

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.1.1 创建列表

列表 (Lists) 属于Python中的序列类型，它是任意对象的有序集合，通过“位置”或者“索引”访问其中的元素，它具有可变对象、可变长度、异构和任意嵌套的特点。

列表里第一个元素的“位置”或者“索引”是从“0”开始，第二个元素的则是“1”，以此类推。

在创建列表时，列表元素放置在方括号[] 中，以逗号来分隔各元素，格式如下：

listname = [元素1, 元素2, 元素3,, 元素n]

举例如下：

```
sample_list0 = [] # 空列表，等价于 sample_list0 = list()
```

```
sample_list1 = [0, 1, 2, 3, 4]
```

```
sample_list2 = ["P", "y", "t", "h", "o", "n"]
```

```
sample_list3 = ['Python', 'sample', 'list', 'for', 'your', 'reference']
```

4.1.1 创建列表

代码运行如下：

```
>>> sample_list1 = [0, 1, 2, 3, 4]          #列表sample_list1
>>> sample_list2 = ["P", "y", "t", "h", "o", "n"] #列表sample_list2
>>> sample_list3 = ['Python', 'sample', 'list', 'for', 'your', 'reference']
#列表sample_list3
>>> print (sample_list1)                    #打印输出列表
[0, 1, 2, 3, 4]                             #输出结果
>>> print (sample_list2)                    #打印输出列表
['p', 'y', 't', 'h', 'o', 'n']             #输出结果
>>> print (sample_list3)                    #打印输出列表
['Python', 'sample', 'list', 'for', 'your', 'reference'] #输出结果
```

4.1.1 创建列表

列表中允许有不同数据类型的元素，例如：

```
sample_list4 = [0, "y", 2, "h", 4, "n", 'Python']
```

但通常建议列表中元素最好使用相同的数据类型。

列表可以嵌套使用，例如：

```
sample_list5 = [sample_list1, sample_list2, sample_list3]
```

运行结果如下：

```
>>> sample_list1 = [0, 1, 2, 3, 4]
```

```
>>> sample_list2 = ["P", "y", "t", "h", "o", "n"]
```

```
>>> sample_list3 = ['Python', 'sample', 'list', 'for', 'your', 'reference']
```

```
>>> sample_list5 = [sample_list1, sample_list2, sample_list3] #创建一个嵌套列表
```

```
>>> print (sample_list5)
```

```
[[0, 1, 2, 3, 4], ['P', 'y', 't', 'h', 'o', 'n'], ['Python', 'sample', 'list', 'for', 'your', 'reference']]
```

4.1.2 使用列表

通过使用“位置”或者“索引”来访问列表中的值，将放在方括号中。特别注意，“位置”或者“索引”是从0开始，例如引用上一节列表示例 sample_list1 中的第1个，可以写成：sample_list1[0]；引用第3个值，可以写成：sample_list1[2]。

代码示例为：

```
>>> sample_list1 = [0, 1, 2, 3, 4]
```

```
>>> print(sample_list1[0]) #输出索引为0的元素
```

```
0
```

```
>>> print(sample_list1[2]) #输出索引为2的元素
```

```
2
```

```
>>> print(sample_list5[1]) #输出嵌套列表中索引为1的"元素"——一个列表  
['P', 'y', 't', 'h', 'o', 'n']
```

```
>>> print(sample_list5[1][0]) #输出索引为1的列表中的第1个元素
```

```
P
```

4.1.2 使用列表

可以在方括号中使用 **“负整数”**，如：sample_list1[-2]，意为从列表的右侧开始倒数2个的元素，即索引倒数第2的元素。

```
>>> sample_list1 = [0, 1, 2, 3, 4]
```

```
>>> print ("sample_list1[-2]: ", sample_list1[-2])#输出索引倒数第2的元素  
sample_list1[-2]: 3
```

a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

	← 从左往右					从右往左 →				
正索引	0	1	2	3	4	5	6	7	8	9
负索引	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
值	0	1	2	3	4	5	6	7	8	9
	↑ 起点									↑ 终点

b=['p', 'y', 't', 'h', 'o', 'n']

	← 反向递减序号					
负索引	-6	-5	-4	-3	-2	-1
	'p'	'y'	't'	'h'	'o'	'n'
正索引	0	1	2	3	4	5
	→ 正向递增序号					

4.1.2 使用列表

切片操作

在Python中处理列表的部分元素，称之为**切片 (slice)**。创建切片，可指定要访问的第一个元素、最后一个元素的索引及其步长，三个参数用冒号分隔。

`listname[起始索引: 终止索引: step]`

步长可正可负，缺省则为1。**step**为正则切片从左向右取，**step**为负则从右向左取。

例如**b[1:4]**，可取得列表**b**中的第2个~第4个元素，不包含第5个元素。

```
>>> b = ["p", "y", "t", "h", "o", "n"]
```

```
>>> print ("b[1:4]:", b[1:4])
```

```
b[1:4]: ['y', 't', 'h']
```

指定切边步长

```
b[1:4] <=> b[1:4:1] <=> b[-5:-2] <=> b[-5:-2:1]
```

```
>>> b[1:6:2]    #间隔为2
```

```
['y', 'h', 'n']
```

```
>>> b[-1:-4:-1]  #步长为负数，逆序取切片
```

```
['n', 'o', 'h']
```

-6	-5	-4	-3	-2	-1
'p'	'y'	't'	'h'	'o'	'n'
0	1	2	3	4	5

4.1.2 使用列表

切片操作

`listname[起始索引: 终止索引: step]`

-6	-5	-4	-3	-2	-1
'p'	'y'	't'	'h'	'o'	'n'
0	1	2	3	4	5

- **step为正时**: 若没有指定起始索引, 则默认为0; 若没有指定终止索引, 即按step顺序取到最后一个元素。

```

b[1:] # 从第2个元素开始的所有元素 ['y', 't', 'h', 'o', 'n']
#<=> b[1:len(b)] <=> b[1:len(b):1] <=> b[1:len(b):]

b[-2:] # 后两个元素 ['o', 'n']
#<=> b[-2:-1] <=> b[-2:-1:1] <=> b[-2:-1:]

b[:2] # 前两个元素 ['p', 'y']
#<=> b[0:2] <=> b[0:2:1] <=> b[0:2:]

b[:] # 从左向右所有元素 ['p', 'y', 't', 'h', 'o', 'n']
#<=> b <=> b[0:len(b):1] <=> b[0:len(b):]

b[:] #<=> b[::1] <=> b[0:len(b):1] <=> b ['p', 'y', 't', 'h', 'o', 'n']
  
```


4.1.2 使用列表

切片操作

`listname[起始索引: 终止索引: step]`

-6	-5	-4	-3	-2	-1
'p'	'y'	't'	'h'	'o'	'n'
0	1	2	3	4	5

- **step为负时**：若没有指定起始索引，则默认为-1；若没有指定终止索引，即按step逆序取到最后一个元素。

`b[:-5:-1]` # 索引号-1~-5, 注意不包括-5, 逆序取元素 ['n', 'o', 'h', 't']

#<=> `b[-1:-5:-1]`

`b[-2::-1]` # 索引号-2~-7, 逆序取元素 ['o', 'h', 't', 'y', 'p']

#<=> `b[-2:-len(b)-1:-1]`

`b[::-1]` # 逆序取所有元素 ['n', 'o', 'h', 't', 'y', 'p']

#<=> `b[-1:-len(b)-1:-1]` <=> `b[:-len(b)-1:-1]` <=> `b[-1::-1]`

注意: `b[-2:-len(b):-1]`中不包括索引号为`-len(b)`的元素。

['o', 'h', 't', 'y']

4.1.2 使用列表

对列表的元素进行修改时，可以使用赋值语句：

```
>>> sample_list3 = ['python', 'sample', 'list', 'for', 'your', 'reference']
>>> sample_list3[4] = 'my'
>>> print ("sample_list3[4]:", sample_list3[4])
sample_list3[4]: my
>>> print ("sample_list3:", sample_list3)
sample_list3: ['python', 'sample', 'list', 'for', 'my', 'reference']
```

4.1.3 删除列表元素

删除列表的元素或切片，可以使用 **del 语句**，格式为：

```
del listname[索引]
```

```
del listname[起始索引 : 终止索引]
```

该索引的元素被删除后，后面的元素将会自动移动并填补该位置。

在不知道或不关心元素的索引时，可以使用列表内置方法**remove()**来删除指定的值，例如：

```
listname.remove(x)
```

清空列表，可以调用列表对象的**clear()**方法，也可以采用重新创建一个与原列表名相同的空列表的方法，例如：

```
listname.clear()
```

```
或者：listname = []
```

删除整个列表，也可以使用del语句，格式为：

```
del listname
```

4.1.3 删除列表元素

代码示例如下：

```
>>> sample_list4 = [0, "y", 2, "h", 4, "n", 'Python']
>>> del sample_list4[5]          #删除列表中索引为5的元素
>>> print ("after deletion, sample_list4: ", sample_list4)
after deletion, sample_list4: [0, 'y', 2, 'h', 4, 'Python']
>>> sample_list4.remove('Python') #删除列表中值为Python的元素
>>> print ("after removing, sample_list4: ", sample_list4)
after removing, sample_list4: [0, 'y', 2, 'h', 4]
>>> sample_list4 = []           #重新创建列表并置为空
>>> print (sample_list4)        #输出该列表
[]
>>> del sample_list4            #删除整个列表
>>> print (sample_list4)        #打印输出整个列表
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    print (sample_list4)
NameError: name 'sample_list4' is not defined #系统报告该列表未定义
```

4.1.3 删除列表元素

如何移除列表所有满足条件的元素?

例如, 删除a中所有小于2的元素 `a=[1, 2, 0, 3, 0, 0, 4, 5, 0, 0]`

```
for x in a:  
    if x<2:  
        a.remove(x) # 移除第1个x
```

`a = [2, 3, 4, 5, 0, 0]`



```
for i in range(len(a)):  
    if a[i]<2:  
        a.pop(i)
```

`a = [2, 3, 0, 4, 5, 0]`



Traceback (most recent call last):
File "<input>", line 2, in <module>
IndexError: list index out of range

请通过调试观察x与a的变化

4.1.3 删除列表元素

如何移除列表所有满足条件的元素？

```
Python 控制台 ×  
1 [1, 2, 0, 3, 0, 0, 4, 5, 0, 0]  
0 [2, 0, 3, 0, 0, 4, 5, 0, 0]  
0 [2, 3, 0, 0, 4, 5, 0, 0]  
4 [2, 3, 0, 4, 5, 0, 0]  
5 [2, 3, 0, 4, 5, 0, 0]  
0 [2, 3, 0, 4, 5, 0, 0]  
>>> a  
[2, 3, 4, 5, 0, 0]
```

最后一次删除的是第一个0



4.1.3 删除列表元素

如何移除列表所有满足条件的元素？

例如，删除a中所有小于2的元素 $a=[1, 2, 0, 3, 0, 0, 4, 5, 0, 0]$

```
i = 0
while True:
    if a[i] < 2:
        del a[i] # 或者 a.pop(i)
    else:
        i += 1
    if i == len(a):
        break
```

繁琐，但高效

$a = [2, 3, 4, 5]$



```
b = []
for x in a:
    if x >= 2:
        b.append(x)
a = b
```

简洁，但低效

$a = [2, 3, 4, 5]$



4.1.4 连接列表和重复列表

列表对 + 和 * 的操作符与字符串相似。+ 号用于连接列表，* 号用于重复列表。代码示例如下：

```
>>> [1, 2, 3] + [4, 5, 6]           #连接列表
[1, 2, 3, 4, 5, 6]
>>> [1,2,3]*3                       #重复列表
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

注意，列表对 + 和 * 的操作都返回一个新的列表，不会改变原有列表。而列表的extend或append方法则改变列表，例如：

```
>>> a= [1, 2, 3]; b=[4, 5, 6]
>>> a.extend(b) # 扩展列表a, b中的元素追加到a中
>>> a
[1, 2, 3, 4, 5, 6]
>>> a.append(7) # 7追加到a的末尾
>>> a
[1, 2, 3, 4, 5, 6, 7]
>>> a.append([8,9]) # [8,9]作为一个元素追加到a的末尾
>>> a
[1, 2, 3, 4, 5, 6, 7, [8, 9]]
```


4.1.5 列表的内置函数与其他方法

函数	说明
len(listname)	返回列表的元素数量
max(listname)	返回列表中元素的最大值
min(listname)	返回列表中元素的最小值
list(tuple)	将元组转换为列表
list(range)	将整数序列转换为列表

代码示例如下：

```
>>> sample_list1 = [0, 1, 2, 3, 4]
>>> len(sample_list1)      #列表的元素数量
5
>>> max(sample_list1)      #列表中元素的最大值
4
>>> min(sample_list1)      #列表中元素的最小值
0
>>> b=list(range(1,6))
>>> print(b)
[1, 2, 3, 4, 5]
```

注意：Python 3 中已经没有了Python 2中用于列表比较的cmp函数。

方法	说明
<code>listname.append(元素)</code>	添加新的元素在列表末尾
<code>listname.count(元素)</code>	统计该元素在列表中出现的次数
<code>listname.extend(序列)</code>	追加另一个序列类型中的多个值，到该列表末尾（用新列表扩展原来的列表）
<code>listname.index(元素)</code>	从列表中找出某个值第一个匹配元素的索引位置
<code>listname.insert(位置, 元素)</code>	将元素插入列表
<code>listname.pop([index=-1])</code>	移除列表中的一个元素（“-1”表示从右侧数第一个元素，也就是最后一个索引的元素），并且返回该元素的值
<code>listname.remove(元素)</code>	移除列表中的第一个匹配某个值的元素
<code>listname.reverse()</code>	将列表中元素反向
<code>listname.sort(key=None, reverse=False)</code>	对列表进行排序。这会更新列表本身，内置函数 <code>sorted(listname)</code> 则返回一个新列表， <code>listname</code> 不更新
<code>listname.clear()</code>	清空列表
<code>listname.copy()</code>	复制列表

`listname.sort(key=None, reverse=False)`

This method sorts the list in place, using only `<` comparisons between items.

`sort()` accepts two arguments that can only be passed by keyword (keyword-only arguments):

- `key` specifies a function of one argument that is used to extract a comparison key from each list element (for example, `key=str.lower`). The key corresponding to each item in the list is calculated once and then used for the entire sorting process. The default value of `None` means that list items are sorted directly without calculating a separate key value.
- `reverse` is a boolean value. If set to `True`, then the list elements are sorted as if each comparison were reversed.

This method modifies the sequence in place for economy of space when sorting a large sequence. To remind users that it operates by side effect, it does not return the sorted sequence (use `sorted()` to explicitly request a new sorted list instance).

`listname.sort(key=None, reverse=False)`

给列表的每个元素x贴一个标签f(x),
然后按这个标签排列元素

- `key` 指定一个用于元素比较的函数（可以是命名函数或匿名函数），然后按函数作用于该元素后的**返回值**作为**比较大小的依据**。例如，`key=str.lower`，则元素按小写形式比较。默认为`key=None`，即按**字面大小**比较。
- `reverse` 升序或降序排列，`reverse = True` 降序，`reverse = False` 升序（默认）

`sort()`函数会修改序列元素的位置。要返回新的列表，则应使用内置函数**`sorted()`**函数。

```

13 a = ['a', 'b', 'B', 'z', 'A']
14 print(a)
15 print(sorted(a))
16 # 使用命名函数str.lower()转化为小写字母后再比较
17 print(sorted(a, key=str.lower))
18 # 大写字母在前，小写字母在后
19 # 使用匿名函数转化为0或1后再比较
20 print(sorted(a, key=lambda x: 0 if x < 'a' else 1))
21
22 # 匿名函数存在一个变量--函数句柄中
23 f = (lambda x: 0 if x < 'a' else 1) # 定义一个匿名函数
24 print(sorted(a, key=f))
  
```

Results shown on the right:

- Line 14: `['a', 'b', 'B', 'z', 'A']`
- Line 15: `['A', 'B', 'a', 'b', 'z']`
- Line 17: `['a', 'A', 'b', 'B', 'z']`
- Line 18: `['B', 'A', 'a', 'b', 'z']`
- Line 20: `['B', 'A', 'a', 'b', 'z']`

Additional visual elements:

- A vertical toolbar with icons for list operations (add, remove, move, etc.).
- A sequence of characters: 'a', 'b', 'b', 'z', 'a' with arrows pointing to the corresponding elements in the sorted lists.
- A sequence of numbers: 1, 1, 0, 1, 0, which are the results of the lambda function applied to the elements.

4.1.6 列表推导式

列表推导式（又称列表解析式）提供了一种简明扼要的方法来创建列表。

其语法结构是在一个中括号里包含一个表达式，然后是一个for语句，然后是 0 个或多个 for 或者 if 语句。

- [表达式 for 变量 in iterable]
- [表达式 for 变量 in iterable if 条件] # 增加一个判断
- [表达式 for 变量x in iterableX if 条件x for 变量y in iterableY if 条件y] # 增加一个嵌套的for循环

表达式可以是任意的，可以在列表中放入任意类型的对象。返回结果将是一个新的列表，在这个以 if 和 for 语句为上下文的表达式运行完成之后产生。

4.1.6 列表推导式

- [表达式 for 变量 in iterable]
- [表达式 for 变量 in iterable if 条件] # 增加一个if条件判断
- [表达式1 if 条件 else 表达式2 for 变量 in iterable] # for循环内有两个分支
- [表达式 for 变量x in iterableX if 条件x for 变量y in iterableY if 条件y] # 增加一个嵌套的for循环

表达式可以是任意的，可以在列表中放入任意类型的对象。返回结果将是一个新的列表，在这个以 if 和 for 语句为上下文的表达式运行完成之后产生。

4.1.6 列表推导式

- [表达式 for 变量 in iterable]

例如：

1. 将10以内所有正整数写入列表

```
a = [x for x in range(1, 11)]  
# 产生列表 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2. 将10以内所有正整数的平方写入列表

```
b = [x**2 for x in range(1, 11)]
```

3. 100以内所有的正偶数写入列表

```
c = [x for x in range(2, 101, 2)]
```

4. 从'python1期'到'python4'期写入列表

```
d = [f'python{x}期' for x in range(1, 5)]  
# 产生列表 ['python1期', 'python2期', 'python3期', 'python4期']
```

等效于：

```
a = []  
for x in range(1, 11):  
    a.append(x)
```

等效于：

```
d = []  
for x in range(1, 5):  
    a.append(f'python{x}期')
```

4.1.6 列表推导式

- [表达式 for 变量 in iterable if 条件] # 增加if条件判断

例如:

1. 将20以内能被3或7整除的数写入列表

```
a = [x for x in range(1,21) if x%3==0 or x%7==0]
```

```
# 产生列表 [3, 6, 7, 9, 12, 14, 15, 18]
```

2. 过滤掉长度小于3的字符串列表, 并将剩下的转换成大写字母

```
l1 = ['sure', 'China', 'do', 'eye', 'is']
```

```
l2 = [x.upper() for x in l1 if len(x)>3]
```

```
# 产生列表 ['SURE', 'CHINA']
```

等效于:

```
a=[]  
for x in range(1,21):  
    if x%3==0 or x%7==0:  
        a.append(x)
```

等效于:

```
l2=[]  
for x in l1:  
    if len(x)>3:  
        l2.append(x.upper())
```


4.1.6 列表推导式

- [表达式1 if 条件 else 表达式2 for 变量 in iterable] # for循环内有两个分支

例如:

1. 将0~10的偶数乘以10, 奇数乘以-10, 结果写入列表

```
a=[x*10 if x%2==0 else -10*x for x in range(0,11)]
```

```
# 产生列表 [0, -10, 20, -30, 40, -50, 60, -70, 80, -90, 100]
```

等效于:

```
a=[]  
for x in range(1,11):  
    if x%2==0:  
        a.append(10*x)  
    else:  
        a.append(-10*x)
```

4.1.6 列表推导式

- [表达式 for 变量x in iterableX if 条件x for 变量y in iterableY if 条件y] # 增加嵌套的for循环

例如：

1. 将列表a中大于0的各元素与列表b中的各偶数的积写入列表c

```
a=[1,0,-1,5]
```

```
b=[3,2,5,6,8,7]
```

```
c=[x*y for x in a if x>0 \
    for y in b if y%2==0]
```

```
# 产生列表 [2, 6, 8, 10, 30, 40]
```

等效于：

```
c=[]
for x in a:
    if x>0:
        for y in b:
            if y%2==0:
                c.append(x*y)
```

4.1.6 列表推导式

- [表达式 for 变量x in iterableX if 条件x for 变量y in iterableY if 条件y] #增加嵌套的for循环

2. 找到嵌套列表中名字含有两个'e'的所有名字

```
names = [['Tom', 'Billy', 'Jefferson', 'Andrew', 'Wesley',  
         'Steven', 'Joe'], ['Alice', 'Jill', 'Ana', 'Wendy',  
         'Jennifer', 'Sherry', 'Eva']]  
c=[name for aList in names \  
      for name in aList if name.count('e')==2]  
# 产生列表 ['Jefferson', 'Wesley', 'Steven', 'Jennifer']
```

等效于:

```
c=[]  
for aList in names:  
    for name in aList:  
        if name.count('e')==2 :  
            c.append(name)
```

4.1.6 列表推导式

● 列表推导式的优缺点

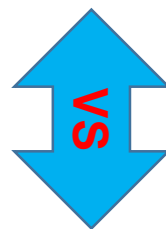
■ 优点:

简单, 高效。

■ 缺点:

可读性不高, 不好排错。

```
c=[name for aList in names \
      for name in aList if name.count('e')==2]
```



```
c=[]
for aList in names:
    for name in aList:
        if name.count('e')==2 :
            c.append(name)
```

4.1.6 列表推导式

- 使用列表推导式的效率远远高于for循环

可能执行一句`print("helloworld")`对于cpu而已只需要0.0002秒，你可能感觉不到差距，如果需要输出一亿次helloworld呢？往往细节决定成败！

例如：将0~100000000(一亿)以内的所有整数存到列表中，对比一下列表推导式和for循环耗时情况。

4.1.6 列表推导式

例如：将0~1亿以内的所有整数存到列表中，对比一下列表推导式和for循环耗时情况。

```
14 import time # 添加time模块, 用于统计代码运行时间
15
16 total_num = int(1e8) # 一共添加1亿次数据到列表中
17
18 # 使用列表推导式
19 start_time = time.time()
20 list1 = [x for x in range(0, total_num)] # 列表推导式
21 end_time = time.time()
22 print("使用列表推导式耗时: {}秒".format(end_time - start_time))
23 del list1 # 释放内存, 否则可能因内存耗尽而出现异常
24
25 # 使用普通for循环
26 start_time = time.time()
27 list2 = list()
28 for x in range(0, total_num): # for循环
29     list2.append(x)
30 end_time = time.time()
31 print("使用普通for循环耗时: {}秒".format(end_time - start_time))
```

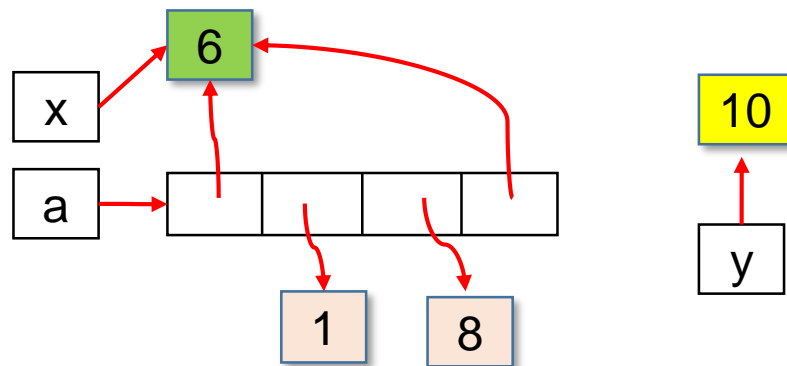
使用列表推导式耗时: 7.508276462554932秒

使用普通for循环耗时: 16.439942359924316秒

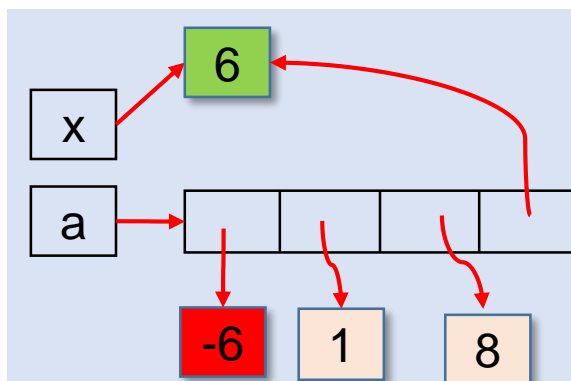
4.1.7 列表是可变对象

列表是可变对象，可以添加或删除元素。

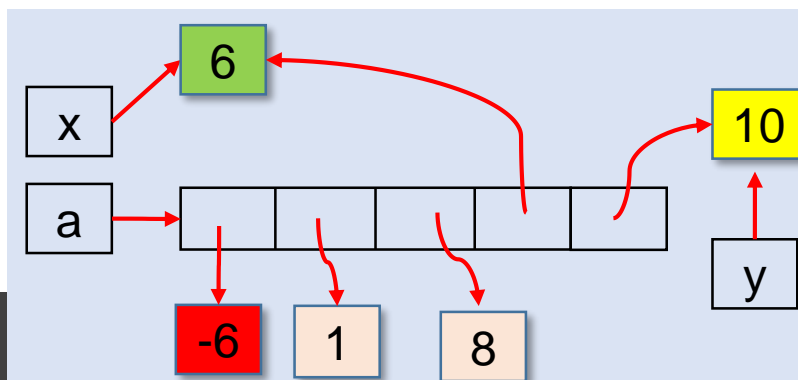
列表元素是引用对象的变量，可以引用不可变对象，也可以可变对象。



```
>>> x=6; y=10
>>> a=[6,1,8,6]
>>> id(a)
2463421082240
>>> id(a[0])
140713598916528
>>> id(a[1])
140713598916368
>>> id(x[2])
140713598916592
>>> id(a[3])
140713598916528
>>> id(x)
140713598916528
```



```
>>> a[0]=-a[0]
>>> id(a[0])
2463420616016
>>> id(a)
2463421082240
```

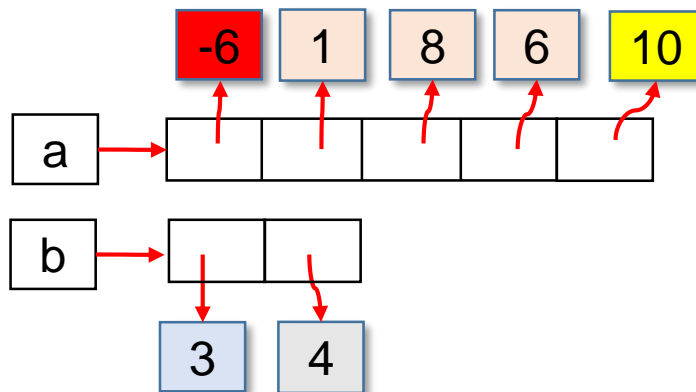


```
>>> a.append(10)
>>> id(a[4])==id(y)
True
>>> id(a)
2463421082240
```

4.1.7 列表是可变对象

```
>>> a=[-6,1,8,6,10]
```

```
>>> b=[3,4]
```



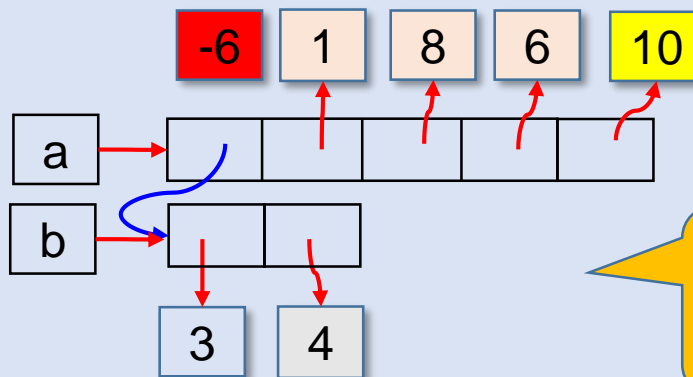
```
>>> a[0]=b # 引用列表b
```

```
>>> a
```

```
[[3, 4], 1, 8, 6, 10]
```

```
>>> b
```

```
[3, 4]
```



a[0]与b引用的是同一个列表, 它们共享内存

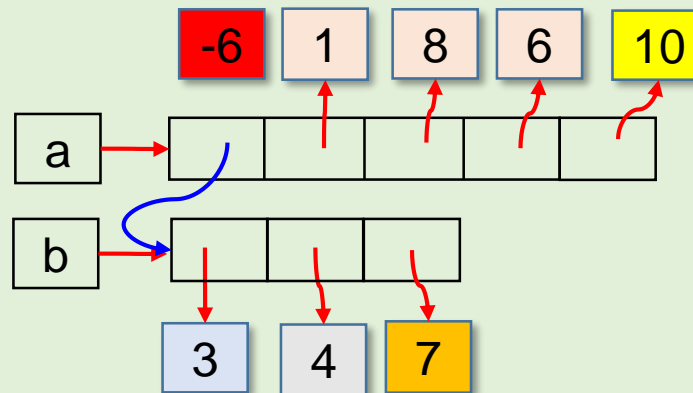
```
>>> b.append(7)
```

```
>>> b
```

```
[3, 4, 7]
```

```
>>> a
```

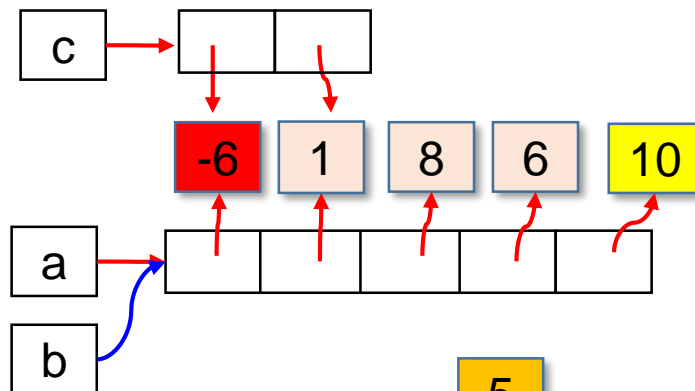
```
[[3, 4, 7], 1, 8, 6, 10]
```



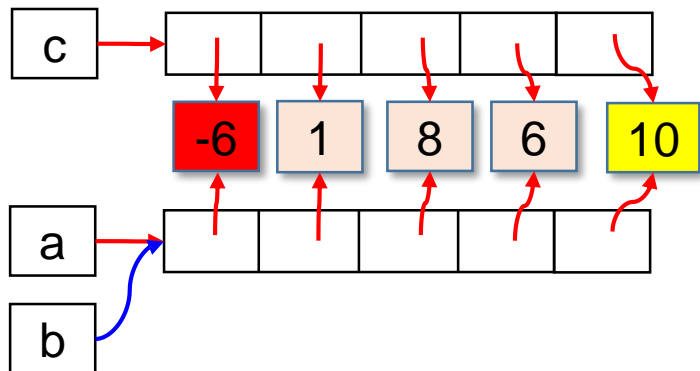
4.1.7 列表是可变对象

- 注意**
- 列表a直接赋值给一个变量b，则是a和b引用同一个列表，因此b是a的别名；
 - 列表a以切片形式赋值给一个变量c，则c引用的是一个新的列表。

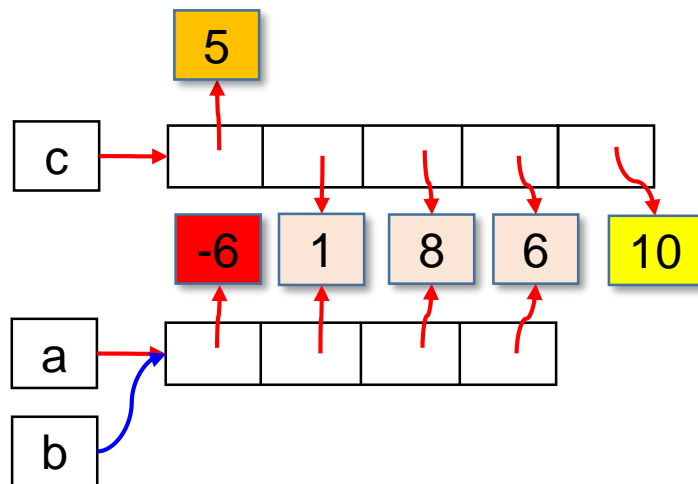
```
>>> a=[-6,1,8,6,10]
>>> b=a # b引用列表a
>>> c=a[0:2] # 返回一个新列表
```



```
>>> c=a[:] # 返回一个新列表
```



```
>>> c[0]=5
>>> b.pop()
```



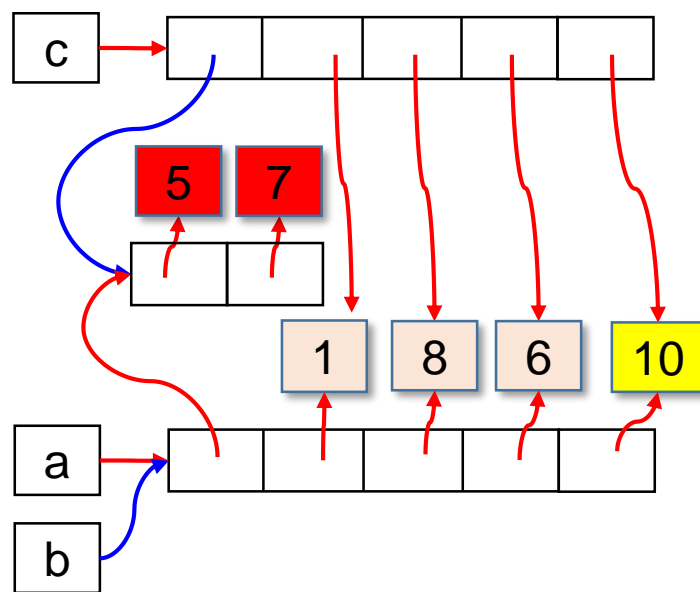
$\text{id}(a) = \text{id}(b) \neq \text{id}(c)$

4.1.7 列表是可变对象

- 注意**
- 列表a直接赋值给一个变量b，则是a和b引用同一个列表，因此b是a的别名；
 - 列表a以切片形式赋值给一个变量c，则c引用的是一个新的列表。

```
>>> a=[5,7,1,8,6,10]
>>> b=a # b引用列表a
>>> c=a[:] # 返回一个新列表
```

$\text{id}(a) = \text{id}(b) \neq \text{id}(c)$,
但是 $\text{id}(a[i]) = \text{id}(b[i]) = \text{id}(c[i])$



修改 $c[1] \sim c[-1]$ ，不影响 $a[1] \sim a[-1]$ ，它们目前引用的是不可变对象；
但是修改 $c[0]$ 所引用的列表（是可变对象），就是修改a和b；
 $c[0]$ 若引用另一个对象，则不再影响a和b。

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.2.1 创建元组

元组 (Tuples) 与列表一样，属于Python中的序列类型，它是任意对象的有序集合，通过“位置”或者“索引”访问其中的元素，它具有可变长度、异构和任意嵌套的特点，与列表不同的是：**元组中的元素是不可修改的。**

4.2.1 创建元组

元组的创建很简单，把元素放入小括号，并在每两个元素中间使用逗号隔开即可，格式为：

```
tuplename = (元素1, 元素2, 元素3, ....., 元素n)
```

举例如下：

```
sample_tuple1 = (1, 2, 3, 4, 5, 6)
```

```
sample_tuple2 = "p", "y", "t", "h", "o", "n" # 不需要括号也可以
```

```
sample_tuple3 = ('python', 'sample', 'tuple', 'for', 'your', 'reference')
```

```
sample_tuple4 = ('python', 'sample', 'tuple', 1989, 1991, 2018)
```

元组也可以为空：

```
sample_tuple5 = ()
```

需要注意的是，为避免歧义，当元组中只有一个元素时，必须在该元素后加上逗号，否则括号会被当作运算符，例如：

```
sample_tuple6 = (123,)
```

4.2.1 创建元组

元组也可以嵌套使用，例如：

```
>>> sample_tuple1 = (1, 2, 3, 4, 5, 6)
>>> sample_tuple2 = "P", "y", "t", "h", "o", "n"
>>> sample_tuple7 = (sample_tuple1, sample_tuple2)
>>> print (sample_tuple7)
((1, 2, 3, 4, 5, 6), ('P', 'y', 't', 'h', 'o', 'n'))
```

4.2.2 使用元组

与列表相同，我们可以通过使用“位置”或者“索引”来访问元组中的值，“位置”或者“索引”也是从0开始，例如：

```
sample_tuple1 = (1, 2, 3, 4, 5, 6)
```

sample_tuple1[1]表示元组tuple1中的第2个元素:2。

sample_tuple1[3:5] 表示元组sample_tuple1中的第4个和第5个元素，不包含第6个元素：4,5。

sample_tuple1[-2] 表示元组sample_tuple1中从右侧向左数的第2个元素：5。

代码示例为：

```
>>> sample_tuple1 = (1, 2, 3, 4, 5, 6)
```

```
>>> print (sample_tuple1[1])    #截取第2个元素
```

```
2
```

```
>>> print (sample_tuple1[3:5])  #第4个和第5个元素，不包含第6个元素  
(4, 5)
```

```
>>> print (sample_tuple1[-2])   #从右侧向左数的第2个元素
```

```
5
```

4.2.2 使用元组

元组也支持“切片”操作，例如

```
sample_tuple2 = "P", "y", "t", "h", "o", "n"
```

sample_tuple2[:] 表示取元组sample_tuple2的所有元素；

sample_tuple2[3:] 表示取元组sample_tuple2的索引为3的元素之后的所有元素；

sample_tuple2[0:4:2] 表示元组sample_tuple2的索引为0到4的元素，每隔一个元素取一个。

代码示例为：

```
>>> sample_tuple2 = "P", "y", "t", "h", "o", "n"
>>> print (sample_tuple2[:])      #取元组sample_tuple2的所有元素
('P', 'y', 't', 'h', 'o', 'n')
>>> print (sample_tuple2[3:])     #取元组的第4个元素之后的所有元素
('h', 'o', 'n')
>>> print (sample_tuple2[0:4:2])  #元组sample_tuple2的第1个到第5
元素，每隔一个元素取一个
('P', 't')
```


4.2.3 删除元组

由于元组中的元素是不可变的，也就是不允许被删除的，但可以使用del 语句删除整个元组：

del tuple

代码示例如下：

```
>>> sample_tuple3 = ('Python', 'sample', 'tuple', 'for', 'your', 'reference')
>>> print (sample_tuple3)          #输出删除前的元组sample_tuple3
('Python', 'sample', 'tuple', 'for', 'your', 'reference')
```

```
>>> del sample_tuple3              #删除元组sample_tuple3
>>> print (sample_tuple3)          #输出删除后的元组sample_tuple3
```

Traceback (most recent call last):

File "<pyshell#49>", line 1, in <module>

print (sample_tuple3)

NameError: name 'sample_tuple3' is not defined #系统正常报告

sample_tuple3没有定义

4.2.4 元组连接与重复

元组中的元素值是不允许修改的，但我们可以对元组进行连接（运算符+），生成新的元组，也可以使用运算符*重复元组，从而得到一个新元组。

```
>>> tup1 = (12, 34.56)
```

```
>>> tup2 = ('abc', 'xyz')
```

```
>>> tup1[0] = 100 # 修改元组元素操作是非法的
```

```
Traceback (most recent call last):
```

```
File "<input>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> tup3 = tup1 + tup2 # 创建一个新的元组：连接两个元组
```

```
>>> print (tup3)
```

```
(12, 34.56, 'abc', 'xyz')>
```

```
>>> tup4 = tup1 * 2 # 创建一个新的元组：重复元组2次
```

```
>>> print (tup4)
```

```
(12, 34.56, 12, 34.56)
```

```
>>> y=(1, 3, 5)
```

```
>>> id(y)
```

```
1555377596480
```

```
>>> y=y+(8,)
```

```
>>> y
```

```
(1, 3, 5, 8)
```

```
>>> id(y)
```

```
1555378802992
```

4.2.5 元组的内置函数

函数	说明
<code>len(tuplename)</code>	返回元组的元素数量
<code>max(tuplename)</code>	返回元组中元素的最大值
<code>min(tuplename)</code>	返回元组中元素的最小值
<code>tuple(listname)</code>	将列表转换为元组

4.2.5 元组的内置函数

代码示例如下：

```
>>> sample_tuple1 = (1, 2, 3, 4, 5, 6)  #创建元组tuple1
>>> print (len(sample_tuple1))          #输出元组长度
6
>>> print (max(sample_tuple1))          #输出元组最大值
6
>>> print (min(sample_tuple1))          #输出元组最小值
1
>>> a = [1,2,3]                          #创建列表a
>>> print (a)                            #输出列表a
[1, 2, 3]
>>> print (tuple(a))                     #转换列表a为元组后输出
(1, 2, 3)
```

4.2.6 关于元组是不可变的

所谓元组的不可变指的是**元组所指向的内存中的内容不可变**。

代码示例如下：

```
>>> tup = ('r', 'u', 'n', 'o', 'o', 'b')
```

```
>>> tup[0] = 'g'    # 不支持修改元素
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

```
>>> id(tup)    # 查看内存对象的id
```

```
4440687904
```

```
>>> tup = (1,2,3)  # tup引用了一个新的元组
```

```
>>> id(tup)
```

```
4441088800    # id不一样了
```

4.2.7 遍历元组

- 使用**for**循环遍历普通元组。例如：

```
>>> a=(1,2,3)
>>> for x in a:
...     print(x, end=' ')
...
1 2 3
```

- 使用**for**循环遍历嵌套元组。例如：

```
>>> a=((1,2),(3,4))
>>> for x in a:
...     print(x, end=' ')
...
(1, 2) (3, 4)
```

- **for**循环中使用**n个变量的元组**遍历嵌套元组（元素为n元组）。例如：

各元组元素相加：

```
>>> a=((1,2),(3,4))
>>> for (x, y) in a:
...     print(x+y, end=' ')
...
3 7
```

各元组元素交换顺序：

```
>>> a=((1,2),(3,4))
>>> for (x, y) in a:
...     print((y,x), end=' ')
...
(2, 1) (4, 3)
```

4.2.8 元组转换为列表

- 使用内置函数`list()`将元组转换为列表。例如：

```
>>> a=(1,2,3)
>>> b=list(a)
>>> b
[1, 2, 3]
```

- 使用内置函数`list()`将**嵌套元组**转换为列表，则列表元素仍是元组，这样的列表可称为**元组列表**。例如：

```
>>> a=((1,2),(3,4))
>>> b=list(a)
>>> b
[(1, 2), (3, 4)]
```

- 使用循环将**嵌套元组**转换为**嵌套列表**，即列表元素是列表。例如：

```
>>> a=((1,2),(3,4))
>>> b=[list(x) for x in a] # 或者 b=[[x,y] for (x,y) in a]
>>> b
[[1, 2], [3, 4]]
```

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.3.1 创建字典

字典 (Dictionaries) 属于映射类型，它是通过键(key)实现元素值 (value) 存取，具有无序、可变长度、异构、嵌套和**可变类型**容器等特点。

字典的每个键值(key=>value)对用冒号(:)分隔，每个对之间用逗号(,)分隔，整个字典包括在花括号({})中，格式如下：

```
dictname = {key1: value1, key2: value2, ....., keyn: valuen}
```

在同一个字典中，键必须是唯一的，但值则不必。

值可以取任何数据类型，但键必须是不可变的，如字符串，数字或元组。

举例如下：

```
sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
```

```
sample_dict2 = {12: 34, 34: 56, 56: 78}
```

```
sample_dict3 = {'Hello': 'World', 34: 56, 'City': 'CQ' }
```

```
sample_dict4 = {True: 'abc', False: 26}
```

创建字典时，同一个键被多次赋值，则最后一个值被认为是该键的值。

```
sample_dict4 = {'Model': 'PC', 'Brand': 'Lenovo', 'Brand': 'Thinkpad'}
```

这里的键Brand生效的值是Thinkpad。

4.3.1 创建字典

使用内置函数**dict()**或**{}**则创建**空字典**，例如：

```
d1 = dict()
```

```
d2 = {}
```

字典也支持嵌套，格式如下：

```
dictname = {键1: {键11: 值11, 键12: 值12 },
```

```
           键2:{ 键21: 值21, 键2: 值22},
```

```
           .....,
```

```
           键n: {键n1: 值n1, 键n2: 值n2}}
```

例如：

```
sample_dict5 = {'office':{ 'room1':'Finance ', 'room2':'logistics'},
```

```
               'lab':{ 'lab1':'Physics', 'lab2':'Chemistry'}}
```

字典键值对(key/value)中的还可以是其他任意类型，例如列表：

```
sample_dict6 = {'list1': [1,2,3], 'list2': ["China","UK","USA"]}
```

4.3.2 使用字典

使用字典中的值时，只需要把对应的键放入方括号，格式为：

dictname[键]

举例如下：

```
>>> sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
```

```
>>> print ("sample_dict1['Hello']: ", sample_dict1['Hello'])
```

```
sample_dict1['Hello']: World #输出键为Hello的值
```

```
>>> sample_dict2 = {12: 34, 34: 56, 56: 78}
```

```
>>> print ("sample_dict2[12]: ", sample_dict2[12])
```

```
sample_dict2[12]: 34 #输出键为12的值
```

使用包含嵌套的字典，例如：

```
>>> sample_dict5 = {'office': {'room1': 'Finance', 'room2': 'logistics'},  
                    'lab': {'lab1': 'Physics', 'lab2': 'Chemistry'}}
```

```
>>> print (sample_dict5['office'])
```

```
{'room1': 'Finance', 'room2': 'logistics'} #输出键为office 的值—是一个字典
```

4.3.2 使用字典

可以对字典中的已有的值进行修改，例如：

```
>>> sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
>>> print (sample_dict1['City'])      #输出键为City的值
CQ
>>> sample_dict1['City'] = 'NJ'       #把键为City的值修改为NJ
>>> print (sample_dict1['City'])      #输出键为City的值
NJ
>>> print (sample_dict1)
{'Hello': 'World', 'Capital': 'BJ', 'City': 'NJ'} #输出修改后的字典
```

可以向字典末尾**追加新的键值**，例如：

```
>>> sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
>>> sample_dict1['viewspot'] = 'HongYaDong' #把新的键和值添加到字典
>>> print (sample_dict1)                  #输出修改后的字典
{'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ', 'viewspot': 'HongYaDong'}
```

4.3.3 删除元素和字典

可以使用**del**语句删除字典中的键和对应的值，格式为：

`del dictname[键]`

使用**del**语句删除字典，格式为：

`del dictname`

举例如下：

```
>>> sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
```

```
>>> del sample_dict1['City']      #删除字典中的键City和对应的值
```

```
>>> print (sample_dict1)         #打印结果
```

```
{'Hello': 'World', 'Capital': 'BJ'}
```

```
>>> del sample_dict1             #删除该字典
```

```
>>> print (sample_dict1)         #打印该字典
```

```
Traceback (most recent call last): #系统正常报错，该字典未定义
```

```
File "<pyshell#71>", line 1, in <module>
```

```
    print (sample_dict1)
```

```
NameError: name 'sample_dict1' is not defined
```

4.3.4 字典的内置函数和方法

函数	说明
<code>len(dictname)</code>	计算键的总数
<code>str(dictname)</code>	输出字典
<code>type(dictname)</code>	返回字典类型

举例如下：

```
>>> sample_dict1 = {'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}
>>> len(sample_dict1)          #计算该字典中键的总数
3
>>> str(sample_dict1)          #输出字典
"{'Hello': 'World', 'Capital': 'BJ', 'City': 'CQ'}"
>>> type(sample_dict1)         #返回数据类型
<class 'dict'>
```

注意：

列表和元组支持连接和重复，但字典和集合不支持。

方法	说明
<code>dictname.clear()</code>	删除字典所有元素，清空字典
<code>dictname.copy()</code>	以字典类型返回某个字典的 浅复制 ，相当于 <code>dict2=dict1</code>
<code>dictname.fromkeys(seq[, value])</code>	创建一个新字典，以序列中的元素做字典的键，值为字典所有键对应的初始值，缺省为None
<code>dictname.get(value, default=None)</code>	返回指定键的值，如果值不在字典中返回default值
<code>key in dictname</code>	如果键在字典dict里返回true，否则返回false
<code>dictname.items()</code>	以列表返回可遍历的(键, 值) 元组数组，注意需要强制转换为list: <code>list(dictname.items())</code>
<code>dictname.keys()</code>	将一个字典所有的键生成列表并返回
<code>dictname.setdefault(value, default=None)</code>	和dictname.get()类似，不同点是，如果键不存在于字典中，将会 添加键 并将值设为default对应的值
<code>dictname.update(dictname2)</code>	把字典dictname2的键/值对更新到dictname里
<code>dictname.values()</code>	以列表返回字典中的所有值
<code>dictname.pop(key[, default])</code>	弹出字典给定键所对应的值，返回值为被删除的值。键值必须给出。否则，返回default值。
<code>dictname.popitem()</code>	弹出字典中的一对键和值(一般删除末尾对)，并删除

4.3.4 字典键的特性

字典值可以是任何的 python 对象，既可以是标准的对象，也可以是用户定义的，但键不行。两个重要的点需要记住：

- 不允许同一个键出现两次。创建时如果同一个键被赋值两次，后一个值会被记住。

```
>>> dict = {'Name': 'UPC', 'Age': 67, 'Name': '石油大学'}
```

```
>>> print (dict['Name'])
```

石油大学

- 键必须不可变，所以可以用数字、字符串或元组充当，而用列表就不行。

```
>>> dict = {['Name']: 'UPC', 'Age': 67}
```

Traceback (most recent call last):

File "<input>", line 1, in <module>

TypeError: unhashable type: 'list'

4.3.5 字典推导式

前面介绍过python列表推导式的使用，字典推导式使用方法其实也类似，也是通过循环和条件判断表达式配合使用，不同的是字典推导式返回值是一个字典，所以整个表达式需要写在{}内部。

- {key_exp:value_exp for key, value in dict.items() }
- {key_exp:value_exp for key, value in dict.items() if condition }
- {key_exp:value_exp1 if condition else value_exp2 for key, value in dict.items() }

其中: key_exp,value_exp1是对key, value的相应处理后的表达式

4.3.5 字典推导式

- 应用1：在字典中提取或者修改数据，返回新的字典

- 例1、获取字典中key值是小写字母的键值对

```
dict1 = {"a": 10, "B": 20, "C": True, "D": "hello world", "e": "python教程"}
```

```
dict2 = {key: value for key, value in dict1.items() if key.islower()}  
{'a': 10, 'e': 'python教程'}
```

- 例2、将字典中的所有key设置为小写

```
dict3 = {key.lower(): value for key, value in dict1.items()}  
{'a': 10, 'b': 20, 'c': True, 'd': 'hello world', 'e': 'python教程'}
```

- 例3、将字典中所有key是小写字母的value统一赋值为'error'

```
dict4 = {key: (value if key.isupper() else "error") for key, value in dict1.items()}  
{'a': 'error', 'B': 20, 'C': True, 'D': 'hello world', 'e': 'error'}
```

- 例4、将字典中的key和value交换

```
dict5 = {value: key for key, value in dict1.items()}  
{10: 'a', 20: 'B', True: 'C', 'hello world': 'D', 'python教程': 'e'}
```

4.3.5 字典推导式

- **应用2：在字符串中提取数据，返回新的字典**

在后期的爬虫课程中，我们需要获取cookies并以字典的形式传参，如果cookies是字符串则需要转换为字典。经典代码案例如下：

```
cookies = "anonymid=jy0ui55o-u6f6zd; depovince=GW; _r01_=1;  
JSESSIONID=abcMktGLRGjLtdhBk7OVw; ick_login=a9b557b8-8138-4e9d-  
8601-de7b2a633f80; .....; loginfrom=null; wp_fold=0"
```

```
cookies = {cookie.split("=")[0]:cookie.split("=")[1] \  
            for cookie in cookies.split("; ")}
```

```
{'anonymid': 'jy0ui55o-u6f6zd', 'depovince': 'GW', '_r01_': '1', 'JSESSIONID':  
'abcMktGLRGjLtdhBk7OVw', 'ick_login': 'a9b557b8-8138-4e9d-8601-  
de7b2a633f80', ....., 'loginfrom': 'null', 'wp_fold': '0'}
```

4.3.5 字典推导式

- 应用2：在字符串中提取数据，返回新的字典

```
cookies = "anonymid=jy0ui55o-u6f6zd; depovince=GW; _r01_=1;  
JSESSIONID=abcMktGLRGjLtdhBk7OVw; ick_login=a9b557b8-8138-4e9d-  
8601-de7b2a633f80; .....; loginfrom=null; wp_fold=0"
```

```
cookies = {cookie.split("=")[0]:cookie.split("=")[1]\  
            for cookie in cookies.split("; ")}
```

在字符串cookies中‘=’前面是key，‘=’后面是value，每一个由分号‘;’分隔的字符串构成一个键值对；多个键值对构成一个字典；

1. 根据‘;’将字符串拆分为列表；
2. 根据第一步获取的列表，遍历时将每一个字符串根据‘=’再次拆分；
3. 根据第二步拆分的结果，列表第一个元素作为key, 列表第二个元素作为value；

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.4.1 创建集合

集合 (set) ， 是一种集合类型， 可以理解为就是数学课里学习的集合。它是一个可以表示任意元素的集合， 它的索引可以通过另一个任意键值的集合进行， 它可以无序排列、 哈希。

集合分为两类：**可变集合 (set)** ， **不可变集合 (frozenset)** 。

可变集合， 在被创建后， 可以通过很多种方法被改变， 例如add()， update()等。

不可变集合， 由于其不可变特性， 它是可哈希的 (hashable， 意为一个对象在其生命周期中， 其哈希值不会改变， 并可以和其他对象做比较)， 也可以作为一个元素被其他集合使用， 或者作为字典的键。

4.4.1 创建集合

使用大括号 {} 或者set()创建非空集合，格式为：

```
sample_set = {值1, 值2, 值3, ....., 值n}
```

或

```
sample_set = set([值1, 值2, 值3, ....., 值n])
```

创建一个不可变集合，格式为：

```
sample_set = frozenset([值1, 值2, 值3, ....., 值n])
```

举例如下：

```
sample_set1 = {1, 2, 3, 4, 5}
```

```
sample_set2 = {'a', 'b', 'c', 'd', 'e'}
```

```
sample_set3 = {'Beijing', 'Tianjin', 'Shanghai', 'Nanjing', 'Chongqing'}
```

```
sample_set4 = set([11, 22, 33, 44, 55])
```

```
sample_set5 = frozenset(['CHS', 'ENG', '', '', '',]) #创建不可变集合
```

但创建**空集合**时必须使用set()，格式：

```
emptyset = set()
```

注意，dict1={}是创建**空字典**

4.4.2 使用集合

集合的一个显著的特点就是可以去掉**重复的元素**，例如：

```
>>> sample_set6 = set([1, 2, 3, 4, 5, 1, 2, 3, 4])
>>> print (sample_set6)           #输出去掉重复的元素的集合
{1, 2, 3, 4, 5}
```

可以使用len()函数来获得集合中元素的数量，例如：

```
>>> sample_set6 = {1, 2, 3, 4, 5, 1, 2, 3, 4,}
>>> len(sample_set6)             #输出集合的元素数量
5
```

注意：集合的元素数量，是去掉重复元素之后的数量。

4.4.2 使用集合

集合是无序的，因此没有“索引”或者“键”来指定调用某个元素，但可以使用for循环输出集合的元素，例如：

```
>>> sample_set6 = {1, 2, 3, 4, 5, 1, 2, 3, 4,}
```

```
>>> for x in sample_set6:
```

```
    print (x)
```

1

2

3

4

5

注意：输出的集合的元素，
也是去掉重复元素之后的。

4.4.2 使用集合

向集合中添加一个元素，可以使用add()方法，即把需要添加的内容作为一个元素（整体），加入到集合中，格式为：

```
setname.add(元素)
```

向集合中添加多个元素，可以使用update()方法，将另一个类型中的**元素拆分**后，添加到原集合中，格式为：

```
setname.update(others)
```

参数others可以是字符串、列表、元组，集合等组合数据类型。

4.4.2 使用集合

两种增加集合元素的方法，对可变集合有效，例如：

```
>>> sample_set1 = {1, 2, 3, 4, 5}
>>> sample_set1.add(6)           #使用add方法添加元素到集合
>>> print ("after being added, the set is: ", sample_set1)
after being added, the set is: {1, 2, 3, 4, 5, 6}
>>> sample_set1.update('python') #使用update方法添加字符串（字符结合）
>>> print ("after being updated, the set is:", sample_set1)
after being updated, the set is: {'y', 1, 2, 3, 4, 5, 6, 'p', 't', 'n', 'o', 'h', }
>>> sample_set1.update(["ENG","CHN"]) # 使用update方法添加一个列表
>>> print (sample_set1)
{'t', 1, 2, 3, 4, 5, 6, 'p', 'ENG', 'CHN', 'y', 'h', 'n', 'o'}
>>> sample_set1.updat({'Yes', 'No'}) # 使用update方法添加另一个集合
>>> print (sample_set1)
{'t', 1, 2, 3, 4, 5, 6, 'Yes', 'ENG', 'p', 'CHN', 'y', 'h', 'No', 'n', 'o'}
```

4.4.2 使用集合

集合可以被用来做成员测试，使用运算符**in**或**not in**检查某个元素是否属于某个集合。例如：

```
>>> sample_set1 = {1, 2, 3, 4, 5}
>>> sample_set2 = {'a', 'b', 'c', 'd', 'e'}
>>> 3 in sample_set1          #判断3是否在集合中，是则返回True
True
>>> 'c' not in sample_set2    #判断 “c没有在集合中”
False                         #如果c在该集合中，返回False，#否则返回True
```

实际上，列表、元组、字典也可以这样做成员测试。例如：

```
>>> 2 in [1, 2, 3, 4, 5]     #判断2是否在列表中
True
>>> 2 not in (1,2,3,4,5)     #判断2是否在元组中
False
>>> 'City' not in {'Capital': 'BJ', 'City': 'CQ'} #判断键是否在字典中
True
```

4.4.2 使用集合

集合之间可以做集合运算，求差集 (difference)、并集 (union)、交集 (intersection)、对称差集 (symmetric difference)

```
>>> s1 = {'C', 'D', 'E', 'F', 'G'}
```

```
>>> s2 = {'E', 'F', 'G', 'A', 'B'}
```

```
>>> s1 - s2      #差集
```

等价方法: `s1.difference(s2)`

```
{'D', 'C'}
```

```
>>> s1 | s2      #并集
```

等价方法: `s1.union(s2)`

```
{'A', 'G', 'B', 'F', 'E', 'D', 'C'}
```

```
>>> s1 & s2      #交集
```

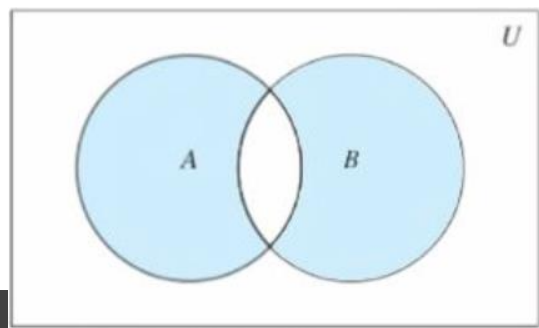
等价方法: `s1.intersection(s2)`

```
{'E', 'G', 'F'}
```

```
>>> s1 ^ s2      #对称差集
```

等价方法: `s1.symmetric_difference(s2)`

```
{'A', 'B', 'D', 'C'}
```



数学上，两个集合的对称差是只属于其中一个集合，而抄不属于另一个集合的元素组成袭的集合。集合论中的这个运算相当于布尔逻辑中的异或运算。

对称差集通常表示为: $A \Delta B = (A - B) \cup (B - A).$

等价于: $A \Delta B = (A \cup B) - (A \cap B).$

$A \Delta B = \{x : (x \in A) \text{ XOR } (x \in B)\}.$

4.4.3 删除元素和集合

可以使用remove()方法删除集合中的元素，格式为：

`setname.remove(元素)`

可使用del方法删除集合，格式为：

`del setname`

举例如下：

```
>>> sample_set1 = {1, 2, 3, 4, 5}
```

```
>>> sample_set1.remove(1)           #使用remove方法删除元素
```

```
>>> print (sample_set1)
```

```
{2, 3, 4, 5}
```

```
>>> sample_set1.clear()
```

```
>>> print (sample_set1)             #清空集合中的元素
```

```
set()                               #返回结果为空集合
```

```
>>> del sample_set1                 #删除集合
```

```
>>> print (sample_set1)
```

```
Traceback (most recent call last):   #系统报告，该集合未定义
```

```
File "<pyshell#64>", line 1, in <module>
```

```
    print (sample_set1)
```

```
NameError: name 'sample_set1' is not defined
```

4.4.4 集合的方法

方法	说明
<code>len(ss)</code> 、 <code>max(ss)</code> 、 <code>min(ss)</code>	返回集合的元素个数、最大值、最小值
<code>x in ss</code>	测试x是否是集合ss中的元素，返回True或False
<code>x not in ss</code>	如果x不在集合ss中，返回True，否则返回False
<code>ss.isdisjoint(otherset)</code>	当集合ss与另一集合otherset 不相交 时，返回True，否则返回False
<code>ss.issubset(otherset)</code> 或 <code>ss <= otherset</code>	如果集合ss是另一集合otherset的 子集 ，返回True，否则返回False
<code>ss < otherset</code>	如果集合ss是另一集合otherset的 真子集 ，返回True，否则返回False
<code>ss.issuperset(otherset)</code> 或 <code>ss >= otherset</code>	如果集合ss是另一集合otherset的 父集 ，返回True，否则返回False
<code>ss > otherset</code>	如果集合ss是另一集合otherset的父集，且otherset是ss的子集，则返回True，否则返回False
<code>ss.union(*othersets)</code> 或 <code>ss otherset1 otherset2 ...</code>	返回ss和othersets的并集，包含有set和othersets的所有元素
<code>ss.intersection(*othersets)</code> 或 <code>ss & otherset1 & otherset2 ...</code>	返回ss和othersets的交集，包含在ss并且也在othersets中的元素
<code>ss.difference(*othersets)</code> 或 <code>ss - otherset1 - otherset2 ...</code>	返回ss与othersets的差集，只包含在ss中但不在othersets中的元素
<code>ss.symmetric_difference(otherset)</code> 或 <code>ss ^ otherset</code>	返回ss与otherset的对称差集，只包含在ss中但不在othersets中，和不在ss中但在othersets中的元素
<code>ss.copy()</code>	返回集合ss的 浅拷贝

4.4.4 集合的方法

带update的方法是将集合运算的结果保存在ss中，不帶update的方法则返回一个新集合，不影响ss

方法	说明
ss.update(*othersets) 或 ss = otherset1 otherset2 ...	将另外的一个集合或多个集合元素，添加到集合ss中
ss.intersection_update(*othersets) 或 ss &= otherset1 & otherset2 ...	在ss中保留它与其他集合的交集
ss.difference_update(*othersets) 或 ss -= otherset1 otherset2 ...	从ss中移除它与其他集合的交集，保留不在其他集合中的元素
ss.symmetric_difference_update(otherset) 或 ss ^= otherset	集合ss与另一集合otherset交集的补集，将结果返回到ss
ss.add(元素)	向集合ss中添加元素
ss.remove(元素)	从集合ss中移除元素，如果该元素不在ss中，则报告KeyError
ss.discard(元素)	从集合ss中移除元素，如果该元素不在ss中，则什么都不做
ss.pop()	移除并返回集合ss中的任一元素，如果ss为空，则报告KeyError
ss.clear()	清空集合ss中所有元素

4.4.5 集合推导式

集合推导式与列表推导式类似，区别在于它使用大括号{}，且元素在集合中唯一。

举例如下：

```
>>> squared = {x**2 for x in [1, 1, 2]} # 集合推导式
```

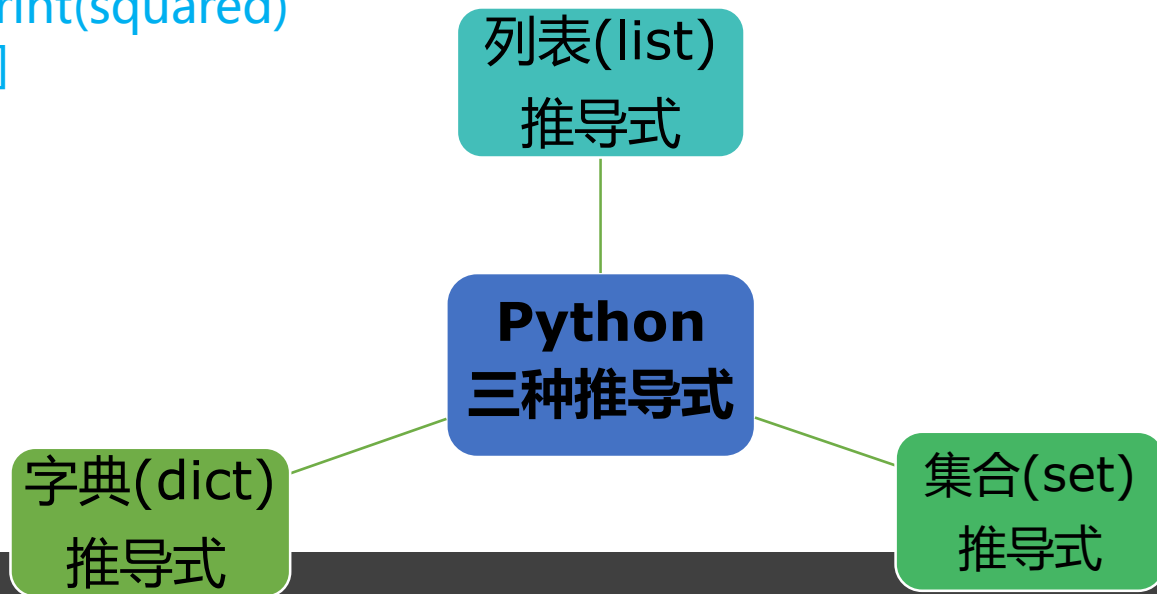
```
>>> print(squared)
```

```
{1, 4}
```

```
>>> squared = [x**2 for x in [1, 1, 2]] # 列表推导式
```

```
>>> print(squared)
```

```
[1, 1, 4]
```



第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.5.1 zip 对象

zip() 函数是 Python 内置函数之一，它可以将多个序列（列表、元组、字典、集合、字符串以及 range() 区间构成的列表）“压缩”成一个 **zip 对象**。所谓“压缩”，其实就是将这些序列中**对应位置**的元素**重新组合**，生成一个个新的元组。

zip() 函数的语法格式为：

zip(X,Y, ...)

其中 X, Y 表示可迭代对象 (iterable) 可以是列表、元组、字典、集合、字符串，甚至还可以为 range() 区间。

zip(X,Y) 对象中的第 i 个元组为 (X_i, Y_i) ，即该元组由可迭代序列 X 中的第 i 个元素 X_i 和 Y 中的第 i 个元素 Y_i 构成。通常，X、Y 的长度是相同的。

zip 对象也是可变迭代的，可以用 next 函数读取其中的元组，也可 for 循环遍历，这样一个 for 循环就可以同时遍历多个序列。

4.5.1 zip 对象

```
names = ['小英', '李四']
```

```
ages = [18, 95]
```

```
g = zip(names, ages) # 捆绑两个可迭代对象X,Y,长度可以不一样
```

```
# 用next函数访问zip对象中的数据, 为元组(Xi, Yi)
```

```
print(next(g)) # 输出: ('小英', 18)
```

```
print(next(g)) # 输出: ('李四', 95)
```

```
# 遍历zip对象
```

```
for x in zip(names, ages):  
    print(x)
```

```
('小英', 18)  
( '李四', 95)
```

```
for key, value in zip(names, ages):  
    print(key, value)
```

```
小英 18  
李四 95
```

```
# 用zip对象初始化一个字典
```

```
d = dict(zip(names, ages))  
print(d)
```

```
{ '小英': 18, '李四': 95 }
```

4.5.1 zip 对象

zip函数可以压缩不同的序列，当压缩的多个序列中元素个数不一致时，会以最短的序列为准进行压缩。

压缩列表和元组

```
my_list = [11, 12, 13]
```

```
my_tuple = (21, 22, 23)
```

```
print([x for x in zip(my_list, my_tuple)])
```

```
[(11, 21), (12, 22), (13, 23)]
```

压缩字典和集合，字典取了键，且字典与集合长度不一致

```
my_dic = {31: 2, 32: 4, 33: 5}
```

```
my_set = {41, 42, 43, 44}
```

```
print([x for x in zip(my_dic, my_set)])
```

```
[(31, 41), (32, 42), (33, 43)]
```

压缩字符串和字符串，且两者长度不一致

```
my_pychar = "python"
```

```
my_shechar = "shell"
```

```
print([x for x in zip(my_pychar, my_shechar)])
```

```
[('p', 's'), ('y', 'h'), ('t', 'e'),  
( 'h', 'l'), ('o', 'l')]
```

压缩字符串和列表，且两者长度不一致

```
print([x for x in zip(my_pychar, my_list)])
```

```
[('p', 11), ('y', 12), ('t', 13)]
```

4.5.1 zip 对象

对于 zip() 函数返回的 zip 对象，既可以像上面程序那样，通过遍历提取其存储的元组，也可以向下面程序这样，通过调用 list() 函数将 zip() 对象强制转换成元组列表：

```
print(list(zip(names, ages))) # 输出: [('小英', 18), ('李四', 95)]
```

传递参数时，可以使用*运算符解压（解包）：

```
g = zip(names, ages) # 压缩
```

```
k = zip(*g) # 与 zip 相反，*g 可理解为解压。
```

*但不允许单独使用，例如keys, values= *g*

```
for x, y in k:  
    print(x, y)
```

```
小英 18  
李四 95
```

4.5.2 enumerate 对象

enumerate 对象是用于枚举可迭代对象seq的元素的元组序列，元组由元素及其序号构成，默认为(i, seq[i])，即元组序列默认为：

(0, seq[0]), (1, seq[1]), (2, seq[2]), ...

enumerate 对象由enumerate 函数生成，语法为：

enumerate(sequence, [start=0])

其中：sequence -- 一个序列、迭代器或其他支持迭代对象

start -- 元组序列的下标起始位置，默认为0

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
```

```
print(list(enumerate(seasons)))
```

```
# 输出: [(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

```
print(list(enumerate(seasons, start=1))) # 下标从 1 开始
```

```
# 输出: [(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

4.5.2 enumerate 对象

enumerate 对象本身也是可迭代的，一般用在 for 循环中，以便访问 seq 的元素及其索引号。

例如，要访问列表 seasons = ['Spring', 'Summer', 'Fall', 'Winter'] 中的元素及索引号，可以有若干种方法：

方法1：普通for循环，遍历列表，维护计数变量

`i = 0`

`for season in seasons:`

`print(i, season)`

`i = i + 1`

#方法2：普通for循环，利用下标访问列表元素

`for i in range(len(seasons)):`

`print(i, seasons[i])`

0 Spring

1 Summer

2 Fall

3 Winter

4.5.2 enumerate 对象

方法3: for循环使用enumerate枚举元素

```
for i, season in enumerate(seasons):  
    print(i, season)
```

最简洁

方法4: 普通for循环, 利用zip

```
for i, season in zip(range(len(seasons)), seasons):  
    print(i, season)
```

0 Spring
1 Summer
2 Fall
3 Winter

改变起始索引

```
for i, season in enumerate(seasons, start=1):  
    print(i, season)
```

1 Spring
2 Summer
3 Fall
4 Winter

4.5.2 enumerate 对象

枚举字符串中字符和枚举字典中的关键字：

枚举字符串中字符

company = 'IBM'

```
for i, letter in enumerate(company):  
    print(i, letter)
```

0 I

1 B

2 M

枚举字典

data = {'China': 55, 'USA': 34}

```
for i, country in enumerate(data, start=1):  
    print(i, country)
```

1 China

2 USA

4.5.2 enumerate 对象

补充:

如果要统计文件的行数, 可以这样写:

```
count = len( open(filepath, 'r').readlines() )
```

这种方法简单, 但是可能比较慢, 当文件比较大时甚至不能工作。

可以利用enumerate():

```
count = 0
for index, line in enumerate(open(filepath, 'r')):
    count += 1
```

4.5.3 itemgetter 对象

operator 模块的 itemgetter 对象表示一个可调用 (callable) 对象 (可作为一个函数)，用于从序列获取指定数据项。定义该函数时，要传入一些表示序号的若干参数，当该函数被调用时传入需要获取数据的序列。

例如：

定义函数 $f = \text{itemgetter}(2)$ ，则调用 $f(r)$ 将返回序列 r 中元素 $r[2]$ 。

定义函数 $g = \text{itemgetter}(2, 5)$ ，则 $g(r)$ 将返回序列 r 中元组 $(r[2], r[5])$ 。

```
from operator import itemgetter
```

```
f = itemgetter(2) # 返回一个对象itemgetter对象，  
                  用于返回指定的可迭代对象中的索引号为2的元素。
```

```
a = [3, 5, 9]
```

```
print(f(a)) # 等价于print(a[2])
```

9

注意，itemgetter 对象本身定义的是一个对象（可作为一个函数），当它作用到可迭代对象上时才能获取值。

4.5.3 itemgetter 对象

from operator **import** itemgetter

f = itemgetter(2) # 返回一个对象itemgetter对象,
用于返回指定的可迭代对象中的索引号为2的元素。

a = [3, 5, 9]

print(f(a)) # 等价于**print(a[2])** 9

b = [('Japan', 35), ('USA', 50), ('China', 80), ('UK', 20)]

print(f(b)) # 等价于**print(b[2])** ('China',80)

g = itemgetter(2, 0) # 返回一个对象itemgetter对象,
用于返回指定的可迭代对象中的索引号为2和0的元素。

print(g(a)) # 等价于**print(((a[2],a[0])))** (9, 3)

print(g(b)) # 等价于**print((b[2],b[0]))** (('China', 80), ('Japan', 35))

4.5.3 itemgetter 对象

对于元素由多个字段构成的序列，例如二元组列表，如果它需要用 sorted 函数按某个字段排序，则可以使用 itemgetter 对象定义排序时使用的函数。

```
from operator import itemgetter
```

```
# 对元组列表中的元素按元组中的第2个元素的降序排列
```

```
b = [('Japan', 35), ('USA', 50), ('China', 80), ('UK', 20)]
```

```
g = itemgetter(1)
```

```
result = sorted(b, key=g, reverse=True)
```

```
print(result)
```

```
[('China', 80), ('USA', 50),  
 ('Japan', 35), ('UK', 20)]
```

sorted函数排序时，每次从b中取一个二元组，然后g函数从该元组中取出索引号为1的元素，作为该元组的排序标签

```
result = sorted(b, key=lambda x: x[1], reverse=True)
```

4.5.3 itemgetter 对象

对于元素由多个字段构成的序列，例如二元组列表，如果它需要用 `sorted` 函数按某个字段排序，则可以使用 `itemgetter` 对象定义排序时使用的函数。

from operator **import** itemgetter

```
students = [('John', 'A', 15), ('Mark', 'B', 12), ('Dave', 'B', 10)]  
# 根据元组的第2个和第3个元素进行升序排列元组  
result = sorted(students, key=itemgetter(1, 2))  
print(result)
```

```
[('John', 'A', 15), ('Dave', 'B', 10), ('Mark', 'B', 12)]
```



```
result = sorted(students, key=lambda x: (x[1], x[2]))
```

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

4.6.1 元组的使用

例：将元组(1, 2, 3, 4)中第二个元素修改为5

```
x = (1, 2, 3, 4) # 创建元组x
list1 = list(x) # 首先将元组转为列表类型
list1[1] = 5 # 在列表类型上修改内容
print(list1)
y = tuple(list1) # 将列表转为元组
print(type(y), y)
print("id(x) =", id(x), "\nid(y) =", id(y))
```



```
[1, 5, 3, 4]
```

```
<class 'tuple'> (1, 5, 3, 4)
```

```
id(x) = 1497752910480
```


```
id(y) = 1497752935296
```

注意：元组本身不允许修改，因此，先将其转换为列表，然后在列表上修改，最后将列表转换为一个新的元组。两个元组具有不同的内存地址。

4.6.2 列表的使用

例：创建一个整数列表num，生成10个200以内的随机数，然后求出最大值、最小值以及总和，并将这些随机数排序（升序或降序），最后将结果输出。

```
7 import random # 导入random模块
8
9 num = random.sample(range(200), 10) # 生成200以内的10个随机数
10 print("10个随机数为: \n", num, type(num))
11 print("其中最大的数为: ", max(num)) # 输出最大值
12 print("其中最小的数为: ", min(num)) # 输出最小值
13 print("总和为: ", sum(num)) # 输出总和
14 print("升序排列为: \n", sorted(num))
15 print("降序排列为: \n", sorted(num, reverse=True))
```



```
10个随机数为:
[166, 188, 62, 117, 11, 18, 24, 141, 21, 111] <class 'list'>
其中最大的数为: 188
其中最小的数为: 11
总和为: 859
升序排列为:
[11, 18, 21, 24, 62, 111, 117, 141, 166, 188]
降序排列为:
[188, 166, 141, 117, 111, 62, 24, 21, 18, 11]
```

4.6.2 列表的使用

例：字符串列表切片与遍历

```
1  # 字符串列表切片与遍历
2  stars = ['张学友', '刘德华', '黎明', '郭富城']
3  print(stars[0:3])
4  print(stars[1:4])
5  print(stars[:2])
6  print(stars[2:])
7  print(stars[-2:])
8
9  # 列表遍历切片
10 musics = ['只想一生跟着你走', '开心的马骝', '人在黎明', '痛哭']
11 for music in musics[:3]:
12     print("我喜欢的歌曲之一: " + music)
```

['张学友', '刘德华', '黎明']
['刘德华', '黎明', '郭富城']
['张学友', '刘德华']

['黎明', '郭富城']

['黎明', '郭富城']

我喜欢的歌曲之一: 只想一生跟着你走

我喜欢的歌曲之一: 开心的马骝

我喜欢的歌曲之一: 人在黎明

4.6.3 字典的使用

例：创建10个银行卡号，6103452xxx: 6103452001, 6103452002 610345210，这些银行卡号的初始密码为888888，账号与密码的对应关系存储在字典中。使用字典的fromkeys()方法可以快速实现。

```
5 # pprint==pretty print, 更加美观/友好的打印模块
6 import pprint
7
8 # 1) 生成10个卡号, 存储在列表中
9 cards = []
0 for count in range(1, 11):
1     num = "%03d" % count # 格式化为三位数, 不够的左边补0
2     card = '6103452' + num
3     cards.append(card)
4
5 # 2) 快速生成卡号和密码的对应关系, 存储在字典中
6 cards_dict = {}.fromkeys(cards, '888888')
7 # print(cards_dict)
8 pprint.pprint(cards_dict)
```

```
{'6103452001': '888888',
'6103452002': '888888',
'6103452003': '888888',
'6103452004': '888888',
'6103452005': '888888',
'6103452006': '888888',
'6103452007': '888888',
'6103452008': '888888',
'6103452009': '888888',
'6103452010': '888888'}
```

4.6.4 集合的使用

例（华为笔试编程题）：小明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N个1 ~ 100之间的随机整数($N \leq 100$)。对于其中重复的整数，只保留一个，把其余相同的数字去掉，不同的数对应不同的学生的学号，然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。

请你协助小明完成“去重”与排序工作。

```
9  import random  # 导入random模块
10
11  MIN, MAX = 1, 100  # 随机整数的最小值、最大值
12  N = int(input("输入整数N:"))
13  a = set()  # 创建一个空集合
14  for i in range(1, N + 1):
15      n = random.randint(MIN, MAX)
16      a.add(n)  # 向集合添加元素
17  b = sorted(a, reverse=False)  # 升序排列集合元素,返回列表
18  print("集合a:", a)
19  print("集合b:", b)
```

某两次运行结果

输入整数N: 10
集合a: {33, 100, 36, 70, 6, 10, 11, 50, 24, 27}
集合b: [6, 10, 11, 24, 27, 33, 36, 50, 70, 100]

输入整数N: 10
集合a: {96, 97, 68, 5, 13, 80, 21, 23, 30}
集合b: [5, 13, 21, 23, 30, 68, 80, 96, 97]

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

本章主要讲解了Python的4种组合数据类型：列表、元组、字典和集合，以及它们的特点、使用方法等。

结合前几章的知识，我们可以知道：**不可变数据类型**有四个：数字（Numbers）、字符串（Strings）、元组（Tuples）、不可变集合（FrozenSets）；元素可变的数据类型有三个：列表（Lists）、字典（Dictionary）、可变集合(Sets)。其中，列表是Python中使用最为频繁的数据类型之一。

一切语言的基础就是数据类型

第四章 组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合

4.5 zip、enumerate 和 itemgetter对象

4.6 实验

4.7 小结

习题

习题:

- 1、简述元组和列表的相同点和不同点。
- 2、简述字典和集合的相同点和不同点。
- 3、如何判断某列表是否存在重复元素？请举例并编程说明。
- 4、有以下URL地址列表，请设法去除重复值：

```
urls=['http://www.upc.edu.cn', 'http://www.sina.com', \
      'http://www.baidu.com', 'http://www.upc.edu.cn', \
      'http://www.gov.cn/guowuyuan']
```
- 5、生成20个1-100之间的随机整数，并将其中大于等于66的元素和小于66的元素分类存储在一个列表中。
- 6、将列表中的每个元素一次向前移动一个位置,第一个元素移到列表的最后，然后输出这个列表。例如 [1,2,3,4,5,6,7,8,9,0]移动元素后变为[2,3,4,5,6,7,8,9,0,1]
- 7、有一个整数列表，调整其元素顺序：把所有的奇数放在前面，偶数放在后面。

习题：

8、已知元组列表`a = [(2, 2), (3, 4), (4, 1), (1, 3)]`，要求按元组第二个元素升序排列列表。提示：自定义函数获取返回元组的第二个元素，该函数作为`sort`函数的`key`。

9、利用字典统计文本中各单词出现的频次，并对结果降序排列。主要过程如下：

- ① 文本中的标点符号（`~@#$%^&*()_-=<>?/,.;:{}[]|\'"`这些标点符号在`string`模块的`punctuation`常量中已定义）先替换成空格（对每个字符循环使用`in`判断在指定标点序列中，字符串的`replace`方法）；
- ② 分割字符串，得到单词列表`words`；（字符串的`split`方法）
- ③ 建立空字典`wordCounts`；
- ④ 遍历单词列表`words`中的单词`x`，如果单词`x`不在字典`wordCounts`中（不是其中的键），则将键值对`x:1`加入字典`wordCounts`；否则，将字典`wordCounts`中的键`x`的值加1，这样便可实现单词出现频次统计。（`if x not in wordCounts`）
- ⑤ 从字典中获取数据对(元组序列)并转换为列表（`list(wordCounts.items())`）
- ⑥ 按列表中元组的第二个元素降序排列（`sorted`方法）

感谢聆听

