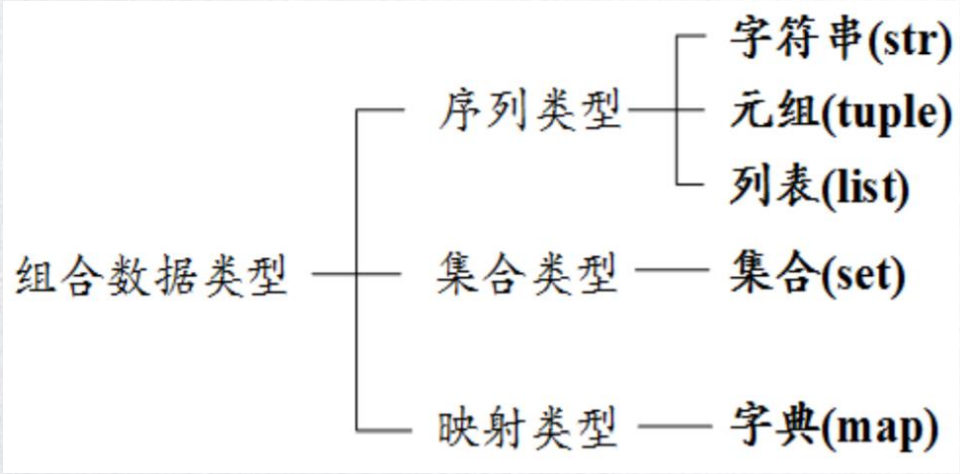
上次课回顾：组合数据类型（一）

- 组合数据类型
  - 组合数据类型概述
    - 序列类型
    - 集合类型
    - 映射类型
  - 列表类型与操作
  - Jieba库的使用与词频统计



列表类型

- 列表（list）是包含0个或多个对象引用的有序序列，属于序列类型。与元组不同，列表的长度和内容都是可变的，可自由对列表中数据项进行增加、删除或替换。列表没有长度限制，元素类型可以不同，使用非常灵活。
- 由于列表属于序列类型，所以列表也支持成员关系操作符（in）、长度计算函数（len()）、分片（[]）。列表可以同时使用正向递增序号和反向递减序号，可以采用标准的比较操作符（<、<=、==、!=、>=、>）进行比较，列表的比较实际上是单个数据项的逐个比较。

列表类型

- 列表用中括号（[]）表示，也可以通过list()函数将元组或字符串转化成列表。直接使用list()函数会返回一个空列表。

```
In [35]: ls = ["信息楼", "RUC", [125, "AI"], 100872, "RUC"]
         ls
Out[35]: ['信息楼', 'RUC', [125, 'AI'], 100872, 'RUC']

In [36]: ls[2][-1][0]
Out[36]: 'A'

In [37]: list(("信息楼", "RUC", (12, "AI"), 100872, "RUC"))
Out[37]: ['信息楼', 'RUC', (12, 'AI'), 100872, 'RUC']

In [38]: list("中国人民大学高瓴人工智能学院")
Out[38]: ['中', '国', '人', '民', '大', '学', '高', '瓴', '人', '工', '智', '能', '学', '院']

In [39]: list()
Out[39]: []
```

列表类型

- 列表必须通过显式的数据赋值才能生成，简单将一个列表赋值给另一个列表不会生成（?)新的列表对象。

```
In [40]: ls = ["信息楼", "RUC", [125, "AI"], 100872, "RUC"]
         lt = ls
         ls[2] = 0
         lt
Out[40]: ['信息楼', 'RUC', 0, 100872, 'RUC']
```

列表类型

- 列表类型的操作

函数或方法	描述
ls[i] = x	替换列表ls第i数据项为x
ls[i: j] = lt	用列表lt替换列表ls中第i到j项数据（不含第j项，下同）
ls[i: j: k] = lt	用列表lt替换列表ls中第i到j以k为步的数据
del ls[i: j]	删除列表ls第i到j项数据，等价于ls[i: j]=[]
del ls[i: j: k]	删除列表ls第i到j以k为步的数据
ls += lt或ls.extend(lt)	将列表lt元素增加到列表ls中
ls *= n	更新列表ls，其元素重复n次
ls.append(x)	在列表ls最后增加一个元素x
ls.clear()	删除ls中所有元素
ls.copy()	生成一个新列表，复制ls中所有元素
ls.insert(i, x)	在列表ls第i位置增加元素x
ls.pop(i)	将列表ls中第i项元素取出并删除该元素
ls.remove(x)	将列表中出现的第一个元素x删除
ls.reverse(x)	列表ls中元素反转

列表类型

- 当使用一个列表改变另一个列表值时，Python不要求两个列表长度一样，但遵循“多增少减”的原则，例子如下。

```
In  [43]: ls = list(range(6))
          ls
Out[43]: [0, 1, 2, 3, 4, 5]

In  [44]: len(ls[3:])
Out[44]: 3

In  [45]: 2 in ls
Out[45]: True

In  [46]: ls[3] = 'replace'
          ls
Out[46]: [0, 1, 2, 'replace', 4, 5]
```

```
[28]: ls[2:4] = ['machine','learning']
      ls
[28]: [0, 1, 'machine', 'learning', 4, 5]

[29]: ls[2:4] = ['machine','learning','AI']
      print(ls)
      [0, 1, 'machine', 'learning', 'AI', 4, 5]

[30]: ls[2:4] = ['AI']
      ls
[30]: [0, 1, 'AI', 'AI', 4, 5]
```

大作业1：

- 给定一个二维卷积核和由图片转换后的矩阵，实现二维卷积操作，观察卷积后的图片所产生的变化

- 提交：
  - 把Jupyter Notebook中的空缺部分填好，并运行得到结果
  - 提交文件

- 提交期限：2周

- 提交平台：未来课堂

关于卷积

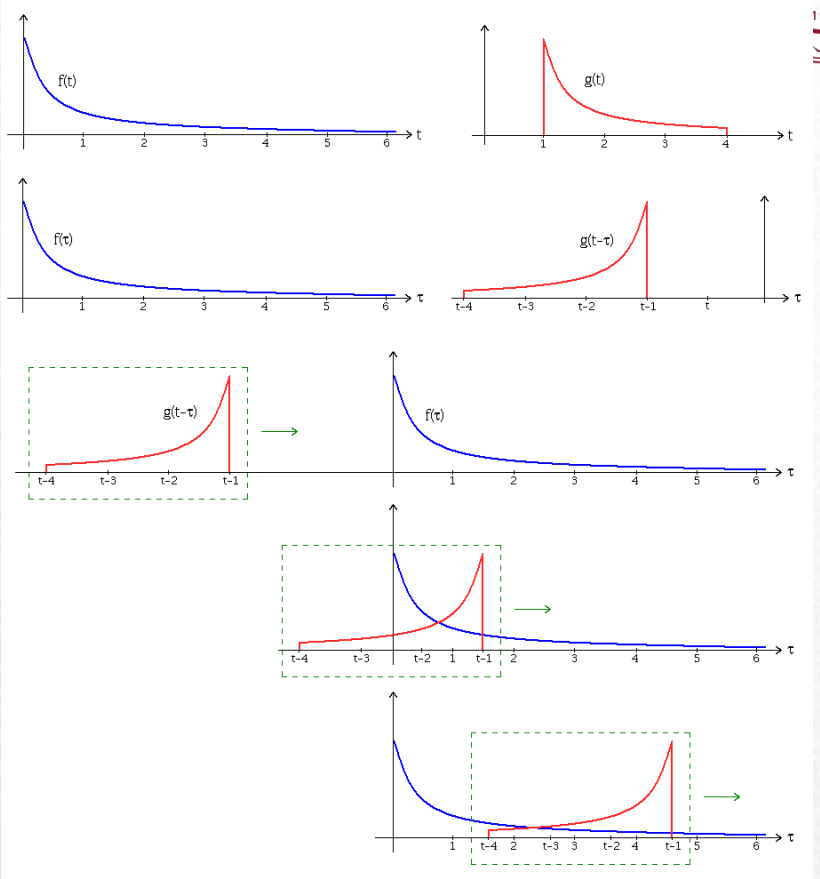
- 什么是卷积?
  - 两个函数f和g生成第三个函数的一种数学算子
  - f与经过翻转和平移的g的乘积函数所围成的曲边梯形的面积(滑动平均)

其连续的定义为:

$$(f*g)(n)=\int_{-\infty}^{\infty}f(\tau)g(n-\tau)d\tau$$

其离散的定义为:

$$(f*g)(n)=\sum_{\tau=-\infty}^{\infty}f(\tau)g(n-\tau)$$



离散卷积举例：2个骰子点数和为4的概率

- 两枚骰子
  - 骰子未必都是公平
    - 第一个骰子出现点i (i = 1, 2, ..., 6) 的概率为f(i)
    - 第二个骰子出现点i的概率为g(i)
  - 把这两枚骰子都抛出去
  - 求2个骰子**点数和为4**的概率
    - $(f * g)(4)$

f	1	2	3	4	5	6

f表示第一枚骰子
$f(1)$表示投出1的概率
$f(2)$、$f(3)$、$\cdots$以此类推

g	1	2	3	4	5	6

g表示第二枚骰子

应用卷积

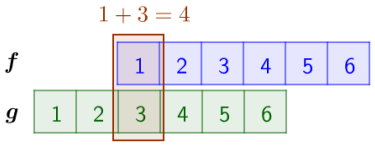
- 关键：两个骰子点数加起来等于
  - $(f * g)(4)$

$$f(1)g(3) + f(2)g(2) + f(3)g(1)$$

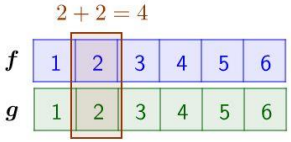
$$(f * g)(4) = \sum_{m=1}^{3} f(4-m)g(m)$$

  - $f * g(k)$：骰子点数和为k的概率

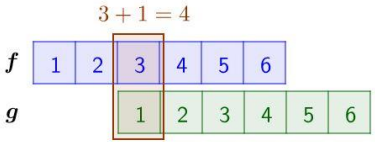
$$f * g(k) = \sum_{m=1}^{k-1} f(k-m)g(m)$$



出现概率为：$f(1)g(3)$



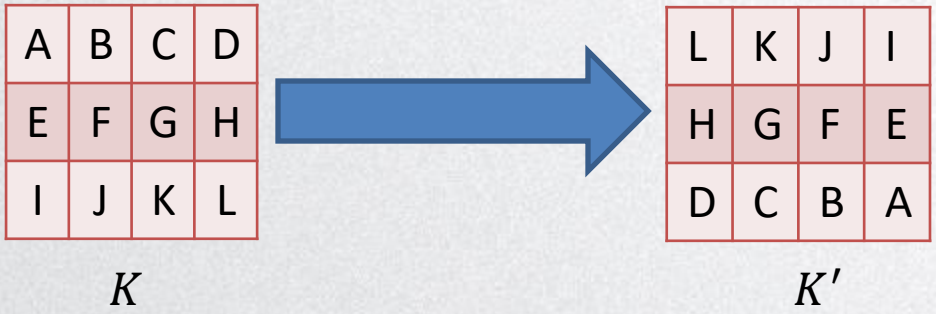
出现概率为：$f(2)g(2)$



出现概率为：$f(3)g(1)$

大作业1

- 步骤1：翻转核矩阵



K K'

  - 输入: 原始矩阵K；输出：翻转后的矩阵K′
  - 方法:
    - 横向翻转 + 纵向翻转

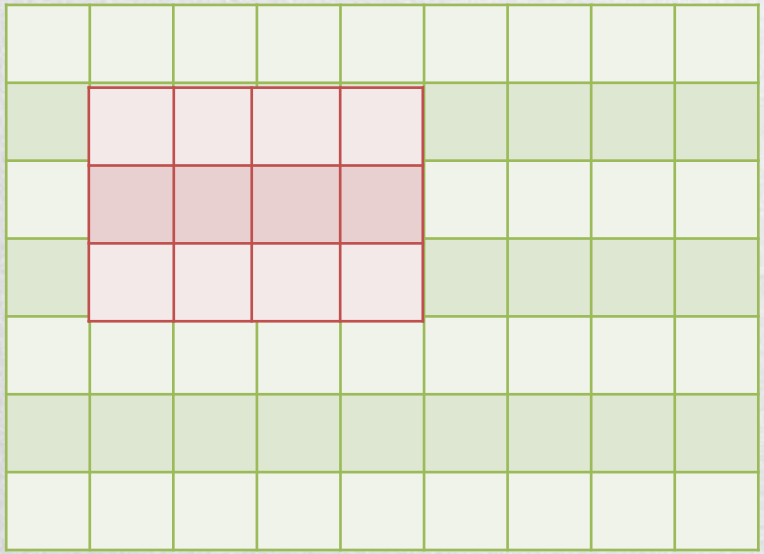
$$K'[i][j] = K[h - i - 1][w - j - 1]$$

    - h:行数
    - w:列数

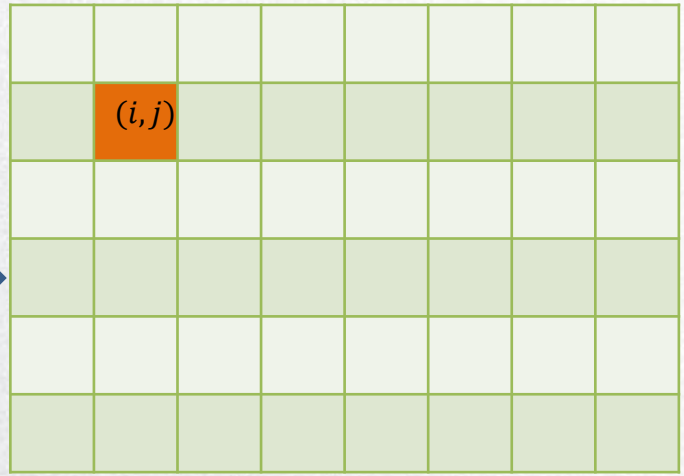
大作业1

- 步骤2：计算卷积
  - 输入：图像矩阵im，翻转后的核矩阵K'
  - 输出：卷积后的图像矩阵im'

$$\mathrm{im'[i][j]} = \sum_{u=0}^{h-1}\sum_{v=0}^{w-1} im[i+u][j+v]\,\mathrm{K'}[u][v]$$



$im_{H\times W}$



im'

大作业1

- 输入：图片im、卷积核k
- 输出：卷积处理后的图片
- 说明：
  - 1. 下载Jupyter Notebook代码和图片
  - 2. 阅读已有的代码（已有代码负责读入图片、转换为二维数值列表以及将二维数值列表转换为图片进行输出）
  - 3. 编写代码，实现conv(im, k, stride)函数，其中stride步长，默认为1
    - 完成k矩阵的翻转操作
    - 完成对im矩阵的卷积操作，返回卷积后的二维数值列表
  - 4. 观察在不同卷积核下输出图片的差异

《人工智能与Python程序设计》—组合数据类型（二）

人工智能与Python程序设计 教研组

教学目标

- 理解字典类型的概念
  - 键和值
- 掌握字典类型的创建方法
  - {}, dict, **kwargs
- 掌握字典类型的操作
  - 索引、遍历、修改……

人工智能与Python程序设计04-1
组合数据类型（二）

提纲

- 字典类型概念与操作
- 组合数据类型的高级操作
- 组合数据类型相关模块

字典类型的概念

- 上节课中我们介绍了序列类型
  - 元组
  - 列表
- 对于序列类型，我们可以使用整数索引来查找元素
  - 类似C语言中的数组

```
[1]: short_names = ['RUC', 'THU', 'PKU', 'BIT', 'BUAA']
     full_names = ['中国人民大学', '清华大学', '北京大学', '北京理工大学', '北京航空航天大学']
     print(short_names[0], full_names[0])
     print(short_names[1], full_names[1])

     RUC 中国人民大学
     THU 清华大学
```

字典类型的概念

- 但很多时候我们需要更加灵活的信息查找方式
  - 例如，我们想通过学校的简称找到学校的全称
    - "RUC"=>"中国人民大学"
    - "THU"=>"清华大学"
    - 需要利用一个字符串，查找另一个字符串
- 上述信息查找，可以看做是一个映射：<key>=><value>
  - <key>：键，用来查找的信息
  - <value>：值，查找到的信息
  - 键-值构成了一个“键值对”(key-value pair)
- Python语言中我们可以使用字典（dict）实现映射

字典类型的概念

- Python语言中我们可以使用字典（dict）实现映射
  - 字典是包含0个或者多个键值对信息的关联数组
  - 关联数组是支持以下操作的抽象数据类型：
    - 向关联数组添加键值对
    - 从关联数组内删除键值对
    - 修改关联数组内的键值对
    - 根据已知的键寻找键值对
  - 注意：字典中的键是唯一的，无法保存一对多的映射
    - "RUC"=>"人民大学"
    - "RUC"=>"中国人民大学"
    - 解决方法"RUC"=>["人民大学"，"中国人民大学"]

简称（键）	全称（值）
RUC	中国人民大学
THU	清华大学
PKU	北京大学
BIT	北京理工大学
……	……

字典类型的创建

- 字典类型的创建方法有以下几种：
  - 使用{}创建

```
[2]: dict1 = {'RUC': '中国人民大学',
              'THU': '清华大学',
              'PKU': '北京大学',
              'BIT': '北京理工大学',
              'BUAA': '北京航空航天大学'} # 用{}创建
     empty_dict = {} # 可以用这个方式创建空字典

     print(dict1)
     print(empty_dict)
```

```
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空
航天大学'}
{}
```

字典类型的创建

- 字典类型的创建方法有以下几种：
  - 使用构造函数dict()创建

[3]:
```
empty_dict = dict() # 利用构造函数创建字典对象
dict2 = dict(dict1) # 利用另一个dict创建，注意这里dict1中的数据会被拷贝一份
dict3 = dict([('RUC', '中国人民大学'),
              ('THU', '清华大学'),
              ('PKU', '北京大学'),
              ('BIT', '北京理工大学'),
              ('BUAA', '北京航空航天大学')]) # 利用包含 键值对 的序列对象创建
dict4 = dict((['RUC', '中国人民大学'],
              ['THU', '清华大学'],
              ['PKU', '北京大学'],
              ['BIT', '北京理工大学'],
              ['BUAA', '北京航空航天大学'])) # 元组也是序列对象；键值对 可以是列表

print(empty_dict)
print(dict2)
print(dict3)
print(dict4)
```

```
{}
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空
航天大学'}
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空
航天大学'}
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空
航天大学'}
```

字典类型的创建

- 字典类型的创建方法有以下几种：
  - 使用dict()和基于键值对的参数来构建

[4]:
```
# 一种"特殊"的创建方式
dict5 = dict(RUC='中国人民大学',
             THU='清华大学',
             PKU='北京大学',
             BIT='北京理工大学',
             BUAA='北京航空航天大学')
dict6 = dict(dict1, FDU='复旦大学') # 还可以和其他创建方式一起使用

print(dict5)
print(dict6)
```

```
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学', 'FDU': '复旦大学'}
```

字典类型的创建

- Python语言提供了一种基于字典的灵活的给函数传递参数的方式
  - 不需在定义函数的时候指定参数的个数和名称
  - 参数会以字典的形式被传递到函数内

[6]:
```
def g(**kwargs):
    print(type(kwargs))
    print(kwargs)

g()
g(RUC='中国人民大学', THU='清华大学')
```

```
<class 'dict'>
{}
<class 'dict'>
{'RUC': '中国人民大学', 'THU': '清华大学'}
```

字典类型的操作

- 利用键索引字典中保存的值：
  - 可以使用[]操作符访问字典中保存的值
  - 若键不存在，会报错并抛出异常

```
[7]: print(dict1['RUC'])
     print(dict1['FDU']) # 若键信息不在字典里，会报错
```

```
中国人民大学
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-227324f668a8> in <module>
      1 print(dict1['RUC'])
----> 2 print(dict1['FDU']) # 若键信息不在字典里，会报错

KeyError: 'FDU'
```

字典类型的操作

- 利用键索引字典中保存的值：
  - 可以使用get()方法来访问字典中保存的值

```
[8]: print(dict1.get('RUC')) # 也可以使用get方法
     print(dict1.get('FDU', '名称未知')) # get方法的第二个参数是若键信息不在字典里时返回的默认值信息
```

```
中国人民大学
名称未知
```

  - 可以使用 <key> in <d>表达式判断键是否在字典中

```
[9]: print('RUC' in dict1)
     print('FDU' in dict1)
```

```
True
False
```

字典类型的操作

- 通过键增加、修改值信息和删除相应的键值对：
  - 可以使用[]来增加、修改和删除字典中保存的信息

[10]:
```
dict2 = dict(dict1)
print(dict2)
dict2['FDU'] = '复旦大学'
dict2['RUC'] = 'Renmin University of China'
print(dict2)
```

```
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'RUC': 'Renmin University of China', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学', 'FDU': '复旦大学'}
```

[11]:
```
dict2 = dict(dict1)
print(dict2)
del dict2['RUC']
print(dict2)
```

```
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
```

字典类型的操作

- 获取一个字典中的所有键、值、以及键值对：
  - <d>.keys(), <d>.values(), <d>.items()

```
[13]: dict2 = dict(dict1)
      keys = dict2.keys()
      print(keys)
      values = dict2.values()
      print(values)
      items = dict2.items()
      print(items)

      dict_keys(['RUC', 'THU', 'PKU', 'BIT', 'BUAA'])
      dict_values(['中国人民大学', '清华大学', '北京大学', '北京理工大学', '北京航空航天大学'])
      dict_items([('RUC', '中国人民大学'), ('THU', '清华大学'), ('PKU', '北京大学'), ('BIT', '北京
      理工大学'), ('BUAA', '北京航空航天大学')])
```

字典类型的操作

- 获取一个字典中的所有键、值、以及键值对：
  - <d>.keys(), <d>.values(), <d>.items()
  - 需要注意的是，上述方法获得均为字典中相应信息的一个视图（view），即如果我们修改了字典中的内容，上述视图对象也会相应变化

```
[14]: del dict2['RUC']
      print(keys)
      print(values)
      print(items)

      dict_keys(['THU', 'PKU', 'BIT', 'BUAA'])
      dict_values(['清华大学', '北京大学', '北京理工大学', '北京航空航天大学'])
      dict_items([('THU', '清华大学'), ('PKU', '北京大学'), ('BIT', '北京理工大学'), ('BUAA', '北京
      航空航天大学')])
```

字典类型的操作

- 使用for … in … 语句遍历字典

```
[15]: dict2 = dict(dict1)
for key in dict2: # 该语句会遍历字典中所有键
    print(key)

for key, value in dict2.items(): # 这样可以遍历所有键值对
    print(key, value)
```

```
RUC
THU
PKU
BIT
BUAA
RUC 中国人民大学
THU 清华大学
PKU 北京大学
BIT 北京理工大学
BUAA 北京航空航天大学
```

字典类型的操作

- 其他一些字典类型的相关操作：

[32]:
```
dict2 = dict(dict1)
print(len(dict2)) # 字典的大小（包含键值对的个数）

dict2.clear() # 清空字典
print(dict2)

dict2 = dict1.copy() # 返回一个字典的拷贝
print(dict2)

dict2.update({'RUC': 'Renmin University of China',
              'THU': 'Tsinghua University'}) # 批量更新字典中键值对
print(dict2)

print(dict2.pop('PKU')) # 按键查找，然后删除相应键值对
print(dict2)
```

```
5
{}
{'RUC': '中国人民大学', 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA':
'北京航空航天大学'}
{'RUC': 'Renmin University of China', 'THU': 'Tsinghua University', 'PKU': '北京大学', 'B
IT': '北京理工大学', 'BUAA': '北京航空航天大学'}
北京大学
{'RUC': 'Renmin University of China', 'THU': 'Tsinghua University', 'BIT': '北京理工大
学', 'BUAA': '北京航空航天大学'}
```

词频统计-字典方法

- 字典方法词频统计：

```
[25]: import jieba
txt = open('./高瓴人工智能学院简介.txt', 'r', encoding='utf-8').read()
words = jieba.lcut(txt)
counts = {}
for word in words:
    if len(word) == 1: #排除单个字符的分词结果
        continue
    else:
        counts[word] = counts.get(word, 0) + 1
items = list(counts.items())
items.sort(key=lambda x: x[1], reverse=True)
for i in range(15):
    word, count = items[i]
    print('{0:<10}{1:>5}'.format(word, count))
```

人工智能	13
学院	8
一流	5
中国人民大学	4
未来	3
高瓴	3
院长	3
打造	3
全球	3
研究	3
联合	3
时代	2
影响	2
技术	2
发展	2

思考：什么信息可以作为字典的值和键

- 基本数据类型、组合数据类型、甚至是函数，均可以作为字典的值
  - 所有的对象（object）均可以作为字典的值

[65]:
```
dict2 = dict(dict1)
dict2['RUC'] = 1 # 基本数据类型
print(dict2)

dict2['RUC'] = ('中国人民大学', 'Renmin University of China') # 组合数据类型，元组
print(dict2)
dict2['RUC'] = ['中国人民大学', 'Renmin University of China'] # 组合数据类型，列表
print(dict2)

# 字典可以嵌套
dict2['RUC'] = {'中文名': '中国人民大学', '英文名': 'Renmin University of China'}
print(dict2)
print(dict2['RUC']['中文名'])
```

```
{'RUC': 1, 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'RUC': ('中国人民大学', 'Renmin University of China'), 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'RUC': ['中国人民大学', 'Renmin University of China'], 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
{'RUC': {'中文名': '中国人民大学', '英文名': 'Renmin University of China'}, 'THU': '清华大学', 'PKU': '北京大学', 'BIT': '北京理工大学', 'BUAA': '北京航空航天大学'}
中国人民大学
```

思考：什么信息可以作为字典的值和键

- 基本数据类型、组合数据类型、甚至是函数，均可以作为字典的值
  - 所有的对象（object）均可以作为字典的值

[64]:
```
# 字典的值还可以是一个函数
def output_ruc():
    print('中文名: 中国人民大学\t英文名: Renmin University of China')
dict2['RUC'] = output_ruc
print(dict2)
dict2['RUC']() # 在一个函数后加上括号就可以调用这个函数!
```

```
{'RUC': <function output_ruc at 0x7f1249ff8ca0>, 'THU': '清华大学', 'PKU': '北京大学', 'BI
T': '北京理工大学', 'BUAA': '北京航空航天大学'}
中文名: 中国人民大学        英文名: Renmin University of China
```

利用字典实现一个简单的计算器

- 函数可以作为字典的值
  - 利用字典将数据映射到相应的操作（函数）

[22]:
```
import math
oper_dict = {
    '+': lambda x, y: x + y,
    '-': lambda x, y: x - y,
    '*': lambda x, y: x * y,
    '/': lambda x, y: x / y,
    'sin': lambda x: math.sin(x),
    'exp': lambda x: math.exp(x),
    'ln': lambda x: math.log(x)
}

while(True):
    s = input('请输入一个前缀表达式，运算符和数字间用空格分开(输入空字符串退出)：')
    if len(s) == 0:
        break
    tokens = s.split(' ')
    operation = oper_dict[tokens[0]]
    operands = []
    for token in tokens[1:]:
        operands.append(float(token))
    print('计算结果:{0:.4f}'.format(operation(*operands)))
```

```
请输入一个前缀表达式，运算符和数字间用空格分开(输入空字符串退出)： + 1 2
计算结果:3.0000
请输入一个前缀表达式，运算符和数字间用空格分开(输入空字符串退出)： sin 3.14159
计算结果:0.0000
请输入一个前缀表达式，运算符和数字间用空格分开(输入空字符串退出)： exp 1
计算结果:2.7183
请输入一个前缀表达式，运算符和数字间用空格分开(输入空字符串退出)：
```

思考：什么信息可以作为字典的值和键

- 然而，并不是所有的对象均可以作为字典的键

[23]:
```
new_dict = {}
new_dict[1] = 1 # 可以用基本数据类型作为字典的键
new_dict['RUC'] = '中国人民大学' # 可以用字符串作为字典的键
new_dict[('RUC', 'GSAI')] = '中国人民大学高瓴人工智能学院' # 可以用元组作为字典的键
# 甚至可以用嵌套的元组作为字典的键
new_dict[(('RUC', 'GSAI'), 'address')] = '北京市海淀区中关村大街59号中国人民大学'
print(new_dict)
```

```
{1: 1, 'RUC': '中国人民大学', ('RUC', 'GSAI'): '中国人民大学高瓴人工智能学院', (('RUC', 'GSA
I'), 'address'): '北京市海淀区中关村大街59号中国人民大学'}
```

[24]:
```
new_dict[['THU', 'DCST']] = '清华大学计算机科学与技术系' # 然而使用列表作为字典的键会报错
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-91b83864798a> in <module>
----> 1 new_dict[['THU', 'DCST']] = '清华大学计算机科学与技术系' # 然而使用列表作为字典的键会报错

TypeError: unhashable type: 'list'
```

[25]:
```
new_dict[set(['THU', 'DCST'])] = '清华大学计算机科学与技术系' # 集合也不行
```

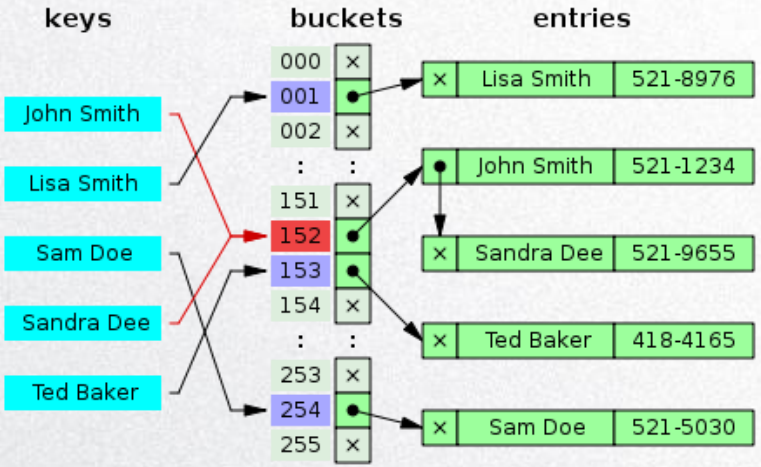
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-997e76ee367d> in <module>
----> 1 new_dict[set(['THU', 'DCST'])] = '清华大学计算机科学与技术系' # 集合也不行

TypeError: unhashable type: 'set'
```

思考：什么信息可以作为字典的值和键

- 那么，到底哪些对象可以作为字典的键呢？
  - 简单来说，所有一旦创建就不能被修改（immutable）的对象，都可以用作字典的键，例如：
    - 基本数据类型
    - 字符串（是的，字符串创建后就不能修改了）
    - 元组
  - 而创建后可以修改的（mutable）对象，都不能用作字典的键，例如：
    - 列表
    - 集合
    - 字典
  - 为什么？

思考：什么信息可以作为字典的值和键

- Python中的字典是基于哈希表（Hash table，又叫散列表）实现的
- 字典保存和索引元素（键值对）的原理：
  - 在将一个键值对添加到字典中时，会先调用键对象的哈希函数，计算哈希值
    - 哈希值通常为一个整数
  - 字典会将键的哈希值相同的元素保存在同一个“桶”（bucket）里
    - 一个桶里保存的元素的键的哈希值相等
  - 在索引时，会调用查询的键的哈希函数计算哈希值，找到相应的桶
    - 即找到所有与查询键哈希值相同的元素
    - 注意：不相等对象的哈希值可能相等（哈希冲突）
  - 再遍历桶中元素的键，调用__eq__()函数，判断其是否和查询相等
    - 相等：找到与查询相等的键，索引成功；
    - 均不相等：找不到查询对应的元素
- 思考：
  - 这样做是正确的的吗？
  - 这样做的目的是什么？



https://en.wikipedia.org/wiki/Hash_table

思考：什么信息可以作为字典的值和键

- 更准确的答案：
  - 所有可哈希（hash）的，即实现了__hash__()和__eq__()两个特殊方法的对象，可以作为字典的键
- Python语言要求：
  - 如果两个对象相等(a.__eq__(b) 或者 a == b返回True），那么他们的__hash__()函数返回值必须相等
  - 只有不可变对象才是可哈希的，才有__hash__()函数
    - 为什么?
- 哈希函数需要满足两个性质：
  - 哈希函数的计算应该比较高效
  - 如果两个对象不相等，那么他们的哈希值相等的可能性很低
    - 哈希值的分布尽可能是“均匀”的

谢谢！