Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第六章 列表与元组

- **Python**中的字符串、列表和元组数据类型均属于序列类型，它们的每个元素是按照位置编号来顺序存取的，就像其他程序设计语言中的数组，但数组通常存储相同数据类型的元素，而列表和元组则可以存储不同类型的元素。
- 序列类型使得批量数据的组织和处理更加方便、灵活。

- 列表和元组在很多操作上是一样的，不同的是元组与字符串一样，是不可变的，而列表则可以修改。
- 在某些情况下，使用其中的一种类型要优于另一种。

- 例：
- 需要把一些重要数据传递给一个并不了解的函数，而希望数据不被这个函数修改，此时就应选择元组。
- 在处理动态数据时，需要经常更新这个数据集，此时就应该选列表。
- 这两种数据类型用处各异，它们之间的转换也十分方便。

6.1 序列的通用操作

- 所有序列类型都可以进行某些操作，包括索引、分片、序列相加、序列相乘以及检查某个元素是否属于序列的成员（成员资格）等。
- Python还有计算序列长度、找出最大元素和最小元素的内置函数等。

- 字符串是单个字符组成的序列，列表和元组是由任意类型数据组成的序列。
- 字符串、列表和元组的许多操作是相同。

- 例：
- >>> s=[[1,2,3],[4,5,6],[7,8,9,0]]
- >>> s[1]
- [4, 5, 6]
- >>> s[1][1]
- 5
- >>> s=["abc","def","ghijk"]
- >>> s[1]
- 'def'
- >>> s[1][1]
- 'e'

- 例：
- s=(list("abcd"),"123",(4,5,6,7))
- >>> s[1]
- '123'
- >>> s[1][2]
- '3'
- >>> s[2][1]
- 5
- >>> s[0][3]
- 'd'

6.1.1 序列的索引与分片

1. 序列的索引

- 序列中的每一个元素被分配一个位置编号，称为索引（ index ）。
- 第一个元素的索引为0，第二个元素的索引为1，依此类推。
- 序列的元素可以通过索引进行访问，一般格式为：
- 序列名[索引]

- 例：
- >>> s="uestc"
- >>> s[0]
- 'u'
- >>> s[-5]
- 'u'
- >>> s=[(1,2,3),"abcd",list("ABCDE")]
- >>> s[-1][3]
- 'D'
- >>> s[2][-2]
- 'D'

- 除了常见的正向索引外，序列还支持反向索引，即负数索引，可以从最后一个元素开始计数，最后一个元素的索引是-1，倒数第二个元素的索引是-2，依此类推。
- 使用负数索引，可以在无须计算序列长度的前提下很方便地定位序列中的元素。

- 例：
- s="uestc"
- >>> s[2:]
- 'stc'
- >>> s[-2:]
- 'tc'
- >>> s[:2]
- 'ue'
- >>> s[:-2]
- 'ues'

>>> s="uestc"

>>> s[::2]

'usc'

>>> s[::-2]

'csu'

>>> s[2::]

'stc'

>>> s[-2::]

'tc'

2. 序列的分片

- 分片（ slice ）就是取出序列中某一范围内的元素，从而得到一个新的序列。
- 序列分片的一般格式为：
- 序列名[起始索引:终止索引:步长]
- 起始索引是提取部分的第一个元素的编号，终止索引对应的元素不包含在分片范围内。
- 步长为非零整数，当步长为负数时，从右到左提取元素。

- 序列名[起始索引:终止索引:步长]
- 当忽略参数时，起始元素默认为第一个（索引为0），终止索引默认为最后一个（索引为-1），步长默认为1。

- 例：
- >>> n=[1,2,3,4,5,6,7,8,9]
- >>> n[3:6]
- [4, 5, 6]
- >>> n[-6:-3]
- [4, 5, 6]
- >>> n[0:1]
- [1]

- 如果需要从序列末尾开始计数，即如果分片所得部分包括序列末尾的元素，那么只需置空最后一个索引。
- 例：
- **>>> n=[1,2,3,4,5,6,7,8,9]**
- **>>> n[6:9]**
- **[7, 8, 9]**
- **>>> n[-3:]**
- **[7, 8, 9]**

- 例：
- >>> n=[1,2,3,4,5,6,7,8,9]
- >>> n[:3]
- [1, 2, 3]
- >>> n[3:]
- [4, 5, 6, 7, 8, 9]
- >>> n[:]
- [1, 2, 3, 4, 5, 6, 7, 8, 9]

- 例：
- >>> x=[1,2,3,4,5,6,7,8,9]
- >>> x[3]
- 4
- >>> x[3:]
- [4, 5, 6, 7, 8, 9]
- >>> x[3::]
- [4, 5, 6, 7, 8, 9]
- >>> x[:3]
- [1, 2, 3]
- >>> x[::3]
- [1, 4, 7]

- 在进行分片时，默认的步长是1。
- 如果设置的步长大于1，那么会跳过某些元素。
- 例：
- >>> n=[1,2,3,4,5,6,7,8,9]
- >>> n[0:9:2]
- [1, 3, 5, 7, 9]
- >>> n[3:6:3]
- [4]

- 步长不能为0，但是可以是负数，即从右到左提取元素。
- 步长为负时，必须让起始索引大于终止索引。
- 例：
- >>> n=[1,2,3,4,5,6,7,8,9]
- >>> n[8:0:-2]
- [9, 7, 5, 3]
- >>> n[0:8:-2]
- []

- 注意：
- 分片操作是产生新的序列，不会改变原来的序列。
- 序列变量之间的赋值语句则是给序列的内容再取一个名字，即两个序列变量都指向相同的存储内容，并没有实现存储内容的真正的复制。

- 例：
- >>> x=['a','b','c','d']
- >>> y=x[:]
- >>> id(x),id(y)
- (39632088, 3460312)
- >>> x=['a','b','c','d']
- >>> y=x
- >>> id(x),id(y)
- (39631928, 39631928)

6.1.2 序列的计算

1. 序列相加

- 通过使用加号可以进行序列的连接操作。
- 例：
- >>> [1,2,3]+[4,5,6]
- [1, 2, 3, 4, 5, 6]
- >>> [1,2,3]+[1,2,3,4,5]
- [1, 2, 3, 1, 2, 3, 4, 5]

- 只有两种相同类型的序列才能进行连接操作。
- 列表和字符串是无法连接到一起的，尽管它们都是序列。
- 同样，列表和元组也不能进行连接操作。
- 例：
- **>>> [1,2,3]+"uestc" #出现错误**
- **>>> [1,2,3]+(4,5,6) #出现错误**

2. 序列乘法

- 用整数n乘以一个序列会生成新的序列，在新序列中，原来的序列将重复n次。
- 当n<1时，将返回一个空列表。
- 例：
- >>> (2,)*4
- (2, 2, 2, 2)
- >>> (2)*4
- 8

- 例：
- >>> 4*(2)
- 8
- >>> 4*(2,)
- (2, 2, 2, 2)
- >>> 4*(1,2)
- (1, 2, 1, 2, 1, 2, 1, 2)
- >>> 4*(1,2,)
- (1, 2, 1, 2, 1, 2, 1, 2)

- 在序列中，Python的内置值None表示什么都没有，可作为占位符。
- 例：空列表可以通过中括号进行表示([])，但是如果想创建一个占用10个元素空间，却不包括任何有用内容的列表，就需要一个值来代表空值。

- 例：
- >>> s=[]*5
- >>> s
- []
- >>> len(s)
- 0
- >>> s=[None]*5
- >>> s
- [None, None, None, None, None]
- >>> len(s)
- 5

3. 序列比较

- 两个序列对象可以进行比较操作，其规则是两个序列的第1个元素先进行比较，如果第1个元素可以得出结果，那么就得出序列比较结果。
- 如果第1个元素相同，则继续比较下一个元素。
- 如果元素本身也是序列，则对元素进行以上过程。

- 例：
- >>> (1,2,4)>(1,2,3)
- True
- >>> (1,2,3,4)>(1,2,3)
- True
- >>> (1,2,3,-999)>(1,2,3)
- True
- >>> [1,2,['a','b','c'],0]>[1,2,['a','b'],8,9]
- True

- 两个不同的对象也可以进行比较操作，但要求它们是兼容的。
- 如果不兼容，则比较时会抛出TypeError错误。
- 例：
- >>> 3>2.8
- True
- >>> [1,2,3]>(1,2,3)　#出现错误
- >>> "abc">['a','b','c']　#出现错误
- >>> list("abc")>=['a','b','c']
- True

- **4. 成员资格**
- 成员资格用于检查一个值是否包含在某个序列中，Python使用in运算符检查元素的成员资格，并返回逻辑型的结果（True或False）。
- 例：
- >>> x=(1,2,3)
- >>> 3 in x
- True
- >>> 3 in "123"　#出现错误

6.1.3 序列处理函数

1.len()、max()和min()函数

- **len(s)**：返回序列中包含元素的个数,即序列长度。
- **min(s)**：返回序列中最小的元素。
- **max(s)**：返回序列中最大的元素。

- 例：
- >>> n=[23,4,67,15]
- >>> len(n)
- 4
- >>> max(n)
- 67
- >>> min(n)
- 4

- 例：
- >>> n=(3,2,1,4,6,5)
- >>> len(n)
- 6
- >>> max(n)
- 6
- >>> min(n)
- 1
- >>> x=(3,2,(1,4),6,5)
- >>> len(x)
- 5
- >>> max(x)　#出现错误
- >>> min(x))　#出现错误

- 例：
- >>> s="uestc"
- >>> len(s)
- 5
- >>> max(s)
- 'u'
- >>> min(s)
- 'c'

- 例：
- >>> x={1,2,3,1,2,3,9,6}　#集合是无序类型
- >>> len(x)
- 5
- >>> max(x)
- 9
- >>> min(x)
- 1

- 可以用列表来表示矩阵或锯齿阵。
- 当列表的元素也是一个列表时，列表就表示二维矩阵或锯齿阵。
- 例：
- >>> s=[[1,2,3],[4,5],[6,7,8,9]]
- >>> len(s)
- 3
- >>> len(s[0])
- 3
- >>> len(s[1])
- 2
- >>> len(s[2])
- 4

- 例：请写出下面程序的运行结果。
- s=[[1,2,3],[4,5],[6,7,8,9]]
- for i in range(len(s)):
- 　for j in range(len(s[i])):
- 　　print("{}".format(s[i][j]),end=" ")
- 　print()

```
1 2 3
4 5
6 7 8 9
```

- 例：请写出下面程序的运行结果。
- n=5
- s=[[0]*n for i in range(n)]
- for k in range(n):
- 　　s[k][k]=1
- for i in range(n):
- 　　for j in range(n):
- 　　　　print(s[i][j],end=" ")
- 　　print()

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

- 例：请写出下面程序的运行结果。
- s=[5,3,4,2,6]
- t=[[i]*(s[i]) for i in range(5)]
- for i in range(len(t)):
- 　for j in range(len(t[i])):
- 　　print(t[i][j],end=" ")
- 　print()

```
0 0 0 0 0
1 1 1
2 2 2 2
3 3
4 4 4 4 4 4
```

- 例：请写出下面程序的运行结果。
- m=4
- n=5
- t=[[i]*n for i in range(m)]
- for i in range(len(t)):
- 　for j in range(len(t[i])):
- 　　print(t[i][j],end=" ")
- 　print()

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

- 例：请写出下面程序的运行结果。
- m=4
- n=5
- t=[[n-i-1]*n for i in range(m)]
- for i in range(len(t)):
- 　print(t[i])

```
[4, 4, 4, 4, 4]
[3, 3, 3, 3, 3]
[2, 2, 2, 2, 2]
[1, 1, 1, 1, 1]
```

2. sum()函数和reduce()函数

- **sum(s)**：返回序列s中所有元素的和。要求元素必须为数值，否则出现TypeError错误。
- 例：
- >>> s=[1,2.3,4]
- >>> sum(s)
- 7.3
- >>> s=["1","2"]
- >>> sum(s)　#出现错误

- 例：
- >>> x=(1,2,3,4,5)
- >>> sum(x)
- 15
- >>> y=(1,2,(3,4),5)
- >>> sum(y)　#出现错误
- >>> z={1,2,3,4}　#集合是无序类型
- >>> sum(z)
- 10

- reduce()函数位于functools模块中。
- 对序列中元素的连续操作可以通过循环来实现，也可以用reduce()函数实现。但在大多数情况下，循环实现的程序更有可读性。

- **reduce(f,s[,n])：reduce()函数把序列s的前两个元素作为参数，传给函数f，返回计算的结果和序列的下一个元素重新作为f的参数，直到序列的最后一个元素。**
- **reduce()函数的返回值是函数f的返回值。**

- 例：
- >>> def add(x,y):return x+y
- >>> import functools
- >>> functools.reduce(add,range(5))
- 10
- >>> functools.reduce(add,range(5),10)
- 20
- >>> functools.reduce(add,[1,2,3],4)
- 10

- 例：
- >>> def f(x,y):return x*y
- >>> import functools
- >>> functools.reduce(f,(1,),2)
- 2
- >>> functools.reduce(f,(1),2) #出现错误
- #TypeError: reduce() arg 2 must support iteration
- >>> functools.reduce(f,1,2) #出现错误

3. enumerate()和zip()函数

- enumerate(iter)：接收一个可迭代对象作为参数，返回一个enumerate对象，该对象生成由iter每个元素的索引值和元素值组成的元组。
- 例：
- >>> s=["aa","bb","cc"]
- >>> for i,obj in enumerate(s):
- print(i,obj)
- 0 aa
- 1 bb
- 2 cc

- 例：
- >>> s="abcd"
- for i,obj in enumerate(s):
- 	print(i,obj)
- 0 a
- 1 b
- 2 c
- 3 d

- 例：
- >>> s=(2,56,3,7)
- >>> for i,j in enumerate(s):
- print(i,j)
- 0 2
- 1 56
- 2 3
- 3 7

- **zip([s0，s1，…，sn])：接收任意多个序列作为参数，返回一个可迭代对象，其第一个元素是s0，s1，…，sn这些元素的第一个元素组成的一个元组，后面的元素依此类推。**
- 若参数的长度不等，则返回列表的长度和参数中长度最短的对象相同。
- 利用*号操作符，可以将对象解压还原。

- 例：
- >>> s=zip([1,2,3],['a','b','c'])
- >>> s
- <zip object at 0x02613F80>
- >>> t=list(s)
- >>> t
- [(1, 'a'), (2, 'b'), (3, 'c')]

- 例：
- >>> s=zip("abc","1234567","ABCDE")
- >>> t=list(s)
- >>> t
- [('a', '1', 'A'), ('b', '2', 'B'), ('c', '3', 'C')]

- 例：
- >>> s=zip("abc","123")
- >>> t=list(s)
- >>> t
- [('a', '1'), ('b', '2'), ('c', '3')]
- >>> m=zip(*t)
- >>> list(m)
- [('a', 'b', 'c'), ('1', '2', '3')]
- # zip(*t)，即在对象前面加个星号，是上述操作的逆操作。

4. sorted()函数和reversed()函数

- sorted(iterable,key=None,reverse=False)：函数返回对可迭代对象iterable中元素进行排序后的列表，函数返回副本，原始输入不变。
- iterable是可迭代类型；
- key指定一个接收一个参数的函数，这个函数用于计算比较的键值，默认值为None；
- reverse代表排序规则，当reverse为True时按降序排序；reverse为False时按升序，默认按升序。

- 例：
- >>> s=["bb","dd","aa","cc"]
- >>> sorted(s)
- ['aa', 'bb', 'cc', 'dd']
- >>> sorted(s,reverse=True)
- ['dd', 'cc', 'bb', 'aa']
- >>> sorted(s,reverse=False)
- ['aa', 'bb', 'cc', 'dd']

- 例：
- >>> x=(23,14,78,4,12)
- >>> sorted(x)
- [4, 12, 14, 23, 78]
- >>> sorted(x,reverse=True)
- [78, 23, 14, 12, 4]
- >>> sorted(x,reverse=False)
- [4, 12, 14, 23, 78]

- 例：
- >>> s="uestc"
- ['c', 'e', 's', 't', 'u']
- >>> sorted(s,reverse=True)
- ['u', 't', 's', 'e', 'c']
- >>> sorted(s,reverse=False)
- ['c', 'e', 's', 't', 'u']

- 例：
- >>> s="uEStC"
- >>> sorted(s)
- ['C', 'E', 'S', 't', 'u']
- sorted(s,key=str.lower)
- ['C', 'E', 'S', 't', 'u']
- >>> sorted(s,key=str.upper)
- ['C', 'E', 'S', 't', 'u']

- 例：
- >>> s="A[b]Cd"
- >>> sorted(s)
- ['A', 'C', '[', ']', 'b', 'd']
- >>> sorted(s,key=str.lower)
- ['[', ']', 'A', 'b', 'C', 'd']
- >>> sorted(s,key=str.upper)
- ['A', 'b', 'C', 'd', '[', ']']

```
>>> ord("A")
65
>>> ord("C")
67
>>> ord("[")
91
>>> ord("]")
93
>>> ord("b")
98
>>> ord("d")
100
```

- reversed(iterable)：对可迭代对象iterable的元素按逆序排列，返回一个新的可迭代变量。
- 例：
- >>> s="abcd"
- >>> for i in reversed(s):
- 	print(i,end="")
- dcba

5. all()和any()函数

- 设s为一个序列，下面的内置函数可用于列表、元组和字符串。
- **all(s)**：如果序列s所有元素都为True，则返回True，否则返回False。
- **any(s)**：如果序列任一元素为True，则返回True，否则返回False。

- 例：
- >>> x=[1,2,0,3]
- >>> all(x)
- False
- >>> any(x)
- True

- 例：
- >>> y=(2,"abc",True,False)
- >>> all(y)
- False
- >>> any(y)
- True

- 例：
- >>> s="ab102"
- >>> all(s)
- True
- >>> any(s)
- True

6.1.4 序列拆分赋值

- 使用赋值语句，可以将序列赋给一个变量，也可以将序列拆分，赋给多个变量。
- 变量个数和序列元素的个数不一致时，将导致**ValueError**错误。

- 例：
- >>> x=[1,2,3,4]
- >>> x
- [1, 2, 3, 4]
- >>> a,b,c,d=[1,2,3,4]
- >>> print(a,b,c,d)
- 1 2 3 4

- 例：
- >>> x=(1,2,3,4)
- >>> a,b,c=x #出现错误
- >>> a,b,c,d=x
- >>> print(a,b,c,d)
- 1 2 3 4
- >>> m=x
- >>> m
- (1, 2, 3, 4)

- 当值太多，变量太少时，可以在变量名前面加星号(*)，将序列的多个元素值赋给相应的变量。
- 例：
- **>>> s="abcdef"**
- **>>> x,*y,z=s**
- **>>> print(x,y,z)**
- **a ['b', 'c', 'd', 'e'] f**

- 例：
- >>> x=[1,2,3,4,5,6]
- >>> a,*b,c,d=x
- >>> print(a,b,c,d)
- 1 [2, 3, 4] 5 6
- >>> m,n,*t,a=x
- >>> print(m,n,t,a)
- 1 2 [3, 4, 5] 6

- 注意：
- 加星号的变量只允许一个，否则会出现语法错误 syntaxError。
- 例：
- >>> s="abcdef"
- >>> *m,*n=s　#出现错误

6.2 列表的专有操作

6.2.1 列表的基本操作

- 列表是可变的序列，因此可以改变列表的内容。
- 列表可以使用所有适用于序列的标准操作，例如
- 索引、分片、连接和乘法。
- 还具有一些可以改变列表的方法，包括元素赋值、元素删除、分片赋值等。

1. 元素赋值

- 使用索引编号来为某个特定的元素赋值，从而可以修改列表。
- 但不能为一个位置不存在的元素进行赋值。即列表索引不能超出列表的范围。

- 例：
- >>> x=[1,1,1]
- >>> x[1]=2
- >>> x
- [1, 2, 1]
- >>> x[5]=3　#出现错误

2. 元素删除

- 从列表中删除元素，可使用del语句来实现。
- 除了删除列表中的元素外，del语句还能用于删除其他对象。

- 例：
- >>> x=[1,2,3,4]
- >>> del x[2]
- >>> x
- [1, 2, 4]
- >>> del x
- >>> x　#出现错误

3. 分片赋值

- 使用分片赋值可以给列表的多个元素提示赋值。
- 在使用分片赋值时，可以使用与原序列不等长的序列将分片替换。

- 例：
- >>> s=[1,2,3,4]
- >>> s[2:]=[5,6,7]
- >>> s
- [1, 2, 5, 6, 7]
- >>> s[2]=[8,9]
- >>> s
- [1, 2, [8, 9], 6, 7]

- 分片赋值语句可以在不需要替换任何原有元素的情况下插入新的元素。
- 也可以通过分片赋值来删除元素。
- 分片赋值时，如果分片步长等于l，则值列表的长度没有要求；如果分片步长大于1，则值列表的长度必须等于分片长度。

- 例：
- >>> s=[1,2,3]
- >>> s[1:1]=[4,5]
- >>> s
- [1, 4, 5, 2, 3]
- >>> s[1:4]=[]
- >>> s
- [1, 3]
- >>> s[:0]=[2,4]
- >>> s
- [2, 4, 1, 3]

- 例：
- >>> x=[1,2,3,4]
- >>> x[1:3:1]=[5,6,7,8]
- >>> x
- [1, 5, 6, 7, 8, 4]
- >>> x[1:4:2]=[22,33,44,55]　#出现错误
- # ValueError: attempt to assign sequence of size 4 to extended slice of size 2
- >>> x[1:4:2]=[22,33]
- >>> x
- [1, 22, 6, 33, 8, 4]

- 例：
- >>> s=list("abcdef")
- >>> s
- ['a', 'b', 'c', 'd', 'e', 'f']
- >>> s[1::2]=[1,2,3]
- >>> s
- ['a', 1, 'c', 2, 'e', 3]
- >>> s[1::2]=[4,5]　　#出现错误
- >>> s[1::2]=[4,5,6,7]　#出现错误
- >>> s[1::2]=[4,5,6]
- >>> s
- ['a', 4, 'c', 5, 'e', 6]

4. 列表解析

- 在一个序列的值上应用一个任意表达式，将其结果收集到一个新的列表中并返回。
- 它的基本形式是一个中括号里面包含一个for语句，对一个可迭代对象进行迭代。
- 例：
- **>>> [i for i in range(5)]**
- **[0, 1, 2, 3, 4]**
- **>>> [i**2 for i in range(5)]**
- **[0, 1, 4, 9, 16]**

- 例：
- >>> s=[ord(x) for x in "uestc"]
- >>> s
- [117, 101, 115, 116, 99]
- >>> t=[i-32 for i in s]
- >>> t
- [85, 69, 83, 84, 67]
- >>> y=[chr(j) for j in t]
- >>> y
- ['U', 'E', 'S', 'T', 'C']

- 在列表解析中，可以增加测试语句和嵌套循环。
- 一般形式是：
- [表达式 for 目标1 in可迭代对象1 [if 条件1] ……
- for 目标n in 可迭代对象n [if 条件 n]
- 任意数量嵌套的for循环同时关联可选的if测试，其中if测试语句是可选的，for上下之间表示的是一个嵌套关系。

- 例：
- >>> [(x,y) for x in range(5) if x%2==0 for y in range(5) if y%2==1]
- [(0, 1), (0, 3), (2, 1), (2, 3), (4, 1), (4, 3)]
- >>> s="uEsTC"
- >>> m=[i for i in s if 'A'<=i<='Z']
- >>> m
- ['E', 'T', 'C']
- >>> n=[j for j in s if 'a'<=j<='z']
- >>> n
- ['u', 's']

- 例：
- >>> s=[[1,2,3],[4,5,6],[7,8,9]]
- >>> [a[1] for a in s]
- [2, 5, 8]
- >>> [s[b][1] for b in range(3)]
- [2, 5, 8]
- >>> s[1][2]
- 6
- >>> [s[1][c] for c in range(3)]
- [4, 5, 6]

6.2.2 列表的常用方法

- Python中字符串、列表和元组实际上是对象。

1. 适用于序列的方法

- 下面的方法主要起查询功能，不改变序列本身，可用于表、元组和字符串。
- 方法中s为序列，x为元素值。
- s.count(x)：返回x在序列s中出现的次数。
- s.index(x)：返回x在s中第一次出现的下标。

- 例：
- >>> x=[1,2,1,2,1,2,3]
- >>> x.count(2)
- 3
- >>> s="Goodbye!"
- >>> s.count("o")
- 2
- >>> s=((1,2),2,(1,2,3))
- >>> s.count(2)
- 1

- 例：
- >>> x=[1,2,1,2,3]
- >>> x.index(2)
- 1
- >>> s="ababc"
- >>> s.index("b")
- 1
- >>> t=([1,2],[1,2,3],2,3)
- >>> t.index(2)
- 2
- >>> t.index([1,2])
- 0

- 例：
- >>> s=["ab","cd"]
- >>> s.count("c")
- 0
- >>> s.index("c")　#出现错误
- # ValueError: 'c' is not in list

2. 只适用于列表的方法

- 以下方法都是在原来的列表上进行操作的，会对原来的列表产生影响，而不是返回一个新列表。
- 由于元组和字符串的元素不可变更，下面方法只适用于列表。
- **s.append(x)**：在列表s的末尾附加x元素。
- **s.extend(s1)**：在列表s的末尾添加列表s1的所有元素。

- 例：
- >>> s=[1,2,3]
- >>> s.append(4)
- >>> s
- [1, 2, 3, 4]
- >>> s[0:0]=[5]
- >>> s
- [5, 1, 2, 3, 4]
- >>> s.append([6])
- >>> s
- [5, 1, 2, 3, 4, [6]]

- 例：
- >>> x=[1,2,3]
- >>> x.append(4)
- >>> x
- [1, 2, 3, 4]
- >>> x.extend([5,6,7])
- >>> x
- [1, 2, 3, 4, 5, 6, 7]
- >>> x.extend(8) # 出现错误
- #TypeError: 'int' object is not iterable

- **s.sort()**：对列表s中的元素排序。
- 如果需要获得排序后的副本，可以使用**sorted()**函数，该函数可用于任何可迭代对象。
- **sort**方法中可以使用**key**、**reverse**参数，其用法与**sorted()**函数相同。

- 例：
- >>> x=[3,1,2,4]
- >>> x.sort()
- >>> x
- [1, 2, 3, 4]
- >>> s=list("uestc")
- >>> s
- ['u', 'e', 's', 't', 'c']
- >>> s.sort()
- >>> s
- ['c', 'e', 's', 't', 'u']

- 例：
- >>> x=[3,1,2,4]
- >>> x.sort(reverse=True)
- >>> x
- [4, 3, 2, 1]
- >>> x.sort(reverse=False)
- >>> x
- [1, 2, 3, 4]

- 例：
- >>> s=["one","three","five","python"]
- >>> s.sort()
- >>> s
- ['five', 'one', 'python', 'three']
- >>> s.sort(key=len)
- >>> s
- ['one', 'five', 'three', 'python']
- >>> s.sort(key=len,reverse=True)
- >>> s
- ['python', 'three', 'five', 'one']

- 例：
- >>> x=[3,1,2,4]
- >>> y=sorted(x)
- >>> y
- [1, 2, 3, 4]
- >>> id(x),id(y)
- (37519448, 37583528)
- >>> s="uestc"
- >>> t=sorted(s)
- >>> t
- ['c', 'e', 's', 't', 'u']

- 例：
- >>> s=["one","three","five","python"]
- >>> t=sorted(s)
- >>> t
- ['five', 'one', 'python', 'three']
- >>> t=sorted(s,key=len)
- >>> t
- ['one', 'five', 'three', 'python']
- >>> t=sorted(s,key=len,reverse=True)
- >>> t
- ['python', 'three', 'five', 'one']

- **s.reverse()：将列表s中的元素逆序排列。**
- 例：
- **>>> s=[1,2,3,4]**
- **>>> s.reverse()**
- **>>> s**
- **[4, 3, 2, 1]**
- **>>> x=(1,2,3,4)**
- **>>> x.reverse() #出现错误**
- **#AttributeError: 'tuple' object has no attribute 'reverse'**

- 例：
- >>> x=[1,[2,3],4,5]
- >>> x.reverse()
- >>> x
- [5, 4, [2, 3], 1]
- >>> x=[(1,2),3,4]
- >>> x.reverse()
- >>> x
- [4, 3, (1, 2)]

- **s.pop([i])：删除并返回列表s中指定位置i的元素，默认是最后一个元素。若i超出列表长度，则抛出IndexError异常。**

- 例：
- >>> x=[1,2,3,4]
- >>> x.pop()
- 4
- >>> x
- [1, 2, 3]
- >>> x.pop(0)
- 1
- >>> x
- [2, 3]
- >>> x.pop(5)　#出现错误
- # IndexError: pop index out of range

- **s.insert(i,x)**：在列表s的i位置处插入x。如果i大于列表的长度，则插入到列表最后。

- 例：
- >>> s=[1,2,3,4]
- >>> s.insert(1,5)
- >>> s
- [1, 5, 2, 3, 4]
- >>> s.insert(-1,6)
- >>> s
- [1, 5, 2, 3, 6, 4]

- 例：
- >>> s=[1,2,3,4]
- >>> s.insert(20,5)
- >>> s
- [1, 2, 3, 4, 5]
- >>> s.insert(-20,6)
- >>> s
- [6, 1, 2, 3, 4, 5]

- s.remove(x)：从列表s中删除x。
- 若x不存在，则抛出ValueError异常。
- 若列表中包含多个待删除的元素，则只删第一个。
- 例：
- >>> s=[1,2,3,2,4]
- >>> s.remove(2)
- >>> s
- [1, 3, 2, 4]

- 例：
- >>> s=[1,2,3,4,5]
- >>> s.remove(3)
- >>> s
- [1, 2, 4, 5]
- >>> s.remove(6)　#出现错误
- #ValueError: list.remove(x): x not in list

6.3 元组与列表的比较

1. 元组与列表的区别

- **Python**元组和列表一样，都是有序序列，在很多情况下可以相互替换，很多操作也类似，但它们也有区别。

- （1）元组是不可变的序列类型，元组能对不需要改变的数据进行写保护，使数据更安全。
- 列表是可变的序列类型，可以添加、删除或搜索列表中的元素。

- （2）元组使用小括号定义用逗号分隔的元素，而列表中的元素应该包括在中括号中。
- 虽然元组使用小括号，但访问元组元素时，要使用中括号按索引或分片来获得对应元素的值。
- （3）元组可以在字典中作为关键字使用，而列表不能作为字典关键字使用，因为列表不是不可改变的。

- （4）只要不尝试修改元组，那么大多数情况下把它们作为列表来进行操作。

2. 元组元素的可变性

- 元组中的数据一旦定义就不允许更改。
- 元组没有append()方法、extend()方法或insert()方法，无法向元组中添加元素；
- 元组也没有pop()方法或remove()方法，不能从元组中删除元素；
- 元组也没有sort方法或revese()方法，不能修改元组的值。

- 删除元组的元素是不可能的，但可以使用del语句删除整个元组。
- 例：
- >>> s=(1,2,3)
- >>> del s[1] #出现错误
- #TypeError: 'tuple' object doesn't support item deletion
- >>> del s
- >>> s
- NameError: name 's' is not defined

- 元组的不可变特性只是对于元组本身而言的，如果元组内的元素是可变的对象，则是可以改变其值的。
- 例：
- >>> s=(1,[2,3,4],5,6)
- >>> s[1][2]=7
- >>> s
- (1, [2, 3, 7], 5, 6)
- >>> del s[1][2]
- >>> s
- (1, [2, 3], 5, 6)

- 例：
- >>> s=(6,[3,2,5,4],7)
- >>> s[1].sort()
- >>> s
- (6, [2, 3, 4, 5], 7)
- >>> s[1].append(8)
- >>> s
- (6, [2, 3, 4, 5, 8], 7)
- >>> s[1].reverse()
- >>> s
- (6, [8, 5, 4, 3, 2], 7)

- 例：
- >>> s=(1,[2,3],4)
- >>> s[1].insert(2,5)
- >>> s
- (1, [2, 3, 5], 4)
- >>> s[1].pop()
- 5
- >>> s
- (1, [2, 3], 4)
- >>> s[1].remove(2)
- >>> s
- (1, [3], 4)

- 例：
- >>> s=(1,[2],3)
- >>> del s[1][0]
- >>> s
- (1, [], 3)
- >>> len(s)
- 3
- >>> s[1].extend([2])
- >>> s
- (1, [2], 3)

- 可以利用现有元组的部分来创建新的元组。
- 例：
- **>>> t=(1,2)**
- **>>> id(t)**
- **37542488**
- **>>> t=t+(3,4)**
- **>>> t**
- **(1, 2, 3, 4)**
- **>>> id(t)**
- **37544944**

3.元组与列表的转换

- 元组和列表可以通过list()函数和tuple()函数实现相互转换。
- list()函数接收一个元组参数，返回一个包含同样元素的列表；
- tuple()函数接收一个列表参数，返回一个包含同样元素的元组。
- 从实现效果上看，tuple()函数冻结列表，达到保护的目的；而list()函数融化元组，达到修改的目的。

- 例：
- >>> s=(1,2,3)
- >>> t=list(s)
- >>> t
- [1, 2, 3]
- >>> x=tuple(t)
- >>> x
- (1, 2, 3)

- 例：
- >>> s=((1,2),[3,4])
- >>> t=list(s)
- >>> t
- [(1, 2), [3, 4]]
- >>> x=tuple(t)
- >>> x
- ((1, 2), [3, 4])

- 例：
- >>> s="abcd"
- >>> s
- 'abcd'
- >>> t=tuple(s)
- >>> t
- ('a', 'b', 'c', 'd')
- >>> x=list(s)
- >>> x
- ['a', 'b', 'c', 'd']

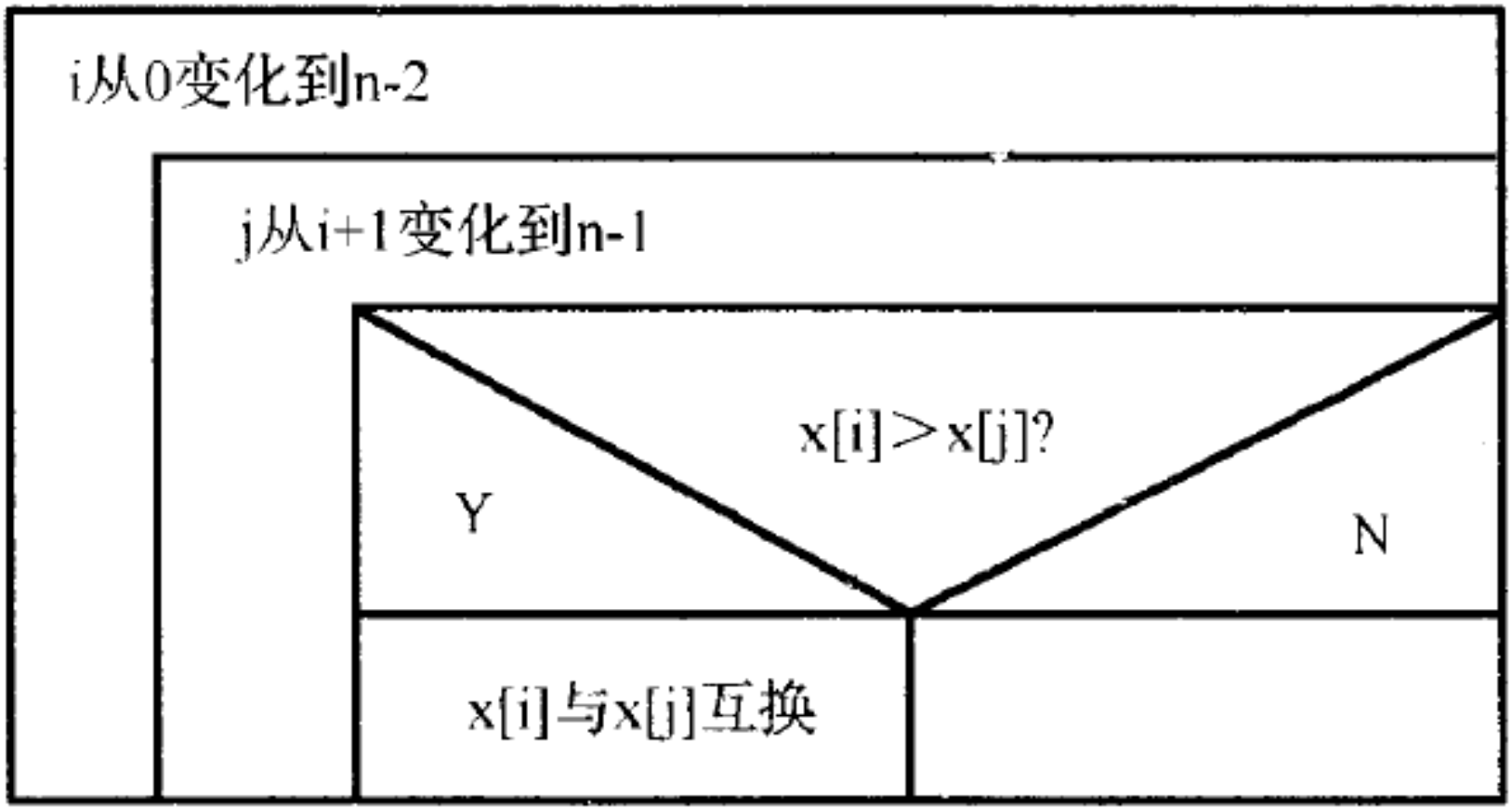
6.4 序列的应用

- 为了加强程序设计基本方法的训练，在此主要从原始的程序设计思路出发，构造算法并编写程序，而并没有过多利用Python本身的功能。
- 在实际应用中，我们完全可以充分利用Python的特点和资源，写出具有Python特征的程序。

6.4.1 数据排序

- 数据排序（sort）是程序设计中很典型的一类算法。
- 在Python中，数据排序可以直接使用sort()方法或sorted()函数，也可以自己编写排序的程序。

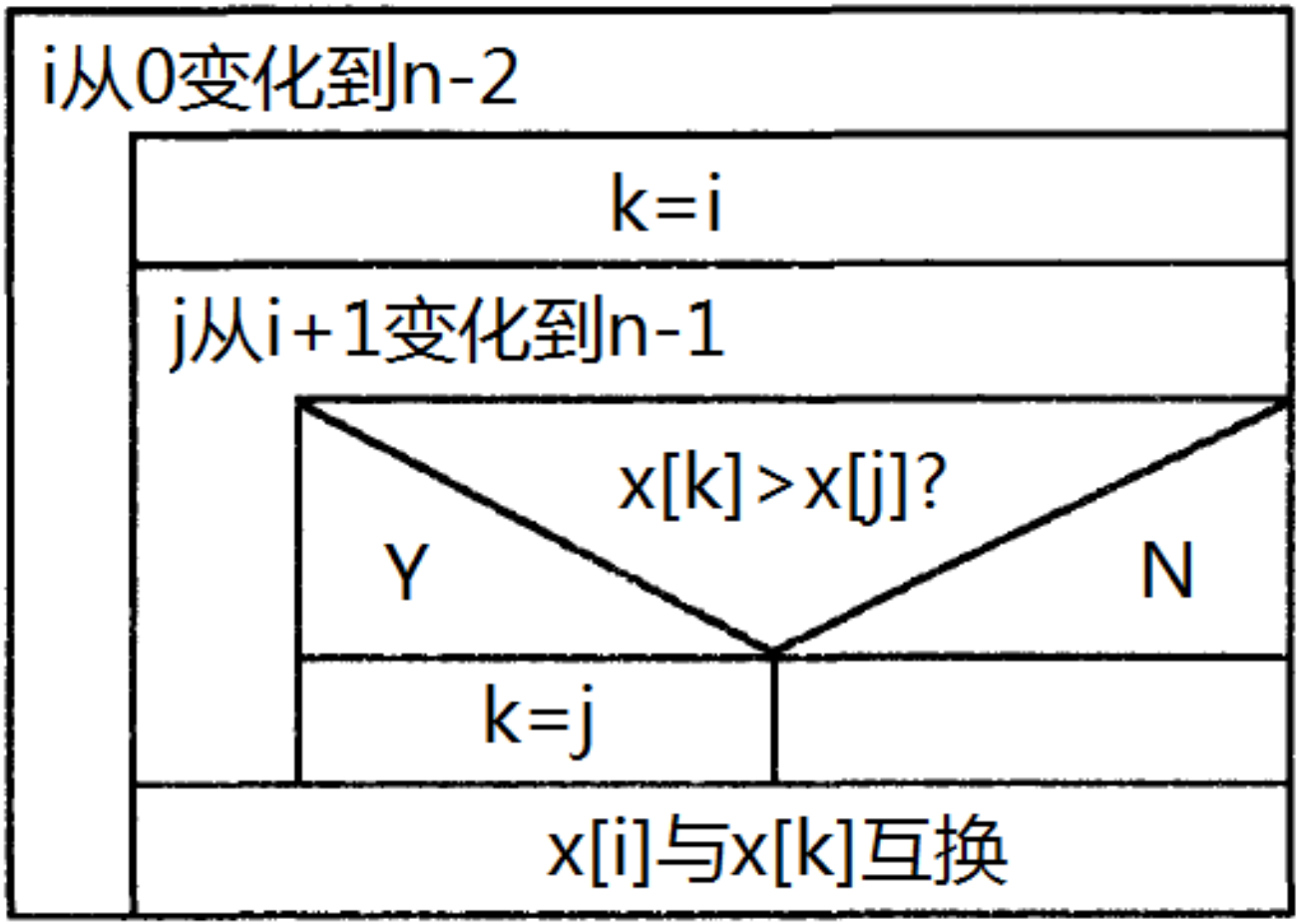
- 例：利用简单交换排序法，将n个数按从小到大顺序排列后输出。



简单交换排序法流程

```
n=int(input('输入数据个数:'))
x=[]
for i in range(n):
    x.append(int(input('输入一个数:')))
for i in range(n-1):            #控制比较的轮数
    for j in range(i+1,n):      #控制每轮比较的次数
        if x[i]>x[j]:   #排在最前面的数与后面的数依次进行比较
            x[i],x[j]=x[j],x[i]
print("排序后数据:",x)
```

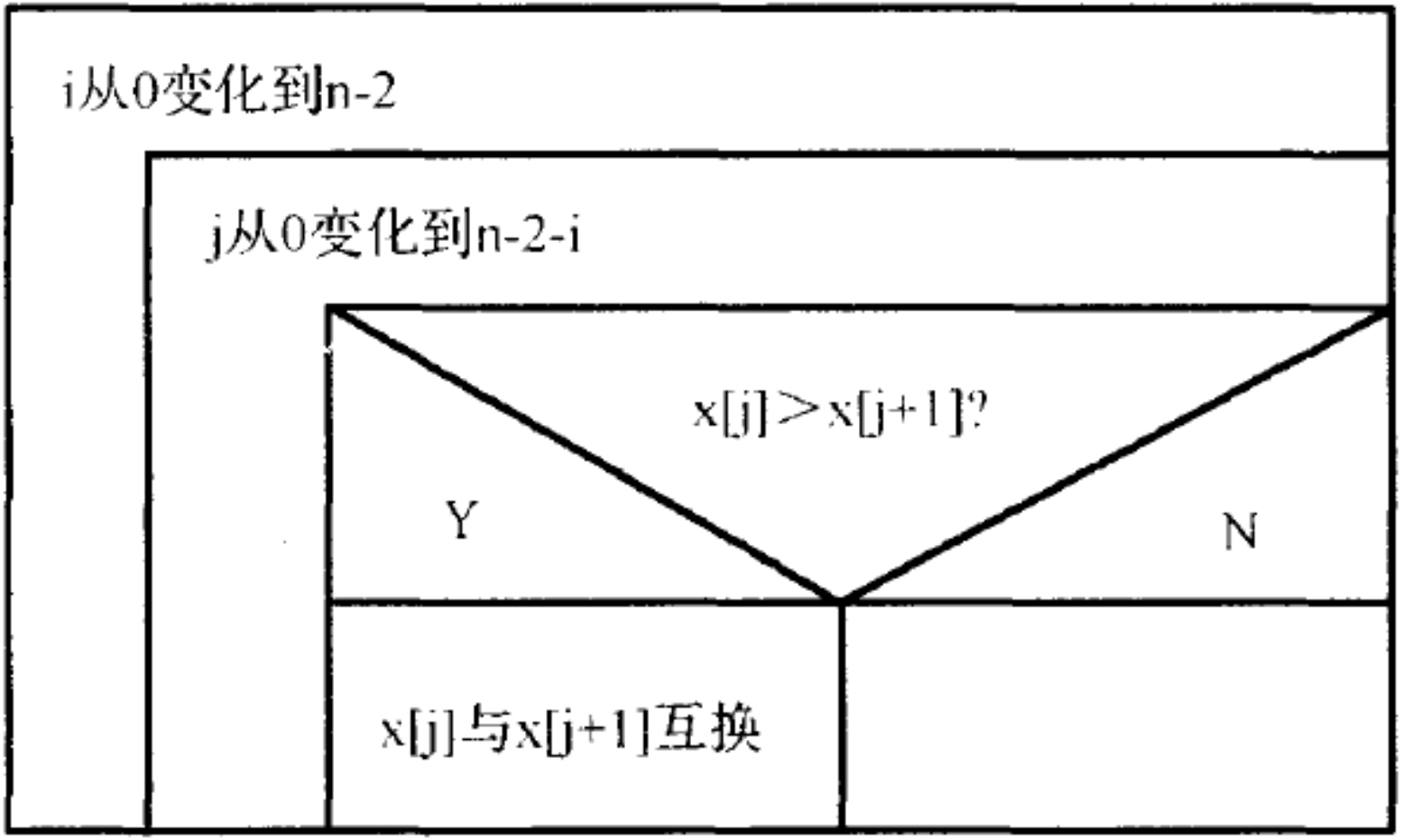
- 例：利用选择排序法，将n个数按从小到大顺序排列后输出。



选择排序算法流程

```
n=int(input('输入数据个数:'))
x=[]
for i in range(n):
    x.append(int(input('输入一个数:')))
for i in range(n-1):
    k=i
    for j in range(i+1,n):    #找最小数的下标
        if x[k]>x[j]:k=j
    if k!=i:                  #将最小数和排在最前面的数互换
        x[i],x[k]=x[k],x[i]
print("排序后数据:",x)
```

- 例：利用冒泡排序法，将n个数按从小到大顺序排列后输出。



冒泡排序法流程

```
n=int(input('输入数据个数:'))
x=[]
for i in range(n):
    x.append(int(input('输入一个数:')))
for i in range(n-1):
    for j in range(n-1-i):
        if x[j]>x[j+1]:    #相邻的两个数两两进行比较
            x[j],x[j+1]=x[j+1],x[j]
print("排序后数据:",x)
```

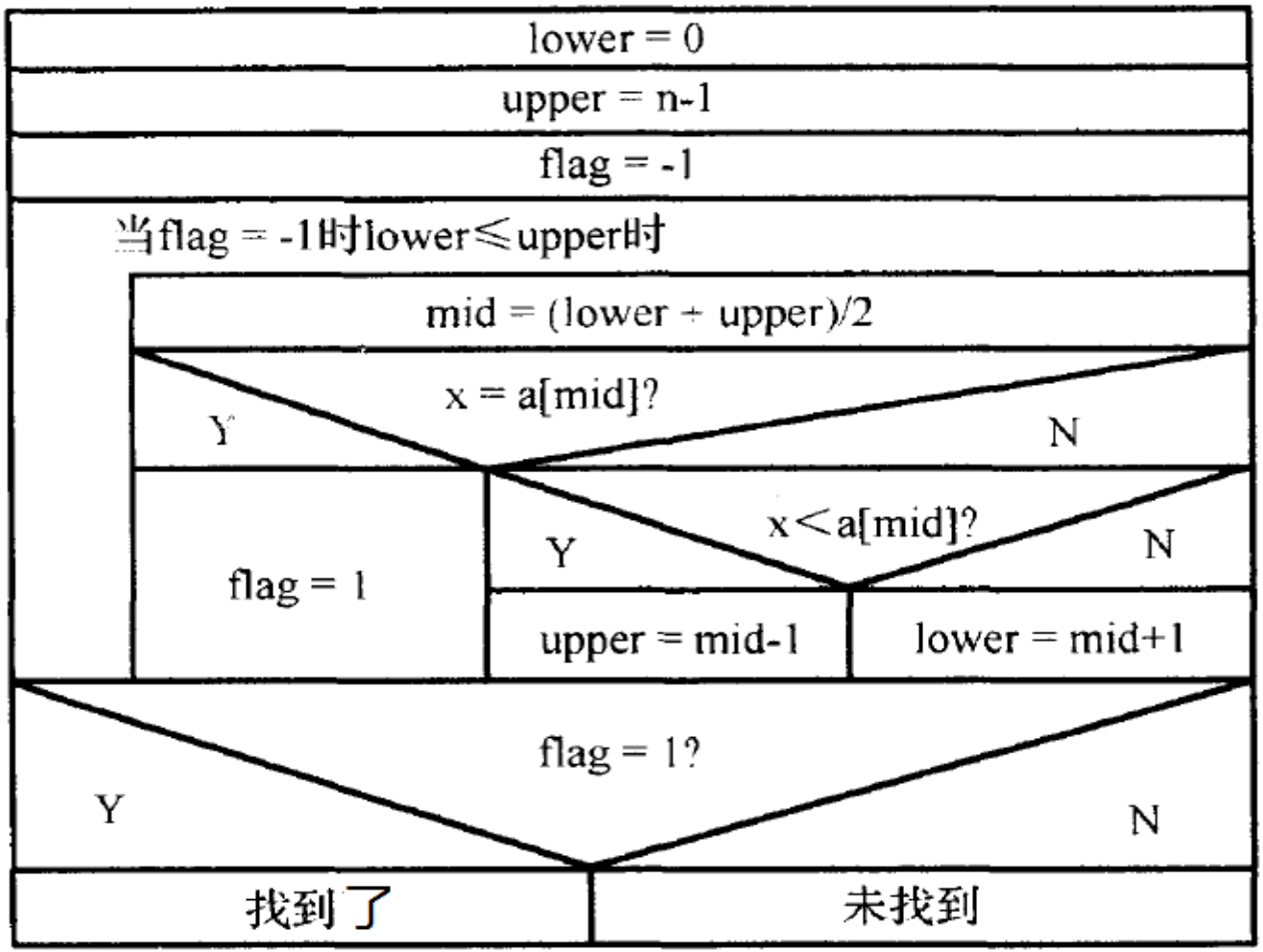
6.4.2 数据查找

- 数据查找（search）是从一组数据中找出具有某种特征的数据项，它是数据处理中应用很广泛的一种操作。
- 常见的数据查找方法有顺序查找（sequential search）和二分查找（binary search）。

- 例：设有n个数已存在序列a中，利用顺序查找法查找数据x是否在序列a中。

```
a=eval(input())
x=eval(input("输入待查数据:"))
n=len(a)
i=0
while i<n and a[i]!=x:        #顺序查找
    i+=1
if i<n:
    print("已找到",x)
else:
    print("未找到",x)
```

- 例：设有n个数已按大小顺序排列好并存于序列a中，利用二分查找法查找数据x是否在序列a中。



二分查找法的算法描述

```
a=eval(input())
x=eval(input("输入待查数据:"))
n=len(a)
lower=0
upper=n-1
flag=-1
while flag==-1 and lower<=upper:          #二分查找
    mid=int((lower+upper)/2)
    if x==a[mid]:                         #已找到
        flag=1
    elif x<a[mid]:                        #未找到
        upper=mid-1
    else:                                 #未找到
        lower=mid+1
if flag==1:
    print("已找到",x)
else:
    print("未找到",x)
```

6.4.3 矩阵运算

- 矩阵运算包括矩阵的建立、矩阵的基本运算、矩阵的分析与处理等操作。
- **Python**的矩阵运算功能非常丰富，应用也非常广泛。
- 许多含有矩阵运算的复杂计算问题，在**Python**中很容易得到解决。

- 例：给定一个m×n矩阵，其元素互不相等，求每行绝对值最大的元素及其所在列号。

```
m,n=eval(input())       #输入矩阵的行数和列数
x=[[0]*n for i in range(m)]    #定义矩阵x
for i in range(m):       #输入矩阵的值
    for j in range(n):
        x[i][j]=eval(input())
print("Matrix x:")       #输出x矩阵的值
for i in range(len(x)):
    print(x[i])
for i in range(m):
    k=0    #假定第0列元素是第i行绝对值最大的元素
    for j in range(1,n):
        if abs(x[i][j])>abs(x[i][k]):  #求第i行绝对值最大元素的列号
            k=j
    print(i,k,x[i][k])
```

- 例：矩阵乘法。已知m×n矩阵A和n×p矩阵B，试求它们乘积C=A×B。

程序运行结果如下：

```
Matrix A:
[2, 1]
[3, 5]
[1, 4]
Matrix B:
[3, 2, 1, 4]
[0, 7, 2, 6]
Matrix C:
[6, 11, 4, 14]
[9, 41, 13, 42]
[3, 30, 9, 28]
```

```
A=[[2,1],[3,5],[1,4]]
B=[[3,2,1,4],[0,7,2,6]]
C=[[0]*len(B[0]) for i in range(len(A))]  #定义矩阵C
for i in range(len(A)):
    for j in range(len(B[0])):
        t=0
        for k in range(len(B)):
            t+=A[i][k]*B[k][j]
        C[i][j]=t
print("Matrix A:")    #输出A矩阵
for i in range(len(A)):
    print(A[i])
print("Matrix B:")   #输出B矩阵
for i in range(len(B)):
    print(B[i])
print("Matrix C:")   #输出C矩阵
for i in range(len(C)):
    print(C[i])
```

- 例：找出一个二维数组中的鞍点，即该位置上的元素是该行上的最大值，是该列上的最小值。二维数组可能不止一个鞍点，也可能没有鞍点。

程序运行时，可以用以下两个矩阵验证程序。

（1）二维矩阵有鞍点。

```
 9    80   205   40
90   -60    96    1
210   -3   101   89
```

（2）二维矩阵没有鞍点。

```
 9    80   205   40
90   -60   196    1
210   -3   101   89
45    54   156    7
```

```
m,n=eval(input())                #输入矩阵的行数和列数
a=[[0]*n for i in range(m)]       #定义矩阵
for i in range(m):               #输入矩阵的值
    for j in range(n):
        a[i][j]=eval(input())
print("Matrix a:")               #输出矩阵的值
for i in range(len(a)):
    print(a[i])
flag2=0                          #flag2作为数组中是否有鞍点的标志
for i in range(len(a)):
    maxx=a[i][0]                 #求每一行最大元素及其所在列
    for j in range(len(a[0])):
        if a[i][j]>maxx:
            maxx=a[i][j]
            maxj=j
    k=0
    flag1=1                      #flag1作为行中的最大值是否有鞍点的标志
    while k<len(a) and flag1:
        if maxx>a[k][maxj]:      #判断行中的最大值是否也是列中的最小值
            flag1=0
        k+=1
    if flag1:
        print("第{}行第{}列的{}是鞍点!".format(i,maxj,maxx))
        flag2=1
if not flag2:
        print("矩阵无鞍点!")
```

- 在Pthon环境下还有专用的科学计算函数模块，如NumPy（Numeric Python）、SciPy等，有需求时可以下载使用。

自测题

- 一、选择题
- 1．访问字符串中的部分字符的操作称为（　　）。
- A．分片　　B．合并
- C．索引　　D．赋值

自测题

- 一、选择题
- 1．访问字符串中的部分字符的操作称为（　　）。

  A
- A．分片　　B．合并
- C．索引　　D．赋值

- 2．下列关于字符串的描述错误的是（　　）。
- A．字符串s的首字符是s[0]
- B．在字符串中，同一个字母的大小是等价的。
- C．字符串中的字符都是以某种二进制编码的方式进行存储和处理的
- D．字符串也能进行关系比较操作

- 2．下列关于字符串的描述错误的是（　　）。B
- A．字符串s的首字符是s[0]
- B．在字符串中，同一个字母的大小是等价的。
- C．字符串中的字符都是以某种二进制编码的方式进行存储和处理的
- D．字符串也能进行关系比较操作

- 3．执行下列语句后的显示结果是（　　）。
- world="world"
- print("hello"+world)
- A．helloworld　　B．"hello"world
- C．hello world　　D．"hello"+world

- 3．执行下列语句后的显示结果是（　　）。A
- world="world"
- print("hello"+world)
- A．helloworld　　B．"hello"world
- C．hello world　　D．"hello"+world

- 4．下列表达式中，有3个表达式的值相同，另一个不相同，与其他3个表达式不同的是（　　）。
- A．"ABC"+"DEF"　　B．''.join(("ABC","DEF"))
- C．"ABC"-"DEF"　　D．'ABCDEF'*1

- 4．下列表达式中，有3个表达式的值相同，另一个不相同，与其他3个表达式不同的是（　　）。C
- A．"ABC"+"DEF"　　B．".join(("ABC","DEF"))
- C．"ABC"-"DEF"　　D．'ABCDEF'*1

- 5．设s="Python Programming"，那么print(s[-5:])的结果是（　　）。
- A．mming　　B．Python
- C．mmin　　D．Pytho
- 6．设s="Happy New Year"，则s[3:8]的值为（　　）。
- A．'ppy Ne'　　B．'py Ne'
- C．'ppy N'　　D．'py New'

- 5．设s="Python Programming"，那么print(s[-5:])的结果是（　　）。A
- A．mming　　B．Python
- C．mmin　　D．Pytho
- 6．设s="Happy New Year"，则s[3:8]的值为（　　）。B
- A．'ppy Ne'　　B．'py Ne'
- C．'ppy N'　　D．'py New'

- 7．将字符串中全部字母转换为大写字母的字符串方法是（　　）。
- A．swapcase　　B．capitalize
- C．uppercase　　D．upper
- 8．下列表达式中，能用于判断字符串s1是否属于字符串s（即s1是否s的子串）的是（　　）。
- ①s1 in s；②s.find(s1)>0；③s.index(s1)>0；④s.rfind(s1)；⑤s.rindex(s1)>0
- A．①　　B．①②
- C．①②③　　D．①②③④⑤

- 7．将字符串中全部字母转换为大写字母的字符串方法是（　　）。D
- A．swapcase　　B．capitalize
- C．uppercase　　D．upper
- 8．下列表达式中，能用于判断字符串s1是否属于字符串s（即s1是否s的子串）的是（　　）。D
- ①s1 in s；②s.find(s1)>0；③s.index(s1)>0；④s.rfind(s1)；⑤s.rindex(s1)>0
- A．①　　B．①②
- C．①②③　　D．①②③④⑤

- 9． re.findall('to','Tom likes to play football too.',re.I) 的值是(　　)。
- A．['To', 'to', 'to']　　B．['to', 'to', 'to']
- C．['To', 'to']　　D．['to', 'to']

- 9． re.findall('to','Tom likes to play football too.',re.I)的值是(　　)。A
- A．['To', 'to', 'to']　　B．['to', 'to', 'to']
- C．['To', 'to']　　D．['to', 'to']

- 10．下列程序执行后，得到的输出结果是（　　）。
- import re
- p=re.compile(r'\bb\w*\b')
- str="Boys may be able to get a better idea."
- print(p.sub('**',str,1))
- A．** may be able to get a better idea.
- B．Boys may be able to get a ** idea.
- C．Boys may ** able to get a better idea.
- D．Boys may ** able to get a ** idea.

- 10．下列程序执行后，得到的输出结果是（　　）。

  C

- import re
- p=re.compile(r'\bb\w*\b')
- str="Boys may be able to get a better idea."
- print(p.sub('**',str,1))
- A．** may be able to get a better idea.
- B．Boys may be able to get a ** idea.
- C．Boys may ** able to get a better idea.
- D．Boys may ** able to get a ** idea.

- 二、填空题
- 1. "4"+"5"的值是（　）。
- 2. 字符串s中最后一个字符的位置是（　）。

- 二、填空题
- 1．"4"+"5"的值是（　）。'45'
- 2．字符串s中最后一个字符的位置是（　）。

  len(s)-1

- 3．设s='abcdefg'，则s[3]的值是（　），s[3:5]的值是（　），s[:5]的值是（　），s[3:]的值是（　），s[::2]的值是（　），s[::-1]的值是（　）,s[-2:-5]的值是（　）。

- 3．设s='abcdefg'，则s[3]的值是（　），s[3:5]的值是（　），s[:5]的值是（　），s[3:]的值是（　），s[::2]的值是（　），s[::-1]的值是（　）,s[-2:-5]的值是（　）。
- 'd'，'de'，'abcde'，'defg'，'aceg'，'gfedcba'，''

- 4．'Python Program'.count('P')的值是（　）。
- 5．'AsDf888'.isalpha()的值是（　）。
- 6．下面语句的执行结果是（　）。
- s='A'
- print(3*s.split())

- 4．'Python Program'.count('P')的值是（　）。2
- 5．'AsDf888'.isalpha()的值是（　）。False
- 6．下面语句的执行结果是（　）。['A', 'A', 'A']
- s='A'
- print(3*s.split())

- 7．已知s1='red hat'，print(s1.upper())的结果是（　），s1.swapcase()的结果是（　），s1.title()的结果是（　），s1.replace('hat','cat')的结果是（　）。

- 7．已知s1='red hat'，print(s1.upper())的结果是（　），s1.swapcase()的结果是（　），s1.title()的结果是（　），s1.replace('hat','cat')的结果是（　）。
- RED HAT，'RED HAT'，'Red Hat'，'red cat'

- 8．设s='a,b,c'，s2=('x','y','z')，s3=':'，则s.split(',')的值为（　），s.rsplit(',',1)的值为（　），s.partition(',')的值为（　），s.rpartition(',')的值为（　），s3.join('abc')的值为（　），s3.join(s2)的值为（　）。

- 8．设s='a,b,c'，s2=('x','y','z')，s3=':'，则s.split(',')的值为（　），s.rsplit(',',1)的值为（　），s.partition(',')的值为（　），s.rpartition(',')的值为（　），s3.join('abc')的值为（　），s3.join(s2)的值为（　）。
- ['a', 'b', 'c']，['a,b', 'c']，('a', ',', 'b,c')，('a,b', ',', 'c'),'a:b:c','x:y:z'

- 9．re.sub('hard','easy','Python is hard to learn.')的值是（　）。
- 10．下列程序执行后，得到的输出结果是（　）。
- import re
- str="An elite university devoted to computer software"
- print(re.findall(r'\b[aeiouAEIOU]\w+?\b',str))

- 9．re.sub('hard','easy','Python is hard to learn.')的值是（　）。
- 'Python is easy to learn.'
- 10．下列程序执行后，得到的输出结果是（　）。

['An', 'elite', 'university']

- import re
- str="An elite university devoted to computer software"
- print(re.findall(r'\b[aeiouAEIOU]\w+?\b',str))

- 三、问答题
- 1．什么叫字符串？有哪些常用的字符编码方案？
- 2．数字字符和数字值（如'5'和5）有何不同？如何转换？
- 3．为什么print('I like Python'* 5)可以正常执行，而print('I like Python'+5)却运行时出错？

- **4．写出表达式。**
- **（1）利用各种方法判断字符变量c是否为字母(不区分大小写字母)。**

- **4．写出表达式。**
- **（1）利用各种方法判断字符变量c是否为字母(不区分大小写字母)。**
- **c.isalpha()或c.lower()<='z' and c.lower()>='a'或c.upper()<='Z' and c.upper()>='A'或c<='Z' and c >='A'or c<='z'and c >='a'**

- （2）利用各种方法判断字符变量c是否为大写字母。
- （3）利用各种方法判断字符变量c是否为小写字母。
- （4）利用各种方法判断字符变量c是否为数字字符。

- （2）利用各种方法判断字符变量c是否为大写字母。
- c.isupper()或者c<='Z' and c>='A'
- （3）利用各种方法判断字符变量c是否为小写字母。
- c.islower()或者c<='z' and c>='a'
- （4）利用各种方法判断字符变量c是否为数字字符。
- c.isdigit()或c<='9' and c>='O'

- **5． re.match("back","text.back")与 re.search("back","text.back")的执行结果有何不同？**