Python 程序设计基础

Python Programming

列表的概念

- 程序往往需要存储和处理大量的数据。
- 列表可以用来存储和处理大量的数据。
- 在列表中，值可以是任何数据类型。列表中的值称为元素。因此列表是一个元素序列，既可以包含同类型的元素也可以包含不同类型的元素。
- 列表中的每个元素都对应一个从 0 开始的顺序编号，该顺序编号称为下标。通过下标可以访问列表中的某个元素。只有一个下标的列表称为一维列表，有两个或以上下标的列表称为多维列表。
- 列表的大小是可变的，可以根据元素的数量自动调整大小。

列表的概念

- **列表中的元素用逗号分隔并且由一对中括号（ [] ）括住。**

```
>>> list1 = []
>>> list1
[]
>>> list2 = [1, 2, 3]
>>> list2
[1, 2, 3]
>>> list3 = ["red", "green", "blue"]
>>> list3
['red', 'green', 'blue']
>>> list4 = [2, "three", 4.5]
>>> list4
[2, 'three', 4.5]
>>> list5 = ["one", 2.0, 5, [100, 200]]
>>> list5
['one', 2.0, 5, [100, 200]]
```

一个不包含任何元素的列表被称为空列表；
可以用空的中括号（ [] ）创建一个空列表。

一个列表在另一个列表中，称为嵌套列表。

列表的概念

- **还可以使用 list 内置函数来创建列表。**

```
>>> list1 = list()
>>> list1
[]
>>> list2 = list(range(1, 10))
>>> list2
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list3 = list("abcd")
>>> list3
['a', 'b', 'c', 'd']
```

list 函数将字符串分割成由单独的字符组成的字符列表。

- **还可以使用 random 模块中的 shuffle 函数随机排列列表中的元素。**

```
>>> from random import shuffle
>>> lst = [1, 3, 5, 7, 9]
>>> shuffle(lst)
>>> lst
[5, 9, 3, 7, 1]
```

列表的基本操作

- **通过下标访问列表中的元素。**

  lst = [1, 3, 5, 7, 9]

  - **任何整数表达式都可以用作下标。列表下标从 0 开始，列表 lst 的下标范围从 0 到 len(lst)-1 ，即 0 到 4 。**
  - **通过：列表变量名 [下标] 或列表字面量 [下标] ，来访问列表中的元素。**
  - **越界访问列表元素是常见的错误，会导致“ IndexError” 异常。**

```
>>> lst = [1, 3, 5, 7, 9]
>>> lst[0]
1
>>> lst[4]
9
>>> [1, 3, 5, 7, 9][2]
5
```

```
>>> lst[5]
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    lst[5]
IndexError: list index out of range
```

列表的基本操作

- **使用负数作为下标来引用相对于列表末端的位置。将列表长度和负数下标相加就可以得到实际的位置。**

```
>>> lst = [1, 3, 5, 7, 9]
>>> lst[-1]
9
>>> lst[-1 + len(lst)]
9
>>> lst[-3]
5
>>> lst[-3 + len(lst)]
5
```

- **列表是可变对象，可以直接修改列表中的元素值。**

```
>>> lst[0] = 111
>>> lst
[111, 3, 5, 7, 9]
```

列表的基本操作

- **通过切片操作获得列表的子列表。**
  - **列表变量名 [start:end:step] 或列表字面量 [start:end:step] ，默认情况下 step 为 1 ，返回下标从 start 到 end-1 的元素构成的一个子列表。**

```
>>> lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> lst
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> lst[2:4]
[3, 4]
>>> lst[0:10:2]
[1, 3, 5, 7, 9]
```

列表的基本操作

- **start 和 end 可以省略。若省略 start ， start 默认为 0 ；若省略 end ， end 默认为列表长度；若 start 和 end 都省略，则切片就是整个列表的一个拷贝。**

```
>>> lst = [2, 4, 6, 8, 10]
>>> lst[:2]
[2, 4]
>>> lst[0:2]
[2, 4]
>>> lst[3:]
[8, 10]
>>> lst[3:len(lst)]
[8, 10]
>>> lst[:]
[2, 4, 6, 8, 10]
>>> lst[::2]
[2, 6, 10]
```

列表的基本操作

- 若 start 大于或等于 end ，将返回一个空列表。若 end 指定了一个超出列表末尾的位置，将使用列表长度替代 end 。

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> lst[3:3]
[]
>>> lst[2:10]
[5, 2, 33, 21]
>>> lst[2:len(lst)]
[5, 2, 33, 21]
```

- 切片也可以使用负数下标。

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> lst[1:-3]
[3, 5]
>>> lst[-4:-2]
[5, 2]
```

列表的基本操作

- **下面的切片操作将列表反转（逆序）。**

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> lst
[2, 3, 5, 2, 33, 21]
>>> lst[::-1]
[21, 33, 2, 5, 3, 2]
```

- **切片操作放在赋值运算符的左边时，可以一次更新列表中的多个元素。**

```
>>> lst = list("abcdef")
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
>>> lst[1:3] = ['x', 'y']
>>> lst
['a', 'x', 'y', 'd', 'e', 'f']
```

列表的基本操作

- **运算符。**
  - **使用 + 运算符来连接两个列表。**

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [4, 5, 6]
>>> lst = lst1 + lst2
>>> lst
[1, 2, 3, 4, 5, 6]
```

  - **使用 * 运算符以给定的次数重复一个列表。**

```
>>> [0] * 5
[0, 0, 0, 0, 0]
>>> 5 * [0]
[0, 0, 0, 0, 0]
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
>>> 2 * [1, 2, 3]
[1, 2, 3, 1, 2, 3]
```

列表的基本操作

- **使用 in 或 not in 运算符来判断元素是否在列表中。**

```
>>> lst = ["C++", "Java", "Python"]
>>> "Python" in lst
True
>>> "Java" not in lst
False
```

- **使用 is 或 is not 来判断两个列表是否是同一个对象。**

```
>>> lst1 = ["C++", "Java", "Python"]
>>> lst2 = ["C++", "Java", "Python"]
>>> id(lst1)
2497971114120
>>> id(lst2)
2497969974600
>>> lst1 is lst2
False
```

列表的基本操作

- **可以使用关系运算符对列表进行比较。进行比较的两个列表必须包含相同类型的元素。对于字符串列表比较使用的是字典顺序。**

```
>>> lst1 = [1, 2, 3, 7, 9, 0, 5]
>>> lst2 = [1, 3, 2, 7, 9, 0, 5]
>>> lst1 > lst2
False
>>> lst3 = ["red", "green", "blue"]
>>> lst4 = ["red", "blue", "green"]
>>> lst3 > lst4
True
```

列表的基本操作

- **遍历列表。**
  - **使用 for 语句**

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> for value in lst:
        print(value, end=' ')


2 3 5 2 33 21
```

  - **使用 for 语句，结合内置函数 range 和 len ，通过下标访问列表中的元素。**

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> for i in range(len(lst)):
        print(lst[i], end=' ')


2 3 5 2 33 21
```

列表的基本操作

- **如果想要写入或者更新列表中的元素，只能通过下标访问。**

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> for i in range(len(lst)):
        lst[i] = lst[i] * 2
>>> print(lst)
[4, 6, 10, 4, 66, 42]
```

- **遍历列表时，既要输出元素又要输出该元素对应的下标，可以使用 enumerate 函数。每次循环， enumerate 函数返回一个包含下标和元素值的元组。**

```
>>> lst = [2, 3, 5, 2, 33, 21]
>>> for index, value in enumerate(lst):
        print(index, value)
```

```
0 2
1 3
2 5
3 2
4 33
5 21
```

列表的基本操作

- **尽管一个列表可以包含另一个列表，嵌套的列表本身还是被看作单个元素。下面这个列表的长度是 4 。**

```
>>> lst = ["Hello", 1, ["Java", "Python", "C++"], [3, 4, 5]]
>>> len(lst)
4
```

列表的基本操作

- **列表解析。**
  - **列表解析提供了一种创建列表的简洁方式。**
  - **一个列表解析由方括号组成。方括号内包含后跟一个 for 子句的表达式，之后是 0 或多个 for 子句或 if 子句。列表解析产生一个由表达式求值结果组成的列表。**

    [expr for iter_var in iterable]

    首先循环 iterable 里所有内容，每一次循环，都把 iterable 里相应内容放到 iter_var 中，再在 expr 中应用该 iter_var 的内容，最后用 expr 的计算值生成一个列表。

    [expr for iter_var in iterable if cond_expr]

    加入了判断语句，只有满足条件的才把 iterable 里相应内容放到 iter_var 中，再在 expr 中应用该 iter_var 的内容，最后用 expr 的计算值生成一个列表。

列表的基本操作

```
>>> lst1 = [value for value in range(0, 11, 2)]
>>> print(lst1)
[0, 2, 4, 6, 8, 10]
>>> lst2 = [0.5 * value for value in lst1]
>>> print(lst2)
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
>>> lst3 = [value for value in lst2 if value > 2.0]
>>> print(lst3)
[3.0, 4.0, 5.0]
```

列表的基本操作

- **向列表添加元素。**
  - **lst.append(x) 方法，将元素 x 添加到列表 lst 的末尾。等价于 lst[len(lst):]=[x] 。**

```
>>> lst = ["C++", "Java", "Python"]
>>> print(lst)
['C++', 'Java', 'Python']
>>> lst.append("C#")
>>> print(lst)
['C++', 'Java', 'Python', 'C#']
>>> lst[len(lst):] = ["JavaScript"]
>>> print(lst)
['C++', 'Java', 'Python', 'C#', 'JavaScript']
```

列表的基本操作

- lst.extend(lst2) 方法，将列表 lst2 的所有元素追加到列表 lst 的末尾。追加后，列表 lst2 的内容保持不变。等价于 lst[len(lst):]=lst2 。

```
>>> lst = ["C++", "Java", "Python"]
>>> print(lst)
['C++', 'Java', 'Python']
>>> lst2 = ["C#", "JavaScript"]
>>> print(lst2)
['C#', 'JavaScript']
>>> lst.extend(lst2)
>>> print(lst)
['C++', 'Java', 'Python', 'C#', 'JavaScript']
>>> print(lst2)
['C#', 'JavaScript']
```

列表的基本操作

- **lst.insert(index, x) 方法，将元素 x 插入到列表 lst 中 index 下标处。**

```
>>> lst = [-22, 3, 45, 11, 62, 38]
>>> print(lst)
[-22, 3, 45, 11, 62, 38]
>>> lst.insert(0, 55)              # 在列表头部插入元素
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38]
>>> lst.insert(len(lst), 985)      # 在列表尾部插入元素
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38, 985]
>>> lst.insert(10, 211)            # 插入位置超过最大下标值，在列表尾部插入元素
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38, 985, 211]
```

列表的基本操作

- **从列表删除元素。**
  - **lst.pop(index) 方法，删除列表 lst 中 index 下标处的元素，并返回该元素。 index 是可选的，若没有指定 index ，则删除并返回列表 lst 中的最后一个元素。**

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 211]
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38, 985, 211]
>>> x = lst.pop(1)
>>> print(lst)
[55, 3, 45, 11, 62, 38, 985, 211]
>>> print(x)
-22
>>> x = lst.pop()
>>> print(lst)
[55, 3, 45, 11, 62, 38, 985]
>>> print(x)
211
```

列表的基本操作

- **lst.remove(x) 方法，删除列表 lst 中元素 x 的第一个匹配项，无返回值。若列表 lst 中元素 x 不存在，则抛出“ ValueError” 异常。**

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst.remove(11)
>>> print(lst)
[55, -22, 3, 45, 62, 38, 985, 11]
>>> lst.remove(88)
Traceback (most recent call last):
  File "<pyshell#172>", line 1, in <module>
    lst.remove(88)
ValueError: list.remove(x): x not in list
```

列表的基本操作

- **del 语句可以删除列表中一个或多个元素，或删除整个列表。**

```
>>> lst = ['a', 'b', 'c', 'd', 'e', 'f']
>>> print(lst)
['a', 'b', 'c', 'd', 'e', 'f']
>>> del lst[1]
>>> print(lst)
['a', 'c', 'd', 'e', 'f']
>>> del lst[1:4]
>>> print(lst)
['a', 'f']
>>> del lst                  # 删除整个列表
>>> print(lst)               # 错误，列表不存在了
Traceback (most recent call last):
  File "<pyshell#181>", line 1, in <module>
    print(lst)              # 错误，列表不存在了
NameError: name 'lst' is not defined
```

列表的基本操作

- **其他常用列表方法。**
  - **lst.count(x) 方法，返回元素 x 在列表 lst 中的出现次数。**
  - **lst.index(x) 方法，返回元素 x 在列表 lst 中第一次出现的位置下标。若列表 lst 中元素 x 不存在，则抛出“ ValueError” 异常。**
  - **lst.clear() 方法，删除列表 lst 中的所有元素。**

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst.count(11)
2
>>> lst.count(8)
0
>>> lst.index(11)
4
>>> lst.index(8)
Traceback (most recent call last):
  File "<pyshell#191>", line 1, in <module>
    lst.index(8)
ValueError: 8 is not in list
```

```
>>> lst.clear()
>>> lst
[]
```

列表的基本操作

- **逆序和排序列表。**
  - **lst[::-1] 可以将列表 lst 中的所有元素逆序。**
  - **使用 lst.reverse() 方法，也可以将列表 lst 中的所有元素逆序。**

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst
[55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst.reverse()
>>> lst
[11, 985, 38, 62, 11, 45, 3, -22, 55]
```

列表的基本操作

- reversed(seq) 函数是 Python 提供的内置函数。使用 reversed 函数可以将列表 seq 中的所有元素逆序，返回由逆序后的所有元素构成的一个可迭代对象，原列表 seq 保持不变。使用 list 函数可以将可迭代对象转换为一个列表并返回该列表。

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst
[55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> r_lst = reversed(lst)
>>> r_lst
<list_reverseiterator object at 0x000002459AB02470>
>>> list(r_lst)
[11, 985, 38, 62, 11, 45, 3, -22, 55]
>>> lst
[55, -22, 3, 45, 11, 62, 38, 985, 11]
```

- reverse 方法只能对列表进行逆序， reversed 函数则可以对列表、字符串、元组等进行逆序。

列表的基本操作

- **lst.sort(key=None, reverse=False) 方法，对列表 lst 中的所有元素升序（默认）或降序（ reverse 参数为 True ）排序。若 key 参数为一个函数名，则按该函数指定的规则进行排序。**

```
>>> lst = [55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> print(lst)
[55, -22, 3, 45, 11, 62, 38, 985, 11]
>>> lst.sort()                  # 升序排序
>>> print(lst)
[-22, 3, 11, 11, 38, 45, 55, 62, 985]
>>> lst.sort(reverse=True)      # 降序排序
>>> print(lst)
[985, 62, 55, 45, 38, 11, 11, 3, -22]
```

```
>>> lst = ["C++", "Java", "Python", "C#", "JavaScript"]
>>> print(lst)
['C++', 'Java', 'Python', 'C#', 'JavaScript']
>>> lst.sort(key=len)
>>> print(lst)
['C#', 'C++', 'Java', 'Python', 'JavaScript']
```

按字符串长度升序排序。 len 是 Python 的内置函数。

列表的基本操作

- sorted(seq,key=None, reverse=False) 函数是 Python 提供的内置函数。使用 sorted 函数可以将列表 seq 中的所有元素升序（默认）或降序（ reverse 参数为 True ）排序。若 key 参数为一个函数名，则按该函数指定的规则进行排序。返回由排序后的所有元素构成的一个列表，原列表 seq 保持不变。

```
>>> lst = ["C++", "Java", "Python", "C#", "JavaScript"]
>>> print(lst)
['C++', 'Java', 'Python', 'C#', 'JavaScript']
>>> print(sorted(lst))
['C#', 'C++', 'Java', 'JavaScript', 'Python']
>>> print(sorted(lst, reverse=True))
['Python', 'JavaScript', 'Java', 'C++', 'C#']
>>> print(sorted(lst, key=len))
['C#', 'C++', 'Java', 'Python', 'JavaScript']
>>> print(lst)
['C++', 'Java', 'Python', 'C#', 'JavaScript']
```

- sort 方法仅对列表进行排序， sorted 函数可以对列表、字符串、元组等进行排序。

列表的基本操作

- **自定义排序函数。**

```
# 自定义排序函数，按单词包含的元音字母个数排序，单词均小写
def vowels(word):
    vowels = ['a', 'e', 'i', 'o', 'u']
    total = 0
    for vowel in vowels:
        total += word.count(vowel)
    return total

def main():
    lst1 = ["democratic", "sequoia", "equals", "brrr", "break", "two"]
    print(lst1)
    lst2 = sorted(lst1, key=lambda x: x[-1])    # 使用lambda表达式，按单词最后一个字母排序
    print(lst2)
    lst3 = sorted(lst1, key=vowels)             # 使用自定义排序函数
    print(lst3)

main()
```

```
['democratic', 'sequoia', 'equals', 'brrr', 'break', 'two']
['sequoia', 'democratic', 'break', 'two', 'brrr', 'equals']
['brrr', 'two', 'break', 'equals', 'democratic', 'sequoia']
```

例子

- 一组数据按从小到大的顺序依次排列，位于中间位置的一个数或位于中间位置的两个数的平均值被称为中位数。如果这组数的个数为奇数，则中位数是位于中间位置的数；如果这组数的个数为偶数，则中位数是位于中间位置的两个数的平均值）。编写程序，给出一组无序整数，整数间以空格分隔，求出中位数。如果求中间位置的两个数的平均值，向下取整即可。
  - 通过 input 函数读取一个字符串，使用 split 方法，将字符串中的内容分解成列表，通过列表解析得到整数列表，对整数列表进行升序排序。

```
line = input().split()
lst = [eval(value) for value in line]
lst.sort()
n = len(lst)
if n % 2 == 0:
    print((lst[n // 2 - 1] + lst[n // 2]) // 2)
else:
    print(lst[n // 2])
```

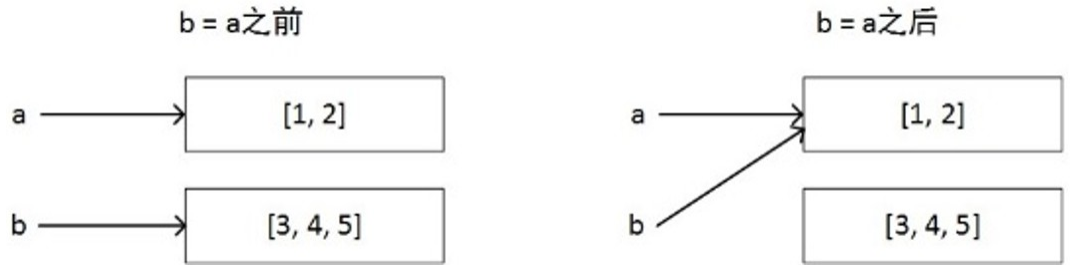
```
10 30 20 40
25

40 30 50
40
```

复制列表

- **变量和对象之间的关联称为引用。**
  - **创建一个对象并把它赋给一个变量，建立了变量对对象的引用；再将变量赋给另一个变量，建立了第二个变量对对象的引用。两个变量共享引用同一个对象。**

```
>>> a = [1, 2]
>>> b = [3, 4, 5]
>>> id(a)
2497971244168
>>> id(b)
2497969974728
>>> b = a
>>> id(b)
2497971244168
>>> a is b
True
```



  - **b=a 之后， b 之前指向的列表将不再被引用，它就变成了垃圾，所占用的内存空间将由 Python 自动回收并重新使用。**

复制列表

- **若一个对象有多个引用，那它也会有多个名称，称这个对象是有别名的。如果一个有别名的对象是可变的，对其中一个别名的改变会影响到其他的别名。**

```
>>> a = [1, 2, 3, 4]
>>> b = a
>>> a is b
True
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3, 4]
>>> a[0] = 111
>>> a
[111, 2, 3, 4]
>>> b
[111, 2, 3, 4]
>>>
```

两（多）个变量共享引用同一个对象会引发关联性问题，容易导致出现错误。

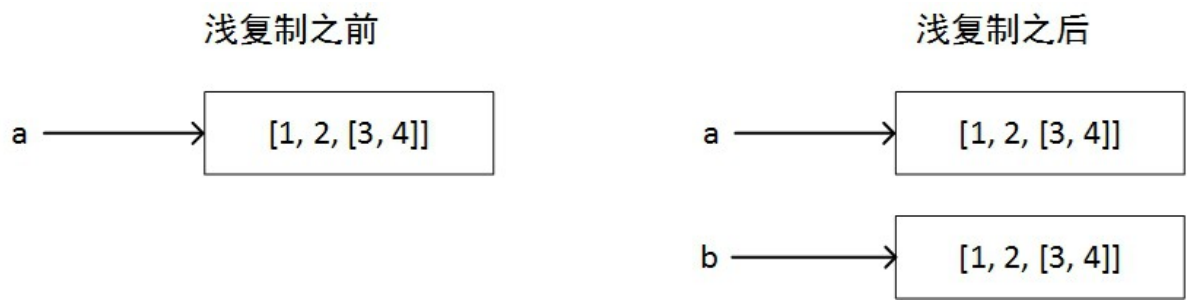
通常，对于可变对象，要尽量避免使用别名。

对于像字符串这样的不可变对象，使用别名没有什么问题。

复制列表

- **为避免可变对象之间使用“=”赋值存在的关联性问题。使用“浅复制”。**
  - **浅复制会创建一个新的对象，并将原始对象中的元素逐个复制过去**

```
>>> a = [1, 2, [3, 4]]
>>> b = [value for value in a]          # 使用列表解析进行浅复制
>>> a is b
False
>>> a[0] = 111                          # 更改列表a首元素的值
>>> a
[111, 2, [3, 4]]
>>> b                                   # 对列表b没有任何影响
[1, 2, [3, 4]]
```



复制列表

```
>>> a = [1, 2, [3, 4]]
>>> b = [] + a                    # 使用运算符+与空列表连接进行浅复制
>>> a is b
False

>>> a = [1, 2, [3, 4]]
>>> b = list(a)                   # 使用list函数进行浅复制
>>> a is b
False

>>> a = [1, 2, [3, 4]]
>>> b = a[:]                      # 使用列表切片操作进行浅复制
>>> a is b
False

>>> a = [1, 2, [3, 4]]
>>> b = a.copy()                  # 使用copy方法进行浅复制
>>> a is b
False
```

复制列表

- **深复制。**
  - **浅复制存在的问题。**

```
>>> a = [1, 2, [3, 4]]
>>> b = a[:]                    # 浅复制
>>> a[2][0] = 777               # 更改列表a中嵌套列表的首元素值
>>> a
[1, 2, [777, 4]]
>>> b
[1, 2, [777, 4]]
```

  - **这里将列表 a 第 2 个元素的第 0 个元素值由 3 改为 777 ，列表 b 发生了关联性变动。**

复制列表

- **使用 copy 模块中的 deepcopy 方法进行“深复制”，避免浅复制存在的问题。**

```
>>> from copy import deepcopy
>>> a = [1, 2, [3, 4]]
>>> b = deepcopy(a)                    # 深复制
>>> a[2][0] = 777                      # 更改列表a中嵌套列表的首元素值
>>> a
[1, 2, [777, 4]]
>>> b
[1, 2, [3, 4]]
```

列表和函数

- 将一个列表作为实参传递给一个函数，函数将得到这个列表的一个引用。由于列表是可变对象，如果函数对这个列表进行了修改，那么这个列表的内容会在函数调用后改变。

```
def delete_list_head(lst):
    del lst[0]    # 修改列表

digits = [1, 2, 3]
delete_list_head(digits)
print(digits)
```

```
[2, 3]
```

- 形参 lst 和实参 digits 是同一个对象的别名。

列表和函数

- **修改列表操作和创建列表操作之间的区别**

```
def delete_list_head(lst):
    lst = lst[1:]    # 创建列表

digits = [1, 2, 3]
delete_list_head(digits)
print(digits)
```

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

  - **在 delete_list_head 函数的开始处，形参 lst 和实参 digits 指向同一个列表。切片操作 lst[1:] 创建了一个新列表， lst 指向该新列表，但是 digits 仍然指向原来的列表。**

列表和函数

- **函数可以返回一个列表。**
  - **当函数返回一个列表时，实际返回的是该列表的引用。**

```
def delete_list_head(lst):
    return lst[1:]    # 创建并返回一个新列表

digits = [1, 2, 3]
rest = delete_list_head(digits)
print(rest)
```

```
[2, 3]
```

列表和函数

列表作为函数默认参数

```
def add_element(value, lst=[]):
    if value not in lst:
        lst.append(value)
    return lst

def main():
    print(add_element(1))
    print(add_element(2))
    print(add_element(3, [-1, -2, -3, -4]))
    print(add_element(4))

main()
```

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

第一次调用 add_element 函数，参数 lst 使用默认值 []，这个默认值只会被创建一次。1 添加到 lst 中，lst 为 [1]。

第二次调用 add_element 函数，参数 lst 使用默认值 [1] 而不是 []，2 添加到 lst 中，lst 为 [1, 2]。

第三次调用 add_element 函数，给出了列表实参，参数 lst 是 [-1, -2, -3, -4] 而不是默认值 [1, 2]，3 添加到 lst 中，lst 为 [-1, -2, -3, -4, 3]

第四次调用 add_element 函数，参数 lst 使用默认值 [1, 2] 而不是 []，4 添加到 lst 中，lst 为 [1, 2, 4]。

列表和函数

- 如果想要默认参数在每次函数调用时都是 [] ，可以使用 None

```
def add_element(value, lst=None):
    if not lst:
        lst = []
    if value not in lst:
        lst.append(value)
    return lst

def main():
    print(add_element(1))
    print(add_element(2))
    print(add_element(3, [-1, -2, -3, -4]))
    print(add_element(4))

main()
```

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

每次调用 add_element 函数且没有给定列表实参时，都会使用默认值 [] 。

如果调用 add_element 函数时给定了列表实参，就不会使用默认参数。

二维列表

- **二维列表，即列表中的每个元素又是另一个列表，也可以认为是嵌套列表。**

```
>>> multilist = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
>>> multilist
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

  - **multilist 是 3 行 3 列的二维列表。**

```
>>> multilist = [[0 for column in range(3)] for row in range(4)]
>>> multilist
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

  - **multilist 是 4 行 3 列的二维列表。**

二维列表

- **二维列表可以理解为一个由行组成的列表。二维列表中的每个值都可以用：列表变量名 [行下标][列下标] 来访问。注意：行下标和列下标都是从 0 开始。**

```
>>> multilist = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> multilist[0]                    # 二维列表第1行
[1, 2, 3]
>>> multilist[1]                    # 二维列表第2行
[4, 5, 6]
>>> multilist[2]                    # 二维列表第3行
[7, 8, 9]
>>> multilist[0][0]                 # 二维列表首元素的值
1
>>> multilist[2][2]                 # 二维列表末元素的值
9
```

二维列表

- **编写程序，创建一个二维列表，元素值是随机的两位正整数，输出该二维列表以及所有元素的和。**

```
m = []
rows = eval(input("请输入二维列表的行数: "))
columns = eval(input("请输入二维列表的列数: "))
for row in range(rows):
    m.append([])
    line = [eval(value) for value in input().split()]
    for column in range(columns):
        m[row].append(line[column])
for row in m:
    for value in row:
        print(value, end = ' ')
    print()
total = 0
for row in m:
    total += sum(row)
print(total)
```

```
请输入二维列表的行数: 3
请输入二维列表的列数: 3
1 2 3
4 5 6
7 8 9
1 2 3
4 5 6
7 8 9
45
```

二维列表

- 可以使用 sort 方法或 sorted 函数对二维列表进行排序。按行排序，即按每一行的第一个元素进行排序；对于第一个元素相同的行，则按它们的第二个元素进行排序；若行中第一个元素和第二个元素都相同，则按它们的第三个元素进行排序，以此类推。

```
>>> multilist = [[4, 2, 1], [1, 6, 7], [4, 5, 6], [1, 2, 3], [4, 2, 8]]
>>> multilist.sort()
>>> multilist
[[1, 2, 3], [1, 6, 7], [4, 2, 1], [4, 2, 8], [4, 5, 6]]
```