# 第4章
# 条件、循环和其他语句

CS, ZJU

2018年12月

# 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
```

# 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))

# 猴子报数选大王（列表运行时会改变）

- 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])
```

# 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5）

- 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)
```

# 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)
```

# 鸡兔同笼----枚举算法1

- 今有鸡兔同笼，有三十五头，有九十四脚，问鸡兔各几何？

```
for k in range(0,36):
    for r in range(0,36):
        if 2*k+4*r ==94 and k+r==35:
            print(k,r)
```

- 如何优化？

# 最大切片问题---枚举算法2

- 在一个实数序列中，求和最大的切片。
- lst=[]
- n=int(input())
- for i in range(n):　#输入序列
- 　lst.append(int(input()))
- result=[sum(lst[i:j]) for i in range(len(n）\
- 　　for j in range(i+1,n+1)]
- print(max(result))

# 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")
```

# 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

# 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)]
- 请注意上面语句i是外层循环，j是内层循环

- 请注意下面语句i是内层循环，j是外层循环
- >>> [[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

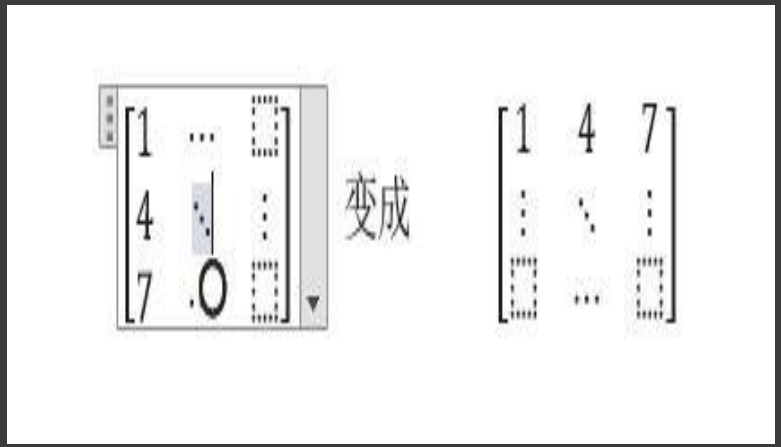
$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix} \longrightarrow \begin{bmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{bmatrix}$$

# 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]]

# 转置解题思路

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]

# 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]]

# 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 |