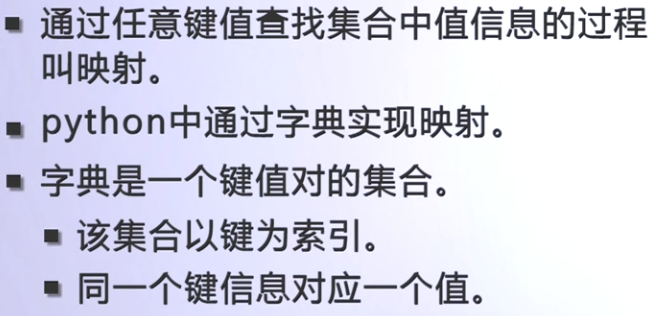
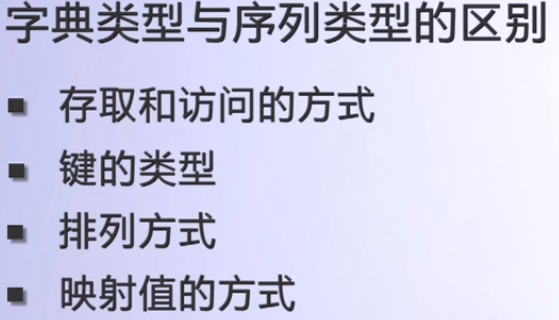
Python中字典数据类型及集合介绍

# 什么是字典？



Python中**只有一个映射类型**，就是**dict**。与集合相似。

# 字典类型与序列类型的区别



# 对字典的操作

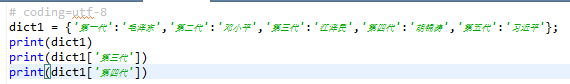
## 创建字典的方法

### 创建一个空字典：

dict1 = {}  
dict2 = dict()

### 初始化定义：第一种定义方式

d2 = {'name':'zhaohong'}

d1 = {}  
print(type(d1))#<class 'dict'>  
d2 = {'name':'zhaohong'}  
d3 = {'name':'zhaohong','age':24}  
print(type(d2))#<class 'dict'>

注意：用**大括号和冒号**来定义，调用的时候用中括号，**冒号前是键key，冒号后是值value。**



### 初始化定义：第二种定义方式：利用dict.fromkeys(seq,value)

Python 字典(Dictionary) fromkeys() 函数**用于创建一个新字典**，以序列seq中元素做字典的键，value为字典所有键对应的初始值。

fromkeys()方法语法：dict.fromkeys(seq[, value]))

参数：seq -- 字典键值列表。

value -- 可选参数, 设置键序列（seq）的值。



**若没有value，则自动会添加None。**



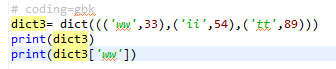


seq = ['name','age','sex']  
dict1 = dict.fromkeys(seq)  
dict2 = dict.fromkeys(seq,10)  
print(dict1)#{'name': None, 'age': None, 'sex': None}  
print(dict2)#{'name': 10, 'age': 10, 'sex': 10}

### 初始化定义：第三种定义方式：

利用**dict()函数**：

dict4 = dict((('key1','value1'),('key2','value2')))  
print(dict4)#{'key1': 'value1', 'key2': 'value2'}



**相当于参数是一个元组，所以需要两个括号，当然用list也可以**。



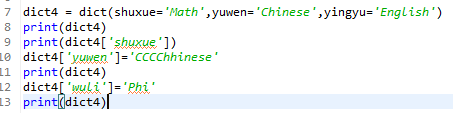
### 初始化定义：第四种定义方式：利用等号=。

注意：此时键已经默认为字符串，所以不能再加引号了。**注意这种方式会对键值对进行随机打乱，没有顺序。**



dict2 = dict(key1='value1',key2='value2')  
print(dict2)#{'key1': 'value1', 'key2': 'value2'}

**修改和添加新的键值对**：方式一样，如果存在该键，就修改，如果不存在，就是添加。



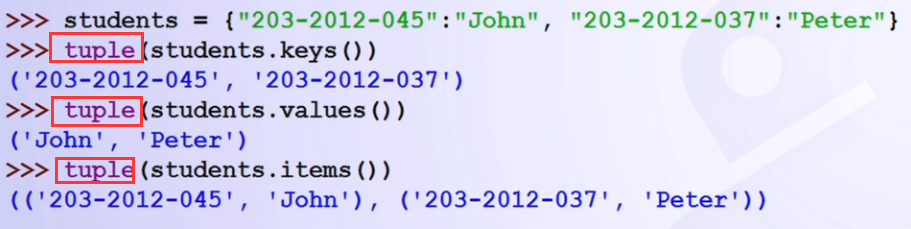
## 字典的常用方法



### keys()、values()、items()

**返回对应列表**。

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
print(type(d3.keys()))#<class 'dict\_keys'>  
print(d3.keys())#dict\_keys(['name', 'age', 'sex'])  
print(type(d3.values()))#<class 'dict\_values'>  
print(d3.values())#dict\_values(['zhaohong', 24, 'man'])  
print(type(d3.items()))#<class 'dict\_items'>  
print(d3.items())#dict\_items([('name', 'zhaohong'), ('age', 24), ('sex', 'man')])



### clear()方法：清空整个字典

**有些利用 dict1={}方法清空**，但是不一样，这样是使dict1指向了一个空的字典，但是**原来的字典没有清空**。建议使用clear()方法。

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
d3.clear()  
print(type(d3))#<class 'dict'>  
print(d3)#{}

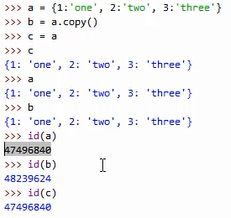
### dict1.update(dict2)方法：根据字典dict2进行更新

dict1.update(dict2) 功能：根据dict2对dict1进行更新。

**在dict1中**，若存在dict2中的键，则就会把dict1中该键值对按照dict2中的value进行更新；若不存在dict2中的键，则就会在dict1中添加该键值对。**dict2**不发生变化。

d3 = {'nickname':'fuyuan','age':24,'sex':'man','addr':'Beijing'}  
d2 = {'name':'zhaoxiyuan','age':22,'sex':'man'}  
d2.update(d3)  
print(d2)#{'name': 'zhaoxiyuan', 'age': 24, 'sex': 'man', 'nickname': 'fuyuan', 'addr': 'Beijing'}  
print(d3)#{'nickname': 'fuyuan', 'age': 24, 'sex': 'man', 'addr': 'Beijing'}

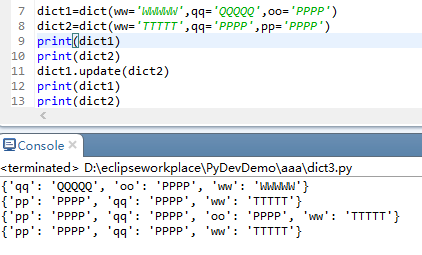
### copy()方法



通过copy方法，就可以判断出copy方法，是把内容拷贝，而赋值的方式，只是把**地址复制**，**指向同一个地址而已**。

### dict1.update(dict2)方法：

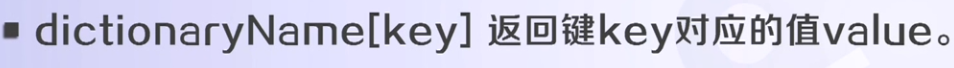
以**参数字典中**的键值对为依照，如果原来的字典中不存在键值对，则就添加，如果存在，就**按照参数字典中的键值对，进行修改**。参数字典中的键值对不会发生改变。



## 获取键key对应的value

### 利用get(key)方法：获取key对应的值。





注意：字典的整体利用**大括号**定义，获取某个键对应的值时，利用**中括号**。

**如果字典中存在该键key，则返回value；如果不存在key，则返回None，即什么都没有。可以在get方法中指定一个缺省value，如果没有该键时，就返回设定的value。**

**get(key,default)**



dict1 = {'name':'zhaohong'}  
print(dict1.get('name','fuyuan'))#zhaohong  
print(dict1.get('age',18))#18

### 利用dictName[key]方法

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
print('name'+' : '+d3['name'])  
print('name'+' : '+d3.get('name'))

### 利用pop(key)方法：取出并删除字典中key对应的键值对



pop方法和get方法一样，可以设置**默认值default**。

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
print('name'+' : '+d3.pop('name')) #name : zhaohong  
print(str(d3)) #{'age': 24, 'sex': 'man'}

print(d3.pop('addr','Beijing'))#Beijing

### popitem()方法：无参数，依次取出item，并删除。

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
print(d3.popitem())#('sex', 'man')  
print(d3.popitem())#('age', 24)  
print(d3.popitem())#('name', 'zhaohong')

## get(key,default)与pop(key,default)方法对比

**get(key,** default**)** ：只是从字典中获取key对应的value；

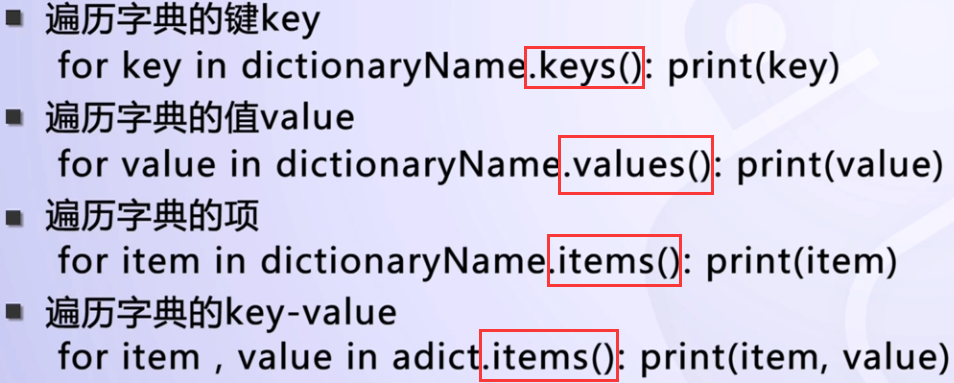
**pop(key,** default**)** : 返回key对应的value的同时，删除该键值对。

## 删除字典中对应的一项



d3 = {'name':'zhaohong','age':24}  
print(d3) #{'name': 'zhaohong', 'age': 24}  
del d3['age']  
print(d3) #{'name': 'zhaohong'}

## 对字典的遍历



示例：

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
for key in d3.keys():  
 print(key+" : " + str(d3[key]))  
for value in d3.values():  
 print(value)  
for item in d3.items():  
 print(item)  
for key,value in d3.items():  
 print(key+" : "+ str(value))

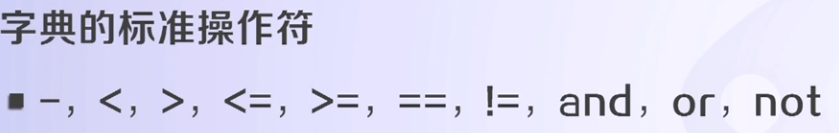
# 利用in 或not in判断键key或值value是否存在

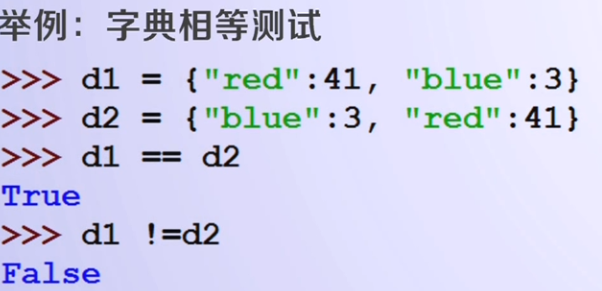
**key in dictName**  就是用来判断某个键是否存在的。对于value的判断是否存在，利用value in d3.values()

**‘key’ in d3 与 ‘key’ in d3.keys() 等价**。

d3 = {'name':'zhaohong','age':24,'sex':'man'}  
print('name' in d3) #True  
print('name' in d3.keys()) #True  
  
print('name' not in d3) #False  
if 'age' in d3:  
 print(d3['age']) #24  
print('zhaohong' in d3.values()) #True

# 字典的标准操作符





# 将json字符串转换成字典dict

注意：**json字符串**中的key与value若为**字符串类型**，必须用**双引号**括起来。

json中，value可以为整数类型，不加双引号。

**将json字符串转换为dict字典类型，利用json库中的loads方法**。

json库中的load方法是从文件中读取json字符串，进行转换成dict，具有参数fp。

示例1：

import json  
# str1 = "{'use':'zhaohong','password':'zhao12'}"#这种错误，解析不出来  
str2 = '{"username":"zhaohong","password":"zhao12"}'  
d = json.loads(str2)  
print(type(d))  
print("username : "+d['username'])  
print("password : "+d['password'])

**示例2** ：多重嵌套，json中某个键对应的也是个json字符串，此时，利用loads转换后，字典中包含字典，及键对应的值也是个字典类型。

import json  
data\_field='{"name":"starmaster","end\_time":"1735660800","category\_id":104,' \  
 '"slot\_no":"1","title":"Python","intro":"AAAAAAAAAA",' \  
 '"intro\_url":"http:\/\/tieba.baidu.com\/f?kw=Python;ie=utf-8",' \  
 '"price":0,"value":"1","sprite":{"1":"1502435970","2":"1502435970",' \  
 '"3":"1502435970","4":"1502435970","5":"1502435970","6":"1502435970"}}'  
dataDict = json.loads(data\_field)  
print(type(dataDict))  
print("title : "+dataDict['title'])  
print(type(dataDict['sprite']))  
print("sprite : "+ str(dataDict['sprite']))  
print(dataDict['sprite']['6'])

# 将字典转换成json字符串

## 方法介绍：

**利用json库中的dumps方法**。

同理，**json库中的dump方法也是从文件中读取进行转换，具有参数fp**。

import json  
# str1 = "{'use':'zhaohong','password':'zhao12'}"#这种错误，解析不出来  
str2 = '{"username":"zhaohong","password":"zhao12"}'  
d = json.loads(str2)  
print(type(d))#<class 'dict'>  
print("username : "+d['username'])  
print("password : "+d['password'])  
  
strJson = json.dumps(d)  
print(type(strJson)) #<class 'str'>  
print(strJson) #{"username": "zhaohong", "password": "zhao12"}

## 注意编码：dumps方法参数

import json

json.dumps(**dict(item)**,**ensure\_ascii=False**)

## 编码解释：

**json.dumps**在默认情况下，对于**非ascii字符**生成的是**相对应的字符编码**，**而非原始字符**，例如：

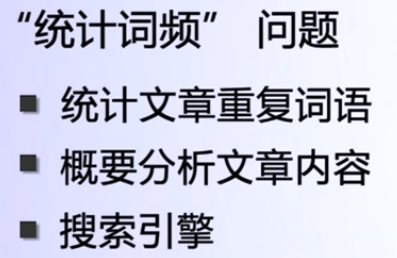
>>> import json  
>>> **js = json.loads('{"haha": "哈哈"}')**  
>>> **print json.dumps(js)**  
{"haha": "\u54c8\u54c8"}

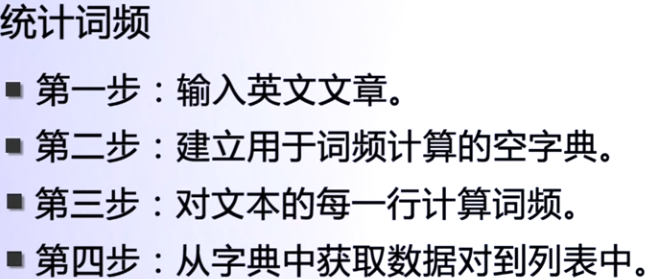
解决办法很简单:

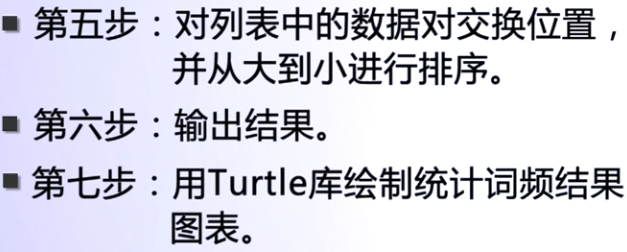
>>> print json.dumps(js, **ensure\_ascii=False**)     
{"haha": "哈哈"}

# 字典使用实例：

## 统计词频





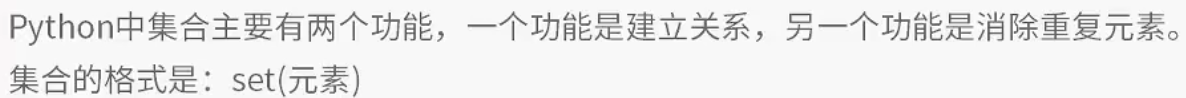


import turtle  
  
##全局变量##  
# 词频排列显示个数  
count = 10  
# 单词频率数组-作为y轴数据  
data = []  
# 单词数组-作为x轴数据  
words = []  
# y轴显示放大倍数-可以根据词频数量进行调节  
yScale = 6  
# x轴显示放大倍数-可以根据count数量进行调节  
xScale = 30  
  
  
################# Turtle Start ####################  
# 从点(x1,y1)到(x2,y2)绘制线段  
def drawLine(t, x1, y1, x2, y2):  
 t.penup()  
 t.goto(x1, y1)  
 t.pendown()  
 t.goto(x2, y2)  
  
  
# 在坐标(x,y)处写文字  
def drawText(t, x, y, text):  
 t.penup()  
 t.goto(x, y)  
 t.pendown()  
 t.write(text)  
  
  
def drawGraph(t):  
 # 绘制x/y轴线  
 drawLine(t, 0, 0, 360, 0)  
 drawLine(t, 0, 300, 0, 0)  
  
 # x轴: 坐标及描述  
 for x in range(count):  
 x = x + 1 # 向右移一位,为了不画在原点上  
 drawText(t, x \* xScale - 4, -20, (words[x - 1]))  
 drawText(t, x \* xScale - 4, data[x - 1] \* yScale + 10, data[x - 1])  
 drawBar(t)  
  
  
# 绘制一个柱体  
def drawRectangle(t, x, y):  
 x = x \* xScale  
 y = y \* yScale # 放大倍数显示  
 drawLine(t, x - 5, 0, x - 5, y)  
 drawLine(t, x - 5, y, x + 5, y)  
 drawLine(t, x + 5, y, x + 5, 0)  
 drawLine(t, x + 5, 0, x - 5, 0)  
  
  
# 绘制多个柱体  
def drawBar(t):  
 for i in range(count):  
 drawRectangle(t, i + 1, data[i])  
 ################# Turtle End ####################  
  
  
# 对文本的每一行计算词频的函数  
def processLine(line, wordCounts):  
 # 用空格替换标点符号  
 line = replacePunctuations(line)  
 # 从每一行获取每个词  
 words = line.split()  
 for word in words:  
 if word in wordCounts:  
 wordCounts[word] += 1  
 else:  
 wordCounts[word] = 1  
  
  
# 空格替换标点的函数  
def replacePunctuations(line):  
 for ch in line:  
 if ch in "~@#$%^&\*()\_-+=<>?/,.:;{}[]|\'""":  
 line = line.replace(ch, " ")  
 return line  
  
  
def main():  
 # 用户输入一个文件名  
 filename = input("enter a filename:").strip()  
 infile = open(filename, "r")  
  
 # 建立用于计算词频的空字典  
 wordCounts = {}  
 for line in infile:  
 processLine(line.lower(), wordCounts)  
  
 # 从字典中获取数据对  
 pairs = list(wordCounts.items())  
  
 # 列表中的数据对交换位置,数据对排序  
 items = [[x, y] for (y, x) in pairs]  
 items.sort()  
  
 # 输出count个数词频结果  
 for i in range(len(items) - 1, len(items) - count - 1, -1):  
 print(items[i][1] + "\t" + str(items[i][0]))  
 data.append(items[i][0])  
 words.append(items[i][1])  
  
 infile.close()  
  
 # 根据词频结果绘制柱状图  
 turtle.title('词频结果柱状图')  
 turtle.setup(900, 750, 0, 0)  
 t = turtle.Turtle()  
 t.hideturtle()  
 t.width(3)  
 drawGraph(t)  
  
#调用main()函数  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Python中的set集合

## Set集合是无序的，不能重复的。

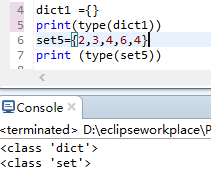
**加入每个元素之前，必须检验垓元素是否是唯一的，若是唯一的，则就添加进去，否则，不能添加。**



## 集合：创建方式：

### {1,3,5}



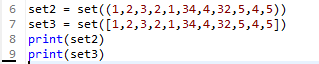


**若是空的，默认为字典dict，若不是空的，则为集合set。**

### 利用set（）工厂函数



**set()函数**内部只可以传入一个参数，可以是字符串，也可以是列表等。



set1 = set('abcd')  
set2 = set('cdef')  
print(set1)#  
print(set2)#  
print(set1&set2)#求交集，返回一个集合{'d', 'c'}  
print(set1|set2)#求并集，返回一个集合{'f', 'a', 'd', 'b', 'e', 'c'}

print(set1-set2)#求差集，返回一个集合{'a', 'b'}

### frozenset:冻结一个集合

定义一个冻结的set集合：就是不可以修改的。

利用frozenset关键词。

## 集合中的元素必须是唯一的。

**应用：可以用来出去列表中的重复元素。**

str1 = 'aaabbbcccddd'  
print(set(str1))#{'c', 'b', 'd', 'a'}



