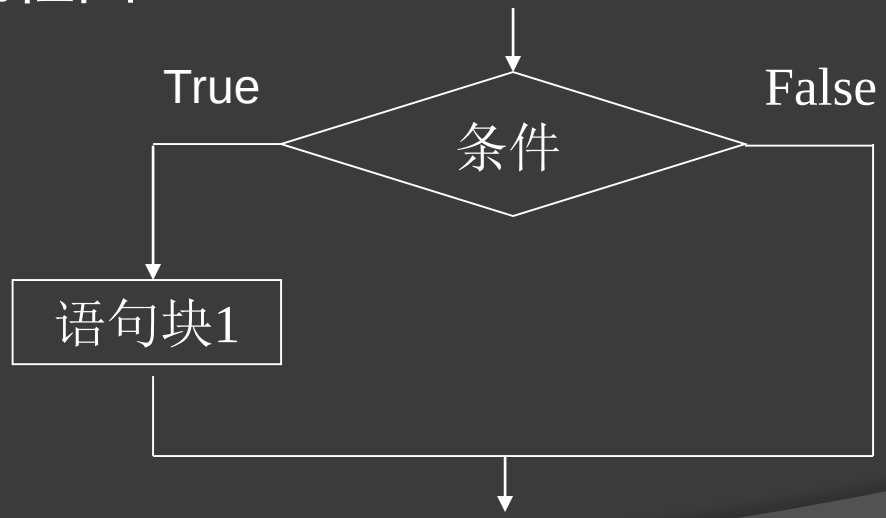
基本的条件语句

- 格式：

  if 条件：

  　　语句块1　　#分支语句块,书写时必须缩进。

- if没有else的流程图



基本的条件语句（续）

- 一个基本的条件语句由一个关键字if开头，跟上一个表示条件的逻辑表达式，然后是一个冒号:。
- if A:B　　A and B
- 从下一行开始，所有缩进了的语句就是当条件成立（逻辑表达式计算的结果为True）的时候要执行的语句。
- 如果条件不成立，就跳过这些语句不执行，而继续下面的其他语句。

```
x = int(input())
y=z=0
if x>20:
    y = 100        # 书写缩进，当x>20时执行
    z = 200        # 书写缩进，当x>20时执行
print(y+z)         # if语句后续的语句
```

基本的条件语句（续2）

例4-3 根据输入的金额决定是否售票

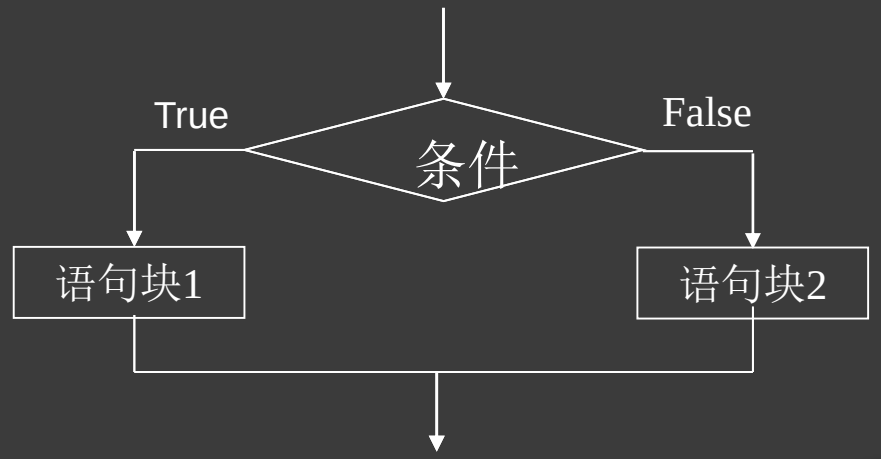
```
#   读入投币金额
amount = int(input('请投币：'))
if amount >= 10:
    #       打印车票
    print('********************')
    print('*Python城际铁路专线*')
    print('*   票价：10元     *')
    print('********************')
    #       计算并打印找零
    print('找零：{}'.format(amount-10))
```

二分支的条件语句

- 格式：

```
if 条件：
        语句块1    #分支语句块,条件成立时执行。
else：
        语句块2    #分支语句块,条件不成立时执行。
```

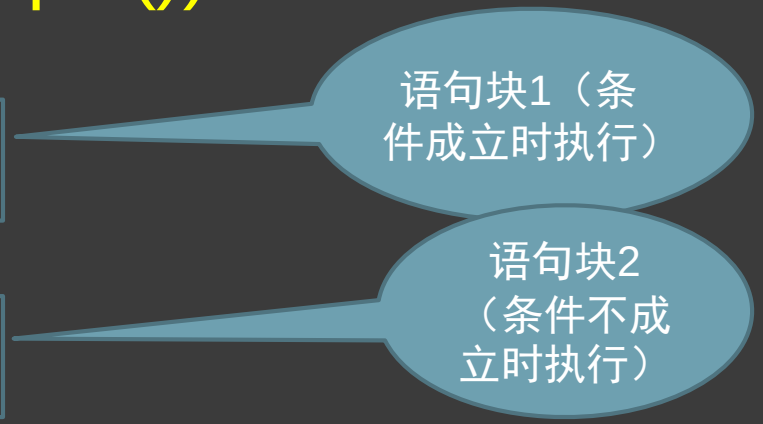
- if-else的流程图



二分支的条件语句（续）

- 例4-7 比较2个数的大小

```
x,y = map(int, input().split())
if x > y:
    max = x
else:
    max = y
print(max)
```



嵌套的条件语句

- 分支语句（块）中包含另一个if语句，这种情况称为条件语句的嵌套
- if code == 'R':

```
if count <20:
    print('一切正常')
else:
    print('继续等待')
```

书写缩进

嵌套的条件语句（续）

- 在嵌套if语句里，最重要的问题是else的匹配。else总是根据它自己所处的缩进和同列的最近的那个if匹配。
- if code == ‘R’:
    if count <20:
        print(‘一切正常’)

```
else:
    print(‘继续等待’)        #当code不为’R’时执行
```

求三个数的最大值

```
x,y,z=input().split()
x,y,z=int(x),int(y),int(z)
if x>y:
    if x>z:
        print(x)
    else:
        print(z)
else:
    if y>z:
        print(y)
    else:
        print(z)
```

连缀的if-elif-else

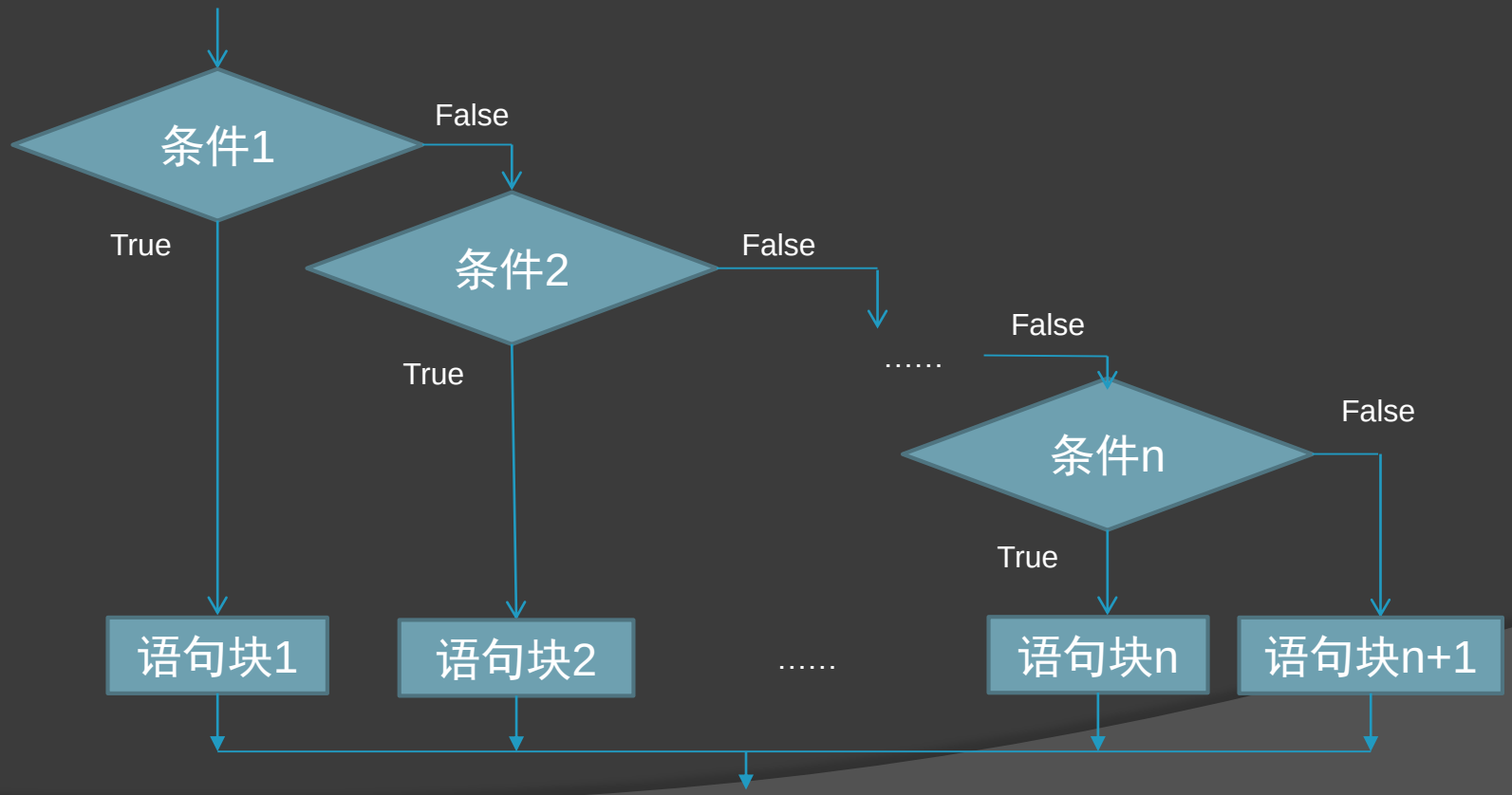
- 使用if-elif-else语句可方便地实现程序多分支结构。
- 格式：

```
if  条件1:
    语句块1                #分支语句块,书写时必须缩进。
elif  条件2:
    语句块2                #分支语句块,书写时必须缩进。
…
elif  条件n:
    语句块n                #分支语句块,书写时必须缩进。
else:
    语句块  n+1            #分支语句块,书写时必须缩进。
```

其中的if，elif和else必须在同一列对齐。

连缀的if-elif-else（续）

if-elif-else的流程图



连缀的if-elif-else（续）

- 分段函数在数学中也是常见的，比如下面的这个函数：

  f(x) = -1; x<0
  f(x) = 0; x=0
  f(x) = 2x; x>0

  例4-10 分段函数

```
x = int(input())
f = 0
if x < 0:
    f = -1
elif x == 0:
    f = 0
else:
    f = 2 * x
print(f)
```

- 2次条件判断，实现3个分支。

条件表达式

- 条件表达式类似if-else语句，是用来直接得到值。
- 条件表达式是三元的，需要三个值：
  - 条件满足时的值
  - 条件
  - 条件不满足时的值
- 如：

  y = 10 if x > 20 else 30

  当x大于20时，条件表达式值为10，否则为30

4.2 while循环

- 循环语句可以重复执行部分语句（循环语句/循环体），while语句是一种循环语句，根据一个逻辑条件（循环条件），在条件成立时执行循环语句，不成立时结束循环。
- 格式

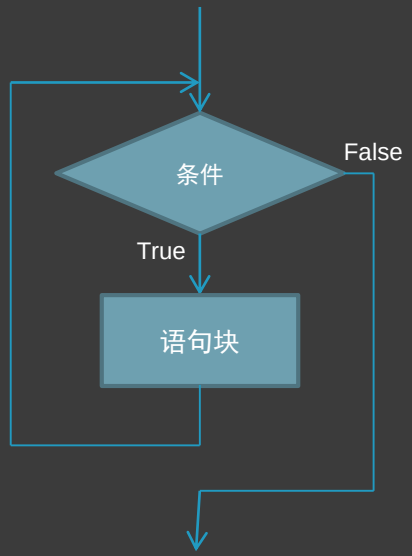
无else子句	有else子句
while 条件： 　语句块1	while 条件： 　语句块1 else: 　语句块2

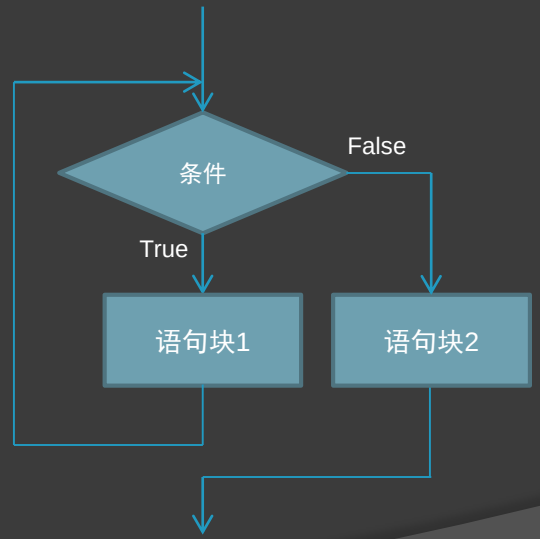
while循环

- While循环的流程图

```
while 条件：
    语句块1    #书写缩进
```

```
while 条件：
    语句块1    #书写缩进
else:
    语句块2    #书写缩进
```





while循环（续）

- 要点
  - 执行while语句的流程：
    1. 判断条件是否成立
    2. 如果条件成立，则执行语句块1
    3. 回到第1步
    4. 如果条件不成立，若有else子句，执行语句块2
    5. 结束
  - 循环语句书写必须缩进
  - 在循环体内部，应该有改变循环条件的语句，以控制循环的次数，避免产生无限循环（死循环）。

while循环（续2）

- 例4-2 计算$\log_2 x$

```
x = int(input())
count = 0
while x >1:
    x //= 2
    count += 1
print(count)
```

缩进

输入：32

输出：5

输入：31

输出：4

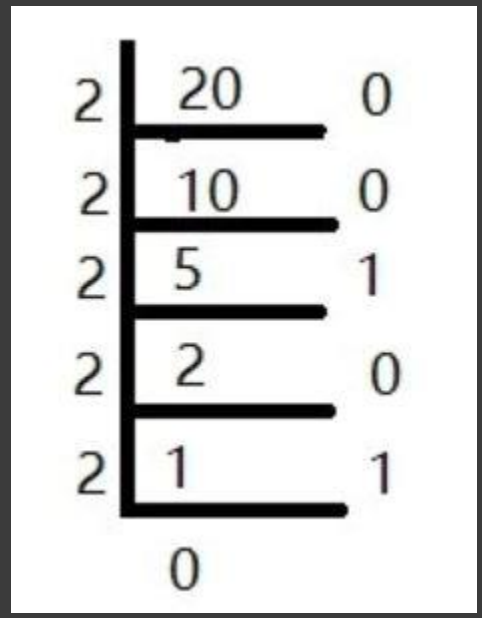
步骤	x	count	说明
1	32	0	进入循环之前
2	16	1	循环的第一轮
3	8	2	
4	4	3	
5	2	4	
6	1	5	

步骤	x	count	说明
1	31	0	进入循环之前
2	15	1	循环的第一轮，31/2->15
3	7	2	
4	3	3	
5	1	4	

练习------二十进制转换

```
n=int(input())
lst=[]
while n>0:
    lst.insert(0,n%2)
    n=n//2
print("".join(map(str,lst)))
```

n	n%2



while循环（续3）

例4-14 求最大公约数

求两个数的最大公约数可以用辗转相除法。

- 算法：
  1. 先用小的一个数除大的一个数，得第一个余数；
  2. 再用第一个余数除小的一个数，得第二个余数；
  3. 又用第二个余数除第一个余数，得第三个余数；
  4. 这样逐次用后一个余数去除前一个余数，直到余数是0为止。那么，最后一个除数就是所求的最大公约数（如果最后的除数是1，那么原来的两个数是互质数）。

while循环（续4）

- 更加形式化的算法描述：
  1. 计算两个数a和b的余数r；
  2. 如果余数r不为0，则以b和r作为新的a和b，回到1重复计算；
  3. 否则b就是余数
- 程序（a>=b）：

```
a, b = map(int, input("请输入两个整数：").split())
r = a % b
while r > 0:
    a, b = b, r
    r = a % b
print("最大公约数是{}".format(b))
```

输入：60 21

输出：最大公约数是3

步骤	a	b	r	说明
1	60	21	18	进入循环前
2	21	18	3	循环的第1轮
2	18	3	0	循环的第2轮

循环内的控制

- 跳出循环break

  break语句的作用是跳出所在的循环。

  例4-15 猜数游戏

```
import random
number = random.randint(0,100)
count = 0
while True:    # 循环条件是逻辑常量True，意味着无限循环
    a = int(input('输入你猜的数:'))
    count += 1
    if a == number:
        break   #  跳出当前循环，执行while后面的语句。
    elif a>number:
        print('你猜的大了')
    else:
        print('你猜的小了')
print('猜中了！ 你用了{}次！ '.format(count))
```

循环内的控制（续）

- 跳过一轮循环continue

  continue语句作用是跳过本次循环，进入到下一次循环。

  例4-16 计算偶数的平均数

  要求：输入一系列的整数，最后输入-1表示输入结束，然后程序计算出这些数字中的偶数的平均数，输出输入的数字中的偶数的个数和偶数的平均数。

循环内的控制（续2）

```
sum = 0
count = 0
while True:
    number = int(input())
    if number == -1:
        break
    if number % 2 == 1:
        continue    # 如果是奇数的话，跳过后面的循环语句，
                    # 进入下一次循环。
    sum += number
    count += 1
average = sum / count
print(average)
```

循环内的控制（续3）

- 例4-17 输入一个大于等于2的正整数，判断是否为素数。

```
num=int(input())
a=num-1
while a>1:
    if num % a == 0:
        print("不是素数")
        break    # 跳出当前循环，包括else子句。
    a=a-1
else:
    print("是素数")
```

循环内的控制（续4）

- 说明
  - 程序中的循环控制变量a从num-1递减到1，程序在每次循环判断a（从num-1到2）是否是num的因数。
  - 如是，则打印”不是素数”,然后break语句跳出while语句，当然也跳过了else子句。
  - 如果循环过程中“num % a”始终不为0，即num不能被从2到num-1中的任何一个数整除，说明num是素数，在循环结束后执行else子句, 打印”是素数”。

4.3 for循环

- 当有一个序列，需要按照其顺序遍历其中的每一个单元的时候，就可以用for循环。

表 4-3　for语句语法格式

无else子句	带else子句
for 循环变量 in 序列： 　　语句块1	for 循环变量 in 序列： 　　语句块1 else: 　　语句块 2

- for i in [1,2,3,4]:
- 　　print(i,end=“ “)
- 结果是：1 2 3 4

for...in循环

for循环又被叫做for ... in循环，它的一般形式是：

```
for <变量> in <序列>:
    缩进代码块
非缩进代码块
```

在循环的每一轮，<变量>会依次取序列中的一个值。对序列中的最后一个值执行完缩进代码块后，程序继续执行非缩进代码块。

两种循环模式

例：针对下列包含十二个月的英文缩写的列表month

```
month = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
```

要遍历这个列表，输出每个月的缩写。

1.计数器循环：

```
for i in range(len(month)):
    print(month[i])
```

2.迭代循环：

```
for name in month:
    print(name)
```

判断素数（1）

输入一个大于等于2的正整数，判断是否为素数。素数是只能被1和自己整除的数，因数的范围是从1到自己。

直接用素数定义求解

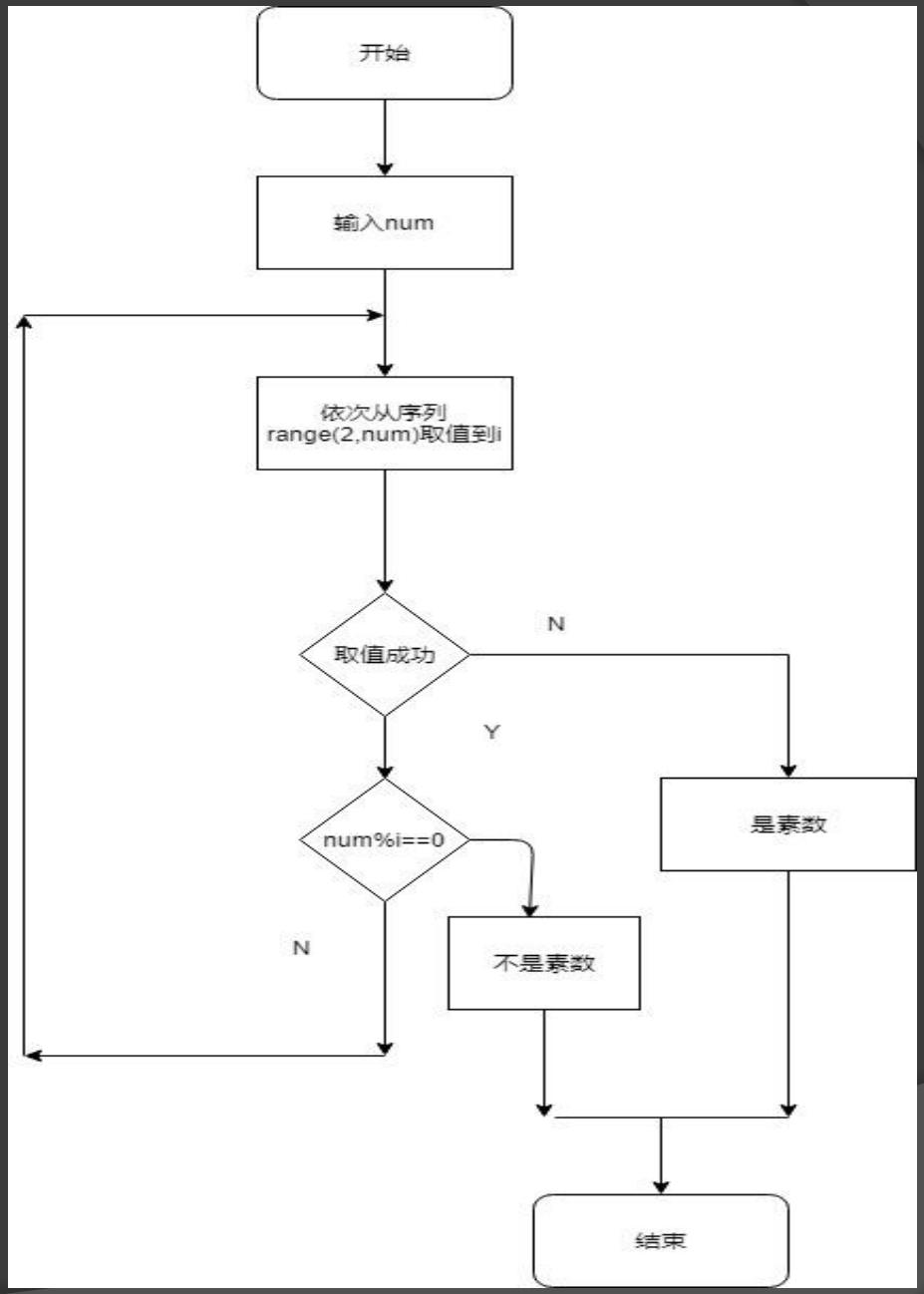
```
num=int(input())
lst=[ factor for factor in range(1,num+1)
      if num % factor==0]
if lst==[1,num]:
    print("是素数")
else:
    print("不是素数")
```

判断素数（2）

例4-18 输入一个大于等于2的正整数，判断是否为素数(for 语句实现)。

```
num=int(input())
for i in range(2,num):
# i从2到num-1
    if num % i == 0:
        print("不是素数")
        break
else:
        print("是素数")
```

如何优化？



判断素数优化程序

- 如 m=a*b
- 则必有：
- a<=sqrt(m)
- 或：
- b<=sqrt(m)

例4-18 输入一个大于等于2的正整数，判断是否为素数。

```
import  math
num=int(input())
for i in range(2,int(math.sqrt(num))+1):
# i从2到sqrt(num)
    if num % i == 0:
        print("不是素数")
        break
else:
        print("是素数")
```

嵌套循环

- 嵌套循环是由一个外循环和一个或多个内层循环构成。每次重复外层循环时，内层循环都要重新进入并要重新循环一编。
- for i in range(3):　　#外层循环
- 　　for j in range(10):　　#内层循环
- 　　print(str(i)+str(j),end=' ')
- 　　print()　　#换行
- 运行结果：
- 00 01 02 03 04 05 06 07 08 09
- 10 11 12 13 14 15 16 17 18 19
- 20 21 22 23 24 25 26 27 28 29
- Print语句执行次数 ： 30

嵌套循环也可for和while混合实现

```
for i in range(1,4):     #外层循环
    j=0                  #内层循环变量初始化
    while j<i:           #内层循环
        print(j,end=' ')
        j+=1
    print()
```

运行结果：

```
0
0 1
0 1 2
```

例4-19 求[m,n]之间的素数和(C语言思路）

```
sum = 0
m,n = map(int, input().split())
if m==1:        # 1不是素数
    m = 2
prime = []    # 记录已知素数的列表
for x in range(2, n+1):
    isprime = True
    for k in prime:
            if x%k == 0:
                isprime = False
                break
    if isprime:
        if x >=m:
            sum += x
        prime.append(x)        # 加入已知素数的列表，
                               # 用于下一次计算
print(sum)
```

- 程序说明：
  - 用一个循环来遍历[m,n]范围内所有的整数，这可以用for x in range(m, n+1)来实现。
  - 对于上面的循环中每一个数x，如果所有小于x的素数都不能整除它，它就是素数。
  - 构造一个已知素数的列表，每次发现一个新的素数，就将它加入这个列表；而需要判断x是否是素数时，就用这个列表中的每一个素数来测试它，用另外一个循环for k in prime: 。
  - 循环体内包含循环语句称为循环嵌套/多重循环。

例4-19 求[m,n]之间的素数和

```
sum = 0
m,n = map(int, input().split())
if m==1:        # 1不是素数
    m = 2
prime = []    # 记录已知素数的列表

for x in range(2, n+1):
    for k in prime:
        if x%k == 0:
            break
    else:
        if x>=m:
            sum += x
        prime.append(x)        # 加入已知素数的列表，
                               # 用于下一次计算
print(sum)
```

for语句和while语句的选择

- 求大于2950的37的第一个倍数

- for multi in range(37,???,37):
- for语句无法确定范围

- bound=2950
- multi=37
- while multi<=bound:
- 	multi+=37
- print(multi)

- 输出：2960

累加误差带来的错误

- #加 0.01,0.02,0.03，...,0.99,1 to #sum
- sum=0

- i=0.01
- while i<=1.0:
- 　　sum+=i
- 　　i=i+0.01
- print("和是：{:.2f}".format(sum))
- 最后一项没有加

- sum=0
- i=0.01
- for count in range(100):
- 　　sum+=i
- 　　i=i+0.01
-
- print("和是：{:.2f}".format(sum))

编程显示如下的图案

- 该题可用嵌套循环解决。外层循环对应一行，内层循环画每层的‘*’号。对于第i行，前面先输出10-i个空格，再输出2*i-1个‘*’号。每行的结果放在字符串s中，特别注意每次进入内层循环s都要初始化：s=""。

程序代码

```
for i in range(1,11):
    s=""
    for j in range(0,10-i):
        s +=" "
    for j in range(0,2*i-1):
        s +="*"
    print(s)
```

练习1---打印图形

- *
- **
- ***
- ****
- *****

```
for i in range(1,6):
    print("*"*i)
```

练习2---打印图形

```
   *
  ***
 *****
  ***
   *
```

```
for i in range(1,4):
    print(' '*(3-i)+'*'*(2*i-1))
for i in range(2,0,-1):
    print(' '*(3-i)+'*'*(2*i-1))
```

求m到n之间的完数

```
m,n=map(int,input().split())
count=0

for i in range(m,n+1):
  lst=[k for k in range(1,i) if i%k==0]
  factorsum=sum(lst)
  if i==factorsum:
    count+=1
    print(str(i)+"=“+”+”.join(map(str,lst)))
if count==0:
  print("None")
```

求m到n之间的完数(优化）

```
import math
m,n=map(int,input().split())
count=0
for i in range(m,n+1):
  lst1=[1]
  for k in range(2,int(math.sqrt(i))+1):
     if i%k==0:
        lst1.append(k)
        if i//k not in lst1:
           lst1.append(i//k)
  lst1.sort()
  factorsum=sum(lst1)
  if i==factorsum:
    count+=1
    print(str(i)+" = "+" + ".join(map(str, lst1)))

if count==0:
   print("None")
```

猴子报数选大王

- 数到3出列，最后一个是几号？

```
N=int(input())     #N是猴子的总数
ls=[i for i in range(1,N+1)]
print(ls)

ptr=1          #从1开始报数，报数猴子的下标-1
while len(ls)>1:
  ptr=ptr+2
  ptr=(ptr-1)%len(ls)+1
  print(ls[ptr-1],end=" ")
  del ls[ptr-1]
print()
print(ls[0])
```

用二维列表表示二维表格

4	**71**	**2**	**5**
58	114	94	2
67	3	6	45

二维列表是一个列表，这个列表的元素本身又是列表。lst是一个二维列表，第一个元素代表第一行，第二个元素代表第二行，第三个元素代表第三行。

- >>>lst=[[4,71,2,5],[58,114,94,2],[67,3,6,45]]
- >>>lst[1]　　#取第二行
-
- [58, 114, 94, 2]
- >>>lst[1][2]　　#取第二行的第三个元素
- 94
- >>> lst[2][1:3]　　#取第三行的第二，三个元素
- [3, 6]

用下标按行显示二维列表：

- lst=[[4,71],[58,114,94,2],[67,6,45]]

- for row in range(len(lst)):
- 	for col in range(len(lst[row])):
- 		print(lst[row][col],end=' ')
- 	print()
-
- 运行结果：
-
- 4 71
- 58 114 94 2
- 67 6 45

直接取列表的元素

- row代表列表lst的某个元素，本身又是一个列表。col代表row列表中的某一个元素。

- lst=[[4,71],[58,114,94,2],[67,6,45]]
- for row in lst:
- 	for col in row:
- 		print(col,end=' ')
- 	print()
-
- 运行结果：
-
- 4 71
- 58 114 94 2
- 67 6 45

嵌套列表求和

```
a = [1, 2, 3, 4,
     [5, 6],
     [7, 8, 9],
     67]
s = 0
for row in a:
    if type(row)==list:
        for elem in row:
            s += elem
    else:
        s+=row
print(s)
```

用嵌套循环产生列表

- >>> [(i,j) for i in range(3) for j in range(3)]
- [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]

- 请注意下面语句i是外层循环，j是内层循环

- >>> [[i+j for i in range(3)] for j in range(3)]
- [[0, 1, 2], [1, 2, 3], [2, 3, 4]]

列表的”*”运算可用于列表的初始化

- >>> [0]*3　　# 0是不可变对象
- [0, 0, 0]
-
- >>> [[0]]*3　　#[0]是可变对象
- [[0], [0], [0]]
-
- >>> [[0]*3]*3　　#[0]是可变对象
- [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

矩阵的列表表示

矩阵是高等代数中的常见工具，也常见于统计分析等应用数学学科中。由 m × n 个数a_{ij}排成的m行n列的数表称为m行n列的矩阵，简称m × n矩阵。记作：

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \cdots & \cdots & & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

这可以用二维列表表示：

lst_a=[[a_{11}, a_{12}, ⋯a_{1n}], [a_{21}, a_{22}, ⋯a_{2n}]⋯[a_{m1}, a_{m2}, ⋯a_{mn}]]

输入一个3行2列的矩阵，求每行的和

```
mat=[]
#输入数据，产生矩阵
for i in range(3):
    row=[]
    for j in range(2):
        row.append(int(input()))
    mat.append(row)
#输出列表mat
print(mat)
#求每行和
for i in range(3):
    s=0
    for j in range(2):
        s+=mat[i][j]
    print(s)
```

程序输入：

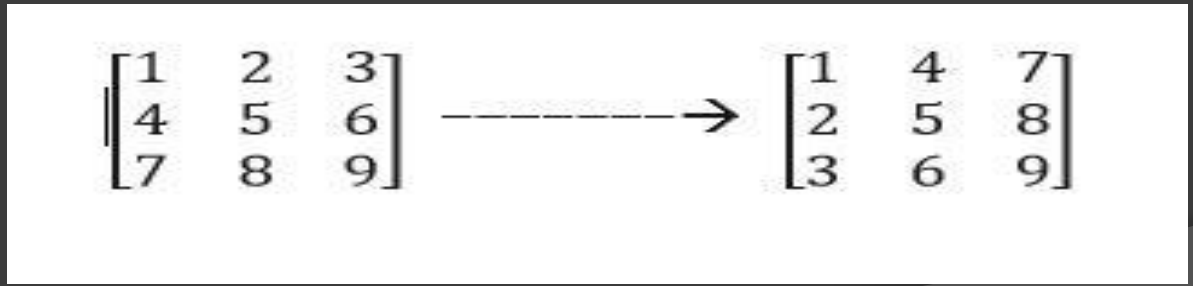
```
1
7
6
8
34
64
```

程序输出

```
[[1, 7], [6, 8], [34, 64]]
8
14
98
```

矩阵转置

- 产生一个如左下边的3行3列矩阵，变成如右下边的3行3列矩阵。这种变换称为矩阵的转置，即行列互换。
- 这个矩阵的元素满足以下公式：
- a[i][j]=i*n+j+1
- 0<=i<n,0<=j<n,n是矩阵的行数

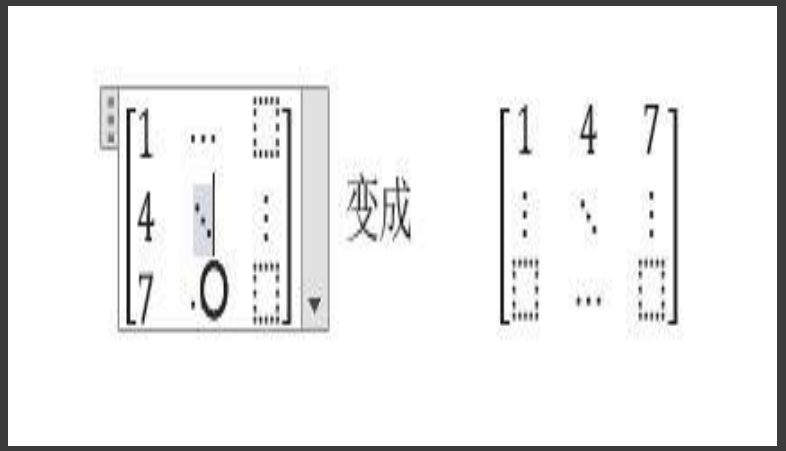


用公式产生矩阵

- >>> mat=[[i*3+j+1 for j in range(3)] for i in range(3)]
- >>> mat
- [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

解题思路

行列互换就是a[i][j]与a[j][i]互换。以第一列为例



- 如何取第一列？row是mat行，同时也是一个列表，row[0]就是某行的第一列。下面的表达式就可取矩阵的第一列。
- [row[0] for row in mat]
- 第二，三列则是：
- [row[1] for row in mat],
- [row[2] for row in mat]
- 转置矩阵：
- [[row[0] for row in mat] ,
- [row[1] for row in mat],
- [row[2] for row in mat]]

矩阵常用术语与列表下标的关系

- （i是行下标，j是列下标，N是行数）

主对角线	i=j	左上角与右下角的连线
付对角线	i+j=N-1	左下角与右上角的连线
上三角	i<=j	主对角线以上部分
下三角	i>=j	主对角线以下部分

程序代码

- mat=[[i*3+j+1 for j in range(3)] for i in range(3)]
- print(mat)
- mattrans=[[row[col] for row in mat] for col in range(3)]
- print(mattrans)
- 运行结果：
- [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

4.4 搜索和排序

- 在一个数据集中搜索某个数据，对已有的数据集按照某个规则进行排序，是数据处理中最常见的两种任务。

线性搜索

- 所谓线性搜索，就是依次检查数据集中的每一个数据，看是否与要搜索的数据相同，如果相同，就得到了结果。

例4-20 线性搜索1

```
a = [2,3,5,7,11,13,17,23,29,31,37]
x = int(input())
found = False
for k in a:
    if k == x:
        found = True
        break
print(found)
```

线性搜索（续）

例4-21 线性搜索2

```
a = [2,3,5,7,11,13,17,23,29,31,37]
x = int(input())
found = -1
for i in range(len(a)):
    if a[i] == x:
            found = i
            break
print(found)
```

搜索最大值、最小值

- 还有一种搜索需求，是在一个数据集中寻找最大值或最小值。如果不需要给出最值所在的位置，可以直接遍历列表的每个单元；而如果需要给出位置，就需要用下标来做搜索。

搜索最大值、最小值（续）

例4-22 搜索最大值所在的位置

```
a = []
while True:
        x = int(input())
        if x == -1:
                break
        a.append(x)
maxidx = 0
for i in range(1, len(a)):
        if a[i] > a[maxidx]:
                maxidx = i
print(maxidx)
```

二分搜索

- 线性搜索当数据集很大的时候，搜索效率很低；当数据集中的数据已经排好序时，可以采用二分搜索，可快速找到目标。
- 二分搜索每次用中间位置的元素做比较，如果中间位置的元素比要搜索的大，就丢掉右边一半，否则丢掉左边的一半。这样的搜索，每次都把数据集分成两部分，所以就叫做二分搜索。

二分搜索算法

- 实现思路
  - 用变量left和right分别表示正在搜索的数据集的上下界。
    1. left=0，right=len(a)-1
    2. mid=(left-right)//2，如果a[mid]>x，则令right=mid-1，搜索范围缩小为左边的一半；如果a[mid]<x，则令left=mid+1，搜索范围缩小为右边的一半；如果a[mid]==x，则找到，位置是mid，搜索结束。
    3. 如果left<=right，重复步骤2，否则如果left>right，则表明未找到，搜索结束。

二分搜索代码

- 例4-23 二分搜索

```
a=[11, 14, 17, 24, 31, 31, 46, 52,61, 1, 62, 73, 80, 90, 92, 93]
x = int(input())
found = -1
left = 0                    #第一个元素下标
right = len(a)-1            #最后一个元素下标
while left<=right:
    mid = (left + right) // 2
    if a[mid] > x:
        right = mid - 1
    elif a[mid] < x:
        left = mid + 1
    else:                   # a[mid]==x
        found = mid
        break
print(found)
```

选择排序

很多场合（如：二分搜索)需要将数据集中的数据排序。选择排序和冒泡排序是常见的排序算法。

- 选择排序算法（升序）：
  1. 从数据集(假设n个数)中找出最大数与最后位置的数（下标n-1）交换。
  2. 从未排序的剩下的数据中（n-1个）找出最大值与倒数第2个位置的数（下标n-2）交换。

     …

  n-1. 从剩下2个数中找出最大值与第2个数交换。
  - 结束
- 如果是降序排序的话，则只要将上述算法中的最大值改为最小值即可。

选择排序（续）

- 以5个数为例，共需要前述的步骤重复4个轮次。
  - 原始数据：[90, 68, 31, 65, 87]
  - 第1轮次：　[87, 68, 31, 65, 90]
  - 第2轮次：　[65, 68, 31, 87, 90]
  - 第3轮次：　[65, 31, 68, 87, 90]
  - 第4轮次：　[31, 65, 68, 87, 90]

选择排序（续2）

例4-24 选择排序
将列表a中的元素按升序排列。

```
a=[80, 58, 73, 90, 31, 92, 39, 24, 14, 79, 46, 61, 31, 61, 93, 62, 11, 5
2, 34, 17]
for right in range(len(a), 1, -1):
    maxidx = 0
    for i in range(1, right):
        if a[i]>a[maxidx]:
            maxidx = i
        a[maxidx],a[right-1] = a[right-1], a[maxidx]
print(a)
```

程序用到双重循环，外循环控制轮次数，内循环用于找最大值。

冒泡排序

- 冒泡排序是一种简单的排序算法
  - 冒泡排序算法（升序）：
    1. 在数据集中依次比较相邻的2个数的大小，如果前面的数大，后面的数小，则交换；n个数需要比较n-1次，其结果是将最大的数交换到最后位置（下标n-1）。
    2. 从剩下的未排序数据中（n-1个）重复上述步骤，n-1个数需要比较n-2次，其结果是将次大的数交换到倒数第2个位置（下标n-2）。

       …

    n-1. 比较剩下2个数，如果前面的大，后面的小，则交换。
    - 结束。
  - 如果是降序排序的话，则只要将上述算法中的交换条件改成前面的数小，后面的数大即可。

冒泡排序（续）

- 以5个数为例，共需要前述的步骤重复4个轮次，每个轮次中需要比较相邻2个数n-i次（i为轮次数）。
  - 原始数据：[60, 56, 45, 31, 28]
  - 第1轮次：[56, 45, 31, 28, 60]
  - 第2轮次：[45, 31, 28, 56, 60]
  - 第3轮次：[31, 28, 45, 56, 60]
  - 第4轮次：[28, 31, 45, 56, 60]
- 在比较交换过程中，大的数逐步往后移动，相对来讲小的数逐步向前移动；如同水中的气泡慢慢向上升。

冒泡排序（续2）

例4-25 冒泡排序

```
a=[80, 58, 73, 90, 31, 92, 39, 24, 14, 79, 46, 61, 31, 61, 93, 62,
11, 52, 34, 17]
for right in range(len(a), 1, -1):
    for i in range(0, right-1):
            if a[i]>a[i+1]:
                    a[i], a[i+1] = a[i+1], a[i]
    print(a)
```

- 程序用到双重循环，外循环控制轮次数，内循环控制相邻2个数的比较（交换）。

4.5 异常处理

在程序运行过程中如果发生异常，Python 会输出错误消息和关于错误发生处的信息，然后终止程序。

例如：

```
>>> short_list = [1, 72, 3]
>>> position = 6
>>> short_list[position]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    short_list[position]
IndexError: list index out of range
```

- 程序由于访问了不存在的列表元素，而发生下标越界异常。

异常处理（续）

- 可使用try-except语句实现异常处理。

```
short_list = [1, 72, 3]
position = 6
try:
    short_list[position]
except:
    print('索引应该在 0 和', len(short_list)-1,"之间,但却是 ",position)
```

输出：

索引应该在 0 和 2 之间，但却是 6

异常处理（续2）

- 语法格式

```
try:
    语句块1
except 异常类型1:
    语句块2
except 异常类型2:
    语句块3
…
except 异常类型N:
    语句块N+1
except:
    语句块N+2
else:
    语句块N+3
finally:
   语句块N+4
```

异常处理（续3）

- 说明
  - 正常程序在语句块1中执行。
  - 如果程序执行中发生异常，中止程序运行，跳转到所对应的异常处理块中执行。
  - 在“except 异常类型 ”语句中找对应的异常类型，如果找到的话，执行后面的语句块。
  - 如果找不到，则执行“except”后面的语句块N+2。
  - 如果程序正常执行没有发生异常，则继续执行else后的语句块N+3。
  - 无论异常是否发生，最后都执行finally后面语句块N+4。

异常处理（续5）

表4-4 Python常见的标准异常

异常名称	描述
SystemExit	解释器请求退出
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除(或取模)零 (所有数据类型)
KeyboardInterrupt	用户中断执行(通常是输入^C)
ImportError	导入模块/对象失败
IndexError	序列中没有此索引(index)
RuntimeError	一般的运行时错误
AttributeError	对象没有这个属性
IOError	输入/输出操作失败
OSError	操作系统错误
KeyError	映射中没有这个键
TypeError	对类型无效的操作
ValueError	传入无效的参数

异常处理（续4）

例4-26 除数为0的异常处理

```
x=int(input())
y=int(input())
try:
    result = x / y
except ZeroDivisionError:
    print("division by zero!")
else:
    print("result is", result)
finally:
    print("executing finally clause")
```

程序输入：

```
5
0
```

程序输出：

```
division by zero!
executing finally clause
```

异常处理（续6）

- 有时需要除了异常类型以外其他的异常细节，可以使用下面的格式获取整个异常对象：

```
except Exception as name
```

- 前面讨论了异常处理，但是其中讲到的所有异常都是在Python或者它的标准库中提前定义好的。根据自己的目的可以使用任意的异常类型，同时也可以自己定义异常类型。

except Exception as name应用举例

```
short_list = [1, 2, 3]
while True:
    value = input('Position [q to quit]? ')
    if value == 'q':
        break
    try:
        position = int(value)
        print(short_list[position])
    except IndexError as err:
        print('Bad index:', position)
    except Exception as other:
        print('Something else broke:', other)
```

本章小结

- 分支结构if语句的用法
- 循环结构for语句的用法
- 循环结构while语句的用法
- 异常处理的程序结构
- break,continue语句的用法
- 二维数据处理

本章例题源代码

书本例题