第四章 List和Tuple类型

一、dict类型

1.什么是dict

名字和分数关联起来，形成一个类似的查找表，名字key，成绩value

```
d = {
    'Adam': 95,
    'Lisa': 85,
    'Bart': 59
}
```

花括号表示这是一个dict，然后按照key:value写出来即可，最后一个key:value的逗号可以省略

由于dict也是集合，len()函数可以计算任意集合的大小，一个key:value算一个

2.访问dict

使用d[key]

list 必须使用索引返回对应的元素，而dict使用key:

```
print d['Adam']
95
print d['Paul']
Traceback (most recent call last):
  File "index.py", line 11, in <module>
   print d['Paul']
KeyError: 'Paul'
```

注意：通过key访问dict的value，只要key存在，dict就返回对应的value，如果key不存在，会直接报错：KeyError

要避免KeyError发生，有两个办法：

1）先判断一下key是否存在，用in操作符

```
if 'Paul' in d:
    print d['Paul']
    #如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。
```

2）使用dict本身提供的一个get办法，在Key不存在的时候，返回None：

```
print d.get('Bart')
#59
print d.get('Paul')
#None
```

练习：打印出一个dict

```
d = {
    'Adam': 95,
    'Lisa': 85,
    'Bart': 59
}
print 'Adam:', d.get('Adam')
print 'Lisa:', d.get('Lisa')
print 'Bart:', d.get('Bart')
```

3.dict的特点

（1）查找速度快，无论dict有10个元素还是10万个元素，查找速度都一样。而list的查找速度随着元素增加而逐渐下降。

不过dict的查找速度快不是没有代价的，dict的缺点是占用内存大，还会浪费很多内容，list正好相反，占用内存小，但是查找速度慢。

由于dict是按 key 查找，所以，在一个dict中，key不能重复。

（2）存储的key-value序对是没有顺序的，和list不一样，

打印的顺序不一定是我们创建时的顺序，而且，不同的机器打印的顺序都可能不同，这说明dict内部是无序的，不能用dict存储有序的集合。

（3）作为 key 的元素必须不可变，Python的基本类型如字符串、整数、浮点数都是不可变的，都可以作为 key。但是list是可变的，就不能作为 key。

不可变这个限制仅作用于key，value是否可变无所谓：

```
{
    '123': [1, 2, 3],  # key 是 str，value是list
    123: '123',  # key 是 int，value 是 str
    ('a', 'b'): True  # key 是 tuple，并且tuple的每个元素都是不可变对象，value是 boolean
}
```

最常用的key还是字符串，因为用起来最方便。

练习：根据分数索引名字

```
# -*- coding: utf-8 -*-
d = {
    95:'Adam',
    85:'Lisa',
    59:'Bart'
}
print d.get(85)
```

4.更新dict

dict是可变的，也就是说，我们可以随时往dict中添加新的 key-value。

要把新同学'Paul'的成绩 72 加进去，用赋值语句：

```
d['Paul'] = 72
```

5.遍历dict

由于dict也是一个集合，所以，遍历dict和遍历list类似，都可以通过 for 循环实现。

使用for循环遍历dict中的key：

```
d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }
for key in d:
    print key
```

通过key可以获取对应的value的值，因此，在循环体内，可以获取value的值

练习：使用for循环遍历dict，打印出name：score出来

```
d = {
    'Adam': 95,
    'Lisa': 85,
    'Bart': 59
}
for key in d:
    print key,":",d[key]
```

二、set类型

1.什么是set

dict的作用是建立一组 key 和一组 value 的映射关系，dict的key是不能重复的。

当我们只想要 dict 的 key，不关心 key 对应的 value，目的就是保证这个集合的元素不会重复，这时，set就派上用场了。

set 持有一系列元素，这一点和 list 很像，但是set的元素没有重复，而且是无序的，这点和 dict 的 key很像。

创建 set 的方式是调用 set() 并传入一个 list，list的元素将作为set的元素：

```
s = set(['A', 'B', 'C'])
print s
```

形式类似 list， 但它不是 list，仔细看还可以发现，打印的顺序和原始 list 的顺序有可能是不同的，因为set内部存储的元素是无序的。

因为set不能包含重复的元素，所以，当我们传入包含重复元素的 list 会怎么样呢?

```
s = set(['A', 'B', 'C', 'C'])
print s
set(['A', 'C', 'B'])
len(s)
```

结果显示，set会自动去掉重复的元素，原来的list有4个元素，但set只有3个元素。

2.访问set

由于set存储的是无序集合，所以我们没法通过索引来访问。

访问 set中的某个元素实际上就是判断一个元素是否在set中。

用 in 操作符判断：注意大小写

```
s = set(['Adam', 'Lisa', 'Bart', 'Paul'])
'Bart' in s
 #True
 'Bill' in s
 #False
 'bart' in s
 #False
```

练习：请改进set，使得 'adam' 和 'bart'都能返回True。

```
s = set([name.lower() for name in ['Adam', 'Lisa', 'Bart', 'Paul']])
print 'adam' in s
print 'bart' in s
# L=['Adam', 'Lisa', 'Bart', 'Paul']
# M=[]
# for x in L:
#     y=x.lower()
#     print x
#     M.append(y)
# s = set(M)
# print 'adam' in s
# print 'bart' in s
```

3.set的特点

（1）set的内部结构和dict很像，唯一区别是不存储value，因此，判断一个元素是否在set中速度很快。

（2）set存储的元素和dict的key类似，必须是不变对象，因此，任何可变对象是不能放入set中的

（3）set存储的元素也是没有顺序的。

```
weekdays = set(['MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'])
x = '???' # 用户输入的字符串
if x in weekdays:
    print 'input ok'
else:
    print 'input error'
```

4.遍历set

for循环

```
s = set(['Adam', 'Lisa', 'Bart'])
for name in s:
    print name
```

注意：for循环在遍历set时，元素的顺序很可能是不同的，而且不同机器上运行的结果也可能会是不同的

练习： for 循环遍历如下的set，打印出 name: score

```
s = set(['Adam', 'Lisa', 'Bart'])
for name in s:
    print name
```

5.更新set

由于set存储的是一组不重复的无序元素，因此，更新set主要做两件事：

（1）把新的元素添加到set中，（2）把已有元素从set中删除。

添加元素时，用set的add()方法

```
s = set([1, 2, 3])
s.add(4)
print s
set([1, 2, 3, 4])
```

如果添加的元素已经存在于set中，add()不会报错，但是不会加进去了：

```
s = set([1, 2, 3])
s.add(3)
print s
set([1, 2, 3])
```

删除set中的元素时，用set的remove()方法：

```
s = set([1, 2, 3, 4])
s.remove(4)
print s
#set([1, 2, 3])
```

如果删除的元素不存在set中，remove()会报错：

```
s = set([1, 2, 3])
s.remove(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
```

所以用add()可以直接添加，而remove()前需要判断。

练习：针对下面的set，给定一个list，对list中的每一个元素，如果在set中，就将其删除，如果不在set中，就添加进去。 s = set(['Adam', 'Lisa', 'Paul']) L = ['Adam', 'Lisa', 'Bart', 'Paul']

```
s = set(['Adam', 'Lisa', 'Paul'])
L = ['Adam', 'Lisa', 'Bart', 'Paul']
for name in L:
    if name in s:
        s.remove(name)
    else:
        s.add(name)
print s
```