```
#encoding=utf-8

#**********      映射    **************

#*************Part 1:   映射类型：字典  ************
#字典是 Python  语言中唯一的映射类型。映射类型对象里哈希值(键)  和指向的对象(值)是一对
#多的关系。 一个字典对象是可变的，它是一个容器类型，能存储任意个数的 Python  对象，其中也包括其他容器
#类型。字典类型和序列类型容器类(列表、元组)的区别是存储和访问数据的方式不同。序列类型只
#用数字类型的键(从序列的开始起按数值顺序索引)。映射类型可以用其他对象类型做键；一般最常
#见的是用字符串做键(keys)。和序列类型的键不同，映射类型的键(keys)直接，或间接地和存储的
#数据值相关联。但因为在映射类型中，我们不再用"序列化排序"的键(keys),所以映射类型中的数据是无序排列的。
#字典是 Python  中最强大的数据类型之一。

#核心笔记：什么是哈希表？它们与字典的关系是什么？
#哈希表是一种数据结构：它按照我们所要求的去工作。哈希表中存储的每一条数据，叫做一个
#值(value)，是根据与它相关的一个被称作为键(key)的数据项进行存储的。键和值合在一起被称为
#“键-值  对”(key-value  pairs)。  哈希表的算法是获取键，对键执行一个叫做哈希函数的操作，
#并根据计算的结果，选择在数据结构的某个地址中来存储你的值。任何一个值存储的地址皆取决于
#它的键。正因为这种随意性，哈希表中的值是没有顺序的。你拥有的是一个无序的数据集。
#你所能获得的有序集合只能是字典中的键的集合或者值的集合。方法 Keys()  或  values()  返回
#一个列表，该列表是可排序的。  你还可以用  items()方法得到包含键、值对的元组的列表来排序。
#由于字典本身是哈希的，所以是无序的。哈希表一般有很好的性能，因为用键查询相当快。

#Python  的字典是作为可变的哈希表实现的。
```

```
#创建字典和给字典赋值
#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#print(dict1)
#-->{'HomeTown': 'anqing', 'Name': 'dave', 'Sex': 'Man'}
#这里可以看出，映射是无序的

#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#访问某一个值
#print(dict1['Name'])
#-->dave

#for 循环列出所有值
#for key in dict1:
#       print('key=%s,value=%s' %(key,dict1[key]))
#-->
#key=HomeTown,value=anqing
#key=Name,value=dave
#key=Sex,value=Man

#判断某个键值是否存在字典里
#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#print('Name' in dict1)
#-->True
#print('dave' in dict1)
#-->False

#在前面的示例中是一次给字典赋值的，也可以分开来赋值
#dict={}
#dict['Name']='Dave'
#dict['Age']=25
#dict['Hometown']='anqing'
#print(dict)
#-->
#{'Hometown': 'anqing', 'Age': 25, 'Name': 'Dave'}


#更新字典
#可以通过以下几种方式对一个字典做修改：添加一个新数据项或新元素(即，一个键-值对)；
修改一个已存在的数据项；或删除一个已存在的数据项

#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#修改 key
#dict1['Name']='David Dai'
#增加 key
```

```
#dict1['Age']=25
#print(dict1['Age'],dict1['Name'])
#-->
#25 David Dai

#删除某个 Key
#del dict1['Name']
#print(dict1)
#-->{'HomeTown': 'anqing', 'Sex': 'Man'}

#删除并返回键为 Key 的条目
#print(dict1.pop('Name'))
#-->dave

#删除所有条目
#dict1.clear()
#print(dict1)
#-->{}

#删除字典
#del dict1
#print(dict1)
#-->NameError: name 'dict1' is not defined


#************* Part 2: 映射类型操作符 *****************
#字典可以和所有的标准类型操作符一起工作,但却不支持像拼接（concatenation）和重复(repetition)这样的操作。

#标准类型操作符
#dict4 = {'abc': 123}
#dict5 = {'abc': 456}
#print(dict4['abc']<dict5['abc'])
#-->True

#映射类型操作符

#字典的键查找操作符（[ ]）
#键查找操作符是唯一仅用于字典类型的操作符，对字典类型来说，是用键(key)查询(字典中的元素),所以键是参数(argument)。
#键查找操作符既可以用于给字典赋值，也可以用于从字典中取值：
#d[k] = v 通过键'k'，给字典中某元素赋值'v'
#d[k] 通过键'k'，查询字典中某元素的值
```

#(键)成员关系操作(in ,not in)
#用 in 和 not in 操作符来检查某个键是否存在于字典中,这个之前有示例

#********** Part 3： 映射类型的内建函数和工厂函数 **********

#映射类型相关的函数
#dict()
#工厂函数被用来创建字典。如果不提供参数，会生成空字典。
#dict1=dict([['x', 1], ['y', 2]])
#print(dict1)

#如果输入参数是(另)一个映射对象，比如，一个字典对象，对其调用 dict()会从存在的字典里
#复制内容来生成新的字典。新生成的字典是原来字典对象的浅复制版本， 它与用字典的内建方法
#copy() 生成的字典对象是一样的。但是从已存在的字典生成新的字典速度比用 copy()方法慢，我们
#推荐使用 copy()。

#dict1={'Name':'Dave','Hometown':'安庆'}
#dict2=dict1.copy()
#print(dict2)
#-->
#{'Hometown': '安庆', 'Name': 'Dave'}

#len()
#内建函数 len()很灵活。它可用在序列、映射类型和集合上。对字典调用 len()，它会返回所有元素(键-值对)的数目：

#dict1={'Name':'Dave','Hometown':'安庆'}
#print(len(dict1))
#-->2

#hash()
#内建函数 hash()本身并不是为字典设计的方法，但它可以判断某个对象是否可以做一个字典的
#键。将一个对象作为参数传递给 hash(), 会返回这个对象的哈希值。 只有这个对象是可哈希的，
#才可作为字典的键 (函数的返回值是整数，不产生错误或异常)。
#如果用比较操作符来比较两个数值，发现它们是相等的，那么即使二者的数据类型不同，它们也会得到相同的哈希值。
#如果非可哈希类型作为参数传递给 hash()方法，会产生 TypeError 错误(因此，如果使用这样的对象作为键给字典赋值时会出错):

```
#>>> hash([])
#Traceback (innermost last): File "<stdin>", line 1, in ?
#TypeError: list objects are unhashable

#***************  Part 4: 映射类型内建方法 ************
```

#基本的字典方法关注他们的键和值。它们有：keys()方法，返回一个列表，包含字典中所有的
#键，values()方法，返回一个列表，包含字典中所有的值，items(), 返回一个包含所有(键, 值)元
#组的列表。这些方法在不按任何顺序遍历字典的键或值时很有用。

```
#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#print(dict1.keys())
#-->dict_keys(['HomeTown', 'Name', 'Sex'])
#print(dict1.values())
#-->dict_values(['anqing', 'dave', 'Man'])
#print(dict1.items())
#-->dict_items([('HomeTown', 'anqing'), ('Name', 'dave'), ('Sex', 'Man')])

#for keys in dict1.keys():
#       print('the key is: %s, and values is:%s' %(keys,dict1[keys]))
#-->
#the key is: HomeTown, and values is:anqing
#the key is: Name, and values is:dave
#the key is: Sex, and values is:Man
```

#方法名字 操作
#dict.clear() 删除字典中所有元素
#dict.copy() 返回字典(浅复制)的一个副本
#dict.fromkeysc(seq,val=None) 创建并返回一个新字典，以 seq 中的元素做该字典的键，val 做该字典中所有键对应的初始值(如果不提供此值，则默认为 None)
#dict.get(key,default=None) 对字典 dict 中的键 key,返回它对应的值 value，如果字典中不存在此键，则返回 default 的值(注意，参数 default 的默认值为 None)
#dict.has_key(key) 如果键(key)在字典中存在，返回 True，否则返回 False. 在 Python2.2 版本引入 in 和 not in 后，此方法几乎已废弃不用了，但仍提供一个可工作的接口。
#dict.items() 返回一个包含字典中(键, 值)对元组的列表
#dict.keys() 返回一个包含字典中键的列表
#dict.iter() 方法 iteritems(), iterkeys(), itervalues()与它们对应的非迭代方法一样，不同的是它们返回一个迭代子，而不是一个列表。
#dict.popc(key[, default]) 和方法 get()相似，如果字典中 key 键存在，删除并返回 dict[key]，如果 key 键不存在，且没有给出 default 的值，引发 KeyError 异常。
#dict.setdefault(key,default=None) 和方法 set()相似，如果字典中不存在 key 键，由

dict[key]=default 为它赋值。
#dict.update(dict2) 将字典 dict2 的键-值对添加到字典 dict
#dict.values() 返回一个包含字典中所有值的列表

#字典的返回值是无序的，可是使用 sorted()方法进行排序
#dict1={'Name':'dave','Sex':'Man','HomeTown':'anqing'}
#for keys in sorted(dict1):
print('the key is: %s, and values is:%s' %(keys,dict1[keys]))
#-->
#the key is: HomeTown, and values is:anqing
#the key is: Name, and values is:dave
#the key is: Sex, and values is:Man

#update()方法可以用来将一个字典的内容添加到另外一个字典中。字典中原有的键如果与新添
#加的键重复，那么重复键所对应的原有条目的值将被新键所对应的值所覆盖。原来不存在的条目则
#被添加到字典中。clear()方法可以用来删除字典中的所有的条目。

#dict2= {'host':'earth', 'port':80}
#dict3= {'host':'venus', 'server':'http'}
#dict2.update(dict3)
#print(dict2)
#dict3.clear()
#print(dict3)
#-->
#{'host': 'venus', 'port': 80, 'server': 'http'}
#{}

#copy() 方法返回一个字典的副本。注意这只是浅复制。最后要说明，get()方法和键查找(key-lookup)操作符([ ])相似，不同的是它允许你为不存在的
#键提供默认值。如果该键不存在，也未给出它的默认值，则返回 None。此方法比采用键查找(key-lookup)更灵活，因为你不必担心因键不存在而引发异常。

#dict2= {'host':'earth', 'port':80}
#dict3= {'host':'venus', 'server':'http'}
#dict4 = dict2.copy()
#print(dict4)
##get 某个键值
#print(dict4.get('host'))
##get 某个不存在的键值
#print(dict4.get('xxx'))
#print(type(dict4.get('xxx')))

```
##指定不存在键值的默认值
#print(dict4.get('xxx', 'no such key'))
#-->
#{'host': 'earth', 'port': 80}
#earth
#None
#<class 'NoneType'>
#no such key


#setdefault()是自 2.0 才有的内建方法，使得代码更加简洁，它实现了常用的语法：检查字典
#中是否含有某键。 如果字典中这个键存在，你可以取到它的值。 如果所找的键在字典中
不存在，
#你可以给这个键赋默认值并返回此值。这正是执行 setdefault()方法的目的：

#myDict = {'host': 'earth', 'port': 80}
#print(myDict.keys())
#print(myDict.items())
#print(myDict.setdefault('port', 8080))
#print(myDict.setdefault('prot', 'tcp'))
#print(myDict.items())
#-->
#dict_keys(['host', 'port'])
#dict_items([('host', 'earth'), ('port', 80)])
#80
#tcp
#dict_items([('host', 'earth'), ('prot', 'tcp'), ('port', 80)])


#********** Part 5: 字典的键 **********

#字典中的值没有任何限制。 他们可以是任意 Python 对象，即，从标准对象到用户自定义
对象
#皆可。但是字典中的键是有类型限制的。

#不允许一个键对应多个值
#你必须明确一条原则：每个键只能对应一个项。也就是说，一键对应多个值是不允许的。
(像列
#表、元组和其他字典这样的容器对象是可以的。) 当有键发生冲突(即，字典键重复赋值)，
取最后(最近)的赋值。

#dict1 = {'foo':789, 'foo': 'xyz'}
#print(dict1['foo'])
#-->
```

```
#xyz

#Python 并不会因字典中的键存在冲突而产生一个错误，它不会检查键的冲突是因为，如果真这
#样做的话，在每个键-值对赋值的时候都会做检查，这将会占用一定量的内存。

#键必须是可哈希的
#大多数 Python 对象可以作为键；但它们必须是可哈希的对象。像列表和字典这样的可变类型，由于它们不是可哈希的，所以不能作为键。
#
#所有不可变的类型都是可哈希的，因此它们都可以做为字典的键。一个要说明的是问题是数字：
#值相等的数字表示相同的键。换句话来说，整型数字 1 和 浮点数 1.0 的哈希值是相同的，即它们是相同的键。
#
#为什么键必须是可哈希的？解释器调用哈希函数，根据字典中键的值来计算存储你的数据的位
#置。如果键是可变对象，它的值可改变。如果键发生变化，哈希函数会映射到不同的地址来存储数
#据。如果这样的情况发生，哈希函数就不可能可靠地存储或获取相关的数据。选择可哈希的键的原因就是因为它们的值不能改变



#字典的一个小示例：login.py

#encoding=utf-8
#先创建一个用户，保存在字典里，然后用这个用户进行登陆，最后显示 welcome ...

db = {}
def newuser():
    prompt = 'please input your new user name:'

    while True:
        name = input(prompt)
        if name in db:
            prompt = 'name taken, try another: '
            continue
        else:
            break

    pwd = input('plz input passwd: ')
    db[name] = pwd
    print('the user info is: %s' % db)
```

```
def olduser():
        name = input('login: ')
        pwd = input('passwd: ')

        passwd = db.get(name)
        if passwd == pwd:
                print('welcome back: %s' % name)
        else:
                print('login incorrect')

def showmenu():
        prompt = """
  (N)ew User Login
  (E)xisting User Login
  (Q)uit

  Enter choice: """

        done = False
        while not done:
                chosen = False
                while not chosen:
                        try:
                                choice = input(prompt).strip()[0].lower()
                        except (EOFError, KeyboardInterrupt):
                                choice = 'q'

                        print('\nYou picked: [%s]' % choice)

                        if choice not in 'neq':
                                print('invalid option, try again')
                        else:
                                chosen = True
                                done = True
                                newuser()
                                olduser()

if __name__ == '__main__':
        showmenu()


#运行结果：
# (N)ew User Login
```

```
# (E)xisting User Login
# (Q)uit
#
# Enter choice: n
#
#You picked: [n]
#please input your new user name:Dave
#plz input passwd: Dave
#the user info is: {'Dave': 'Dave'}
#login: Dave
#passwd: Dave
#welcome back: Dave
```