PYTHON程序设计

计算机学院 王纯

四 组合数据类型

四 组合数据类型

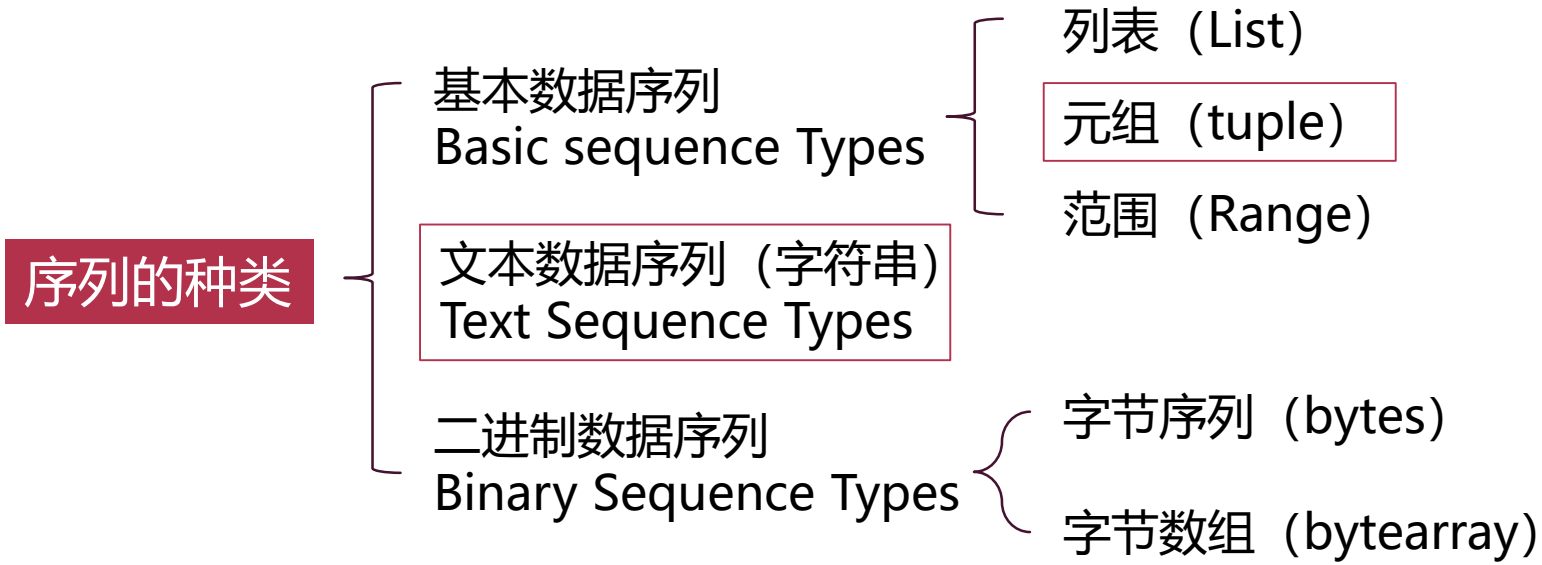
- 组合数据类型概述
- 序列
- 集合
- 字典
- 综合实例

组合数据类型概述

- 计算机不仅对单个数据进行处理， 更多情况下还需要对一组数据进行处理。 如：
  - ✓ 给定一组单词{python, data, function, list, loop}， 计算并输出每个单词的长度；
  - ✓ 给定一个学院学生信息， 统计一下男女生比例；
  - ✓ 一次实验产生了很多组数据， 对这些数据进行分析；
- 组合数据类型能够将多个同类型或不同类型的数据组织起来， 通过单一的表示使数据操作更有序更容易
- Python常见的组合数据类型有三种：序列、集合、字典
  - ✓ 序列类型是一个元素向量， 元素之间存在先后关系， 通过序号访问， 元素之间不排他。
  - ✓ 集合类型是一个元素集合， 元素之间无序， 相同元素在集合中唯一存在。
  - ✓ 字典类型是“键-值” 数据项的组合， 每个元素是一个键值对， 表示为(key, value)。

序列

若干个有序数据组成序列



序列

元组是包含0个或多个数据项的不可变序列类型。 元组生成后是固定的， 其中任何数据项不能替换或删除

列表是一个可以修改数据项的序列类型， 使用也最灵活

列表 List

长度可变

元素可谓多种类型

a = [1,8.0,” abc” ,obj]

VS

数组 array

长度不可变

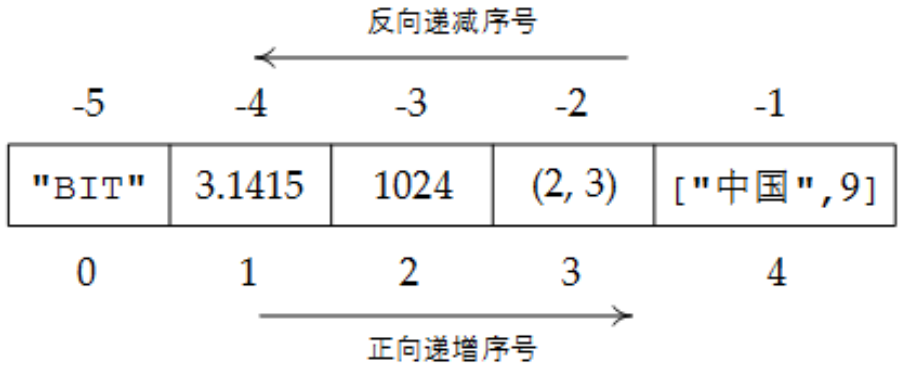
元素必须为同一种类型

int a[5] = {1,2,3} ;

序列的基本操作

- ✓ 切片操作
- ✓ 索引访问
- ✓ 重复操作
- ✓ 求序列长度
- ✓ 成员关系操作
- ✓ 比较运算操作 连接操作
- ✓ 求序列统计信息（例如最大值）

适用于处理动态数据集，尤其是读操作远多于写操作的场景



序列的双向索引

回忆字符串操作，重新认识序列操作

```
s = 'abc'
>>>s[0]  #索引访问
'a'
>>>s[0:2]  #切片操作
'ab'
>>>s + 'd'  #链接操作
'abcd'
>>>s*3   #重复操作
'abcabcabc'
>>>'a' in s   #成员关系操作
True
>>>len(s)   #求序列长度
3
>>>s=='abc'   #比较运算
True
>>>s.count('a')   #序列统计信息
1
```

元组

- ✓ 0个或多个数据的有序序列，用小括号括起来。
- ✓ 元组是不可变数据类型。
- ✓ 元组中的项，数据类型可以不一致。
- ✓ 序列的基本操作，元组都支持

创建元组：

```
a=(1,2,3); #创建tuple
a=1,2,3; #括号可以省略
a=tuple([1,2,3]); #只有一个元素，列表（1,2,3）
a=tuple(); #空元组
a=();
```

索引和切片操作

```
tup1= (0,1,2)
tup1[0] #0
tup1[0:2] #0,1
tup1[1]=5 #错误，不允许修改元组的元素值
```

连接操作

```
tup2=4,5,6
tup3=tup1+tup2 #(0,1,2,4,5,6)
```

其它操作

```
len(tup3) #6
max(tup3) #6
for x in (1, 2, 3): print(x) #1 2 3
```

列表-创建

- ✓ **列表**由一组有序数据组成的序列。列表是可变数据类型。列表长度和内容可变。多个列表项的数据类型可以不一致。
- ✓ 列表与元组可以互相转换生成。
- ✓ 列表可以嵌套。索引也可以使用多重下标。

```
a = [1,8.0,“abc”,obj] #a为列表,obj为其它类型的对象。
b =list((1,8.0,“abc”,obj)) #b为列表,从元组生成
a = tuple(b)          #a为元组,从列表b生成
c= list() #c为列表，空列表
c=[] #也是一个空列表
d=[1] #注意这里和元组不同，此处得到的是列表d，而不是整数1；背后原因，是元组用小括号与运算符括号在元组只1个元素时候的“表义”冲突了；所以只有一个元素的元组应该用逗号 a=（1,）

e = [1,[1,8.0,'abc'],8.0,'abc']
e[1][2]     #'abc',第1个1，代表是e的第2项列表项；第2个2代表第2个列表项的第3个元素
```

列表-常用操作

序列的基本操作，列表都支持

索引操作

a = list([1,2,3])

#a[0]的值为1，a[2]的值为3。索引值可为

负数，例如a[-1]为3，a[-3]为1。

a[1]= ‘abc’

切片操作：**[start: stop: step]**

a = list((1,2,3,4,5))

a[0:4:2]=[1, 3]

a[-3:1]=[]

a[-3:3]=[3]

a[-1:-6:-1]=[5,4,3,2,1]

连接操作

+运算符：

[1,2,3]+[4,5]为[1,2,3,4,5]

重复操作

3*[1,2]和[1,2]*3一样，结果为[1,2,1,2,1,2]

求列表长度

len([])为0; len([1,2,3])为3

集合操作

a = [1,2,3,2,3,1]

b = [4,5]

4 in b　　#True; 4 not in a,True;

a.count(2) #2，计算2在a中出现的次数；

a.index(3,2,4) #2,查找3在a的设定索引区间第1次出现位置

列表-常用操作

比较操作

运算符：**<、>、<=、>=、==和!=**

从第一个元素顺序开始比较，如果相等，则继续，返回第一个不相等元素比较的结果。如果所有元素比较均相等，则长的列表大，一样长则两列表相等。

[1,2]<[1,3]的结果为True，结果为布尔类型

```
s1 = ["abc", 3, 4]
s2 = ["abcc",3, 2]
s3 = ["abc", 3, 2]
print(s1 == s2, s1 != s2, s1 > s2, s1 < s2)
print(s1 == s3, s1 != s3, s1 > s3, s1 < s3)
print(s2 == s3, s2 != s3, s2 > s3, s2 < s3)
```

布尔操作

all(list)：元素全‘真’为True，有‘假’为False。

注：序列对象为空的时候，函数返回值为 True

any(list)：元素全‘假’为False，有‘真’为True " 。

注：序列对象为空的时候，函数返回值为False

```
s1 = [1, 0]
s2 = [1, '']
s3 = [1,'abc']
s4 = [0,'',False]
s5 = []
s6 = ()
```

空序列处理分析 （以all为例）

```
for element in iterable:
    if not element:
        return False
return True
```

all(s1) #all(s1)为False，因为有0。all(s2)为False，因为有空串。all(s3)为True。all(s5)为True，all(s6)为True。

any(s1) #any(s1)为True，因为有1。any(s2)为True，因为有1。any(s4)为False,因为每一个都为“假”。any(s5)为False，any(s6)为False。

列表-常用操作

排序操作

两个内置排序函数

sorted(iterable, key = None, reversed = False)

sorted([1,3,2,4,5]) #返回一个**新的列表**[1,2,3,4,5]

listobj.sort(key = None, reverse = False)

lst=[1,3,2,4,5]

lst.sort() #修改**原列表对象**的内容

都可以按函数key的结果排序，并选择正序和反序。

key参数：可以指定一个函数，用该函数的返回值来作为排序依据。也可以实用匿名函数lambda来实现基于表达式简单排序。

应用key参数

```
def key1( x ):
    return( x * x - 4 * x + 4 )
s = [1, 2, 5, 4, 3]
print(sorted(s, key = key1, reverse = True)) #降序，等同于sorted(s, key=lambda x:x*x-4*x=4, reverse = True)
```

#如上，key参数代表特定的排序规则，也能指定特定的排序对象，例如在类似列表元素是组合类型的场景中，可以使用key参数对组合类型元素中的某一项排序

```
a =  [('b',3), ('a',2), ('d',4), ('c',1)]
#构建一个由多个元组构成的列表
sorted(a,key=lambda x:x[1])
```

#[('c',1),('a',2),('b',3),('d',4)]，即按照每个元组的下标为1的元素进行排序

列表-常用操作

拆封操作

列表的各个数据可一次性赋予多个变量：

a=[1,2,3]

(a1,a2,a3)=a 或者*a1,a2,a3=a*

结果a1=1，a2=2，a3=3

占位符：*(_,b,_)=a*,则b=2

删除和修改操作

a=[1,2,3,4,5]

del a[3] #a=[1,2,3,5]

a[3]=8 #a=[1,2,3,8]

max、min、sum内置函数

取序列中的最大值、最小值和求和

a= [0, 1, 2, 3, 4]

max(sample_list1)　　#4，列表中元素的最大值

min(sample_list1)　　#0，列表中元素的最小值

sum(a)　#10, 列表中元素的和，前提是元素可以相加

sum(iter[, start]) iter为可迭代对象，start指定相加的参数，默认是0

sum也可以用于列表的展开，相当于子列表相加

a=[[1, 2], [3, 4]]

sum(a, []) #[1, 2, 3, 4]

注意，此处 sum的第二个参数不能省略，因为默认值为 0，0和列表不能相加会报错

列表-常用操作

方法	说明
s.append(x)	将x添加到列表末尾，列表长度加1
s.clear()	删除整个列表，s为空列表，等于del s[:]
s.copy()	复制出一个新列表，返回给调用者，复制过程为浅拷贝 。
s.extend(t)	把可迭代对象t附加到s的尾部
s.insert(i,x)	在下标为i的位置插入对象x，i以后的元素后移
s.pop([i])	如果不指定i，则返回列表最末尾的对象并从列表中删除该对象。如果指定i则返回列表中第i个位置的对象并将其从列表中删除。列表为空或i超过正常范围时该方法出错
s.remove(x)	从列表中找到第一次出现的x并删除，若无x出错
s.reverse()	将数据项反序构成新列表
s.sort()	列表排序

```
a=[1,2,3]
a.append(4) #在最后面附加上去元素4
a.clear()  #清空整个列表，得到空列表
a=[1,2,3]
b=a.copy() #a与b相等;下节介绍浅拷贝原理
a=[1,2,3]
b=['x','y']
a.extend(b)   #a变为[1, 2, 3, 'x', 'y']
b.insert(1,'z')  #b变为['x','z','y']
b.pop()   #返回'y'，同时b变为['x','z']
b.pop(0) #返回'x'，同时b变为['z']
a.remove(2)  #a变为[1, 3, 'x', 'y']
a.remove('x') #a变为[1, 3,'y']
a.reverse()   #a变为['y',3,1]
```

列表解析式

定义：可迭代对象可用列表解析快速遍历

语法：**[expr for v^1 in S^1 ..... for v^n in S^n]**

实例：

```
>>> s1=[1, 2, 3, 4, 5]
>>> s2=[1, 2, 3]
>>> print([i*j for i in s1 for j in s2])
[1, 2, 3, 2, 4, 6, 3, 6, 9, 4, 8, 12, 5, 10, 15]
```

列表解析式

提供了一种“优雅、便捷”的生成列表的方法。

生成0~100所有偶数组成的列表

```
#常规方法
a=[]
for i in range(101):
    if i%2==0:
        a.append(i)
#用列表解析，只需要1行代码
a=[x for x in range(101) if x%2==0]
#for可以包含简单条件判断
```

获取文本中所有单词的第1个字符

```
#常规方法
text="My house is full of flowers"
f_ch=[]
for word in text.split():
    f_ch.append(word[0])
#用列表解析，只需要1行代码
f_ch=[word[0] for word in text.split()]
```

对应位相乘

```
a=[2,3,4,5]
b=[3,4,5,6]
c=[a[i]*b[i] for i in range(4)]
#[6, 12, 20, 30]
```

笛卡尔乘积

```
a=[2,3,4,5]
b=[3,4,5,6]
c=[i*j for i in a for j in b]
#[6, 8, 10, 12, 9, 12, 15, 18, 12, 16, 20, 24, 15, 20, 25, 30]
```

列表解析式

带else的解析

```
a=['1','2','i','8']
b=[int(i) if i.isdigit() else 0 for i in a]

#[1,2,0,8]
#是数字则转为数字，否则转为0
#带else的表达式按语法在for子句前
```

去掉列表两层嵌套，生成1个无嵌套的列表

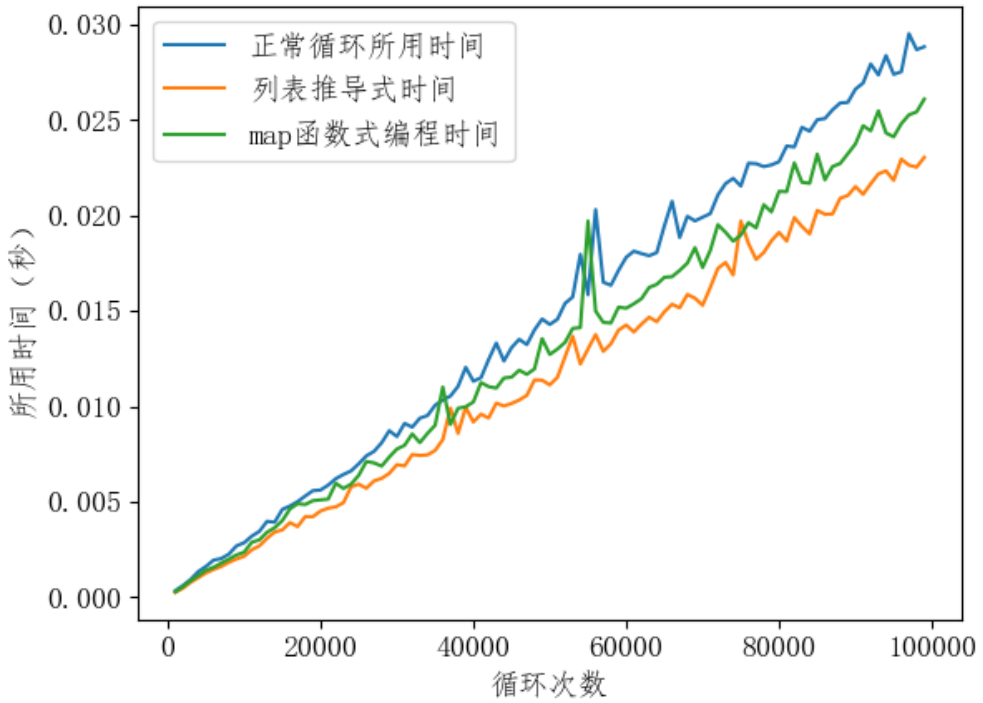
```
a=[[1,2],[3,4,5],[8]]
b=[x  for i in a for x in i]

#[1,2,3,4,5,8]
#i是列表a的子列表，x是列表i中的元素
```

列表解析，底层由C语言进行实现，比Python构造列表的循环方法性能略好。

列表解析式的性能

除了简洁的写法上的优点，列表解析方式在适合的场景下性能通常略优于一般的循环方式。




```
# 比较列表解析式 map函数 循环的性能
import timeit, math
import matplotlib.pyplot as plt
from pylab import mpl

mpl.rcParams['font.sans-serif'] = ['FangSong']  # 设置matplotlib可以显示汉语
mpl.rcParams['axes.unicode_minus'] = False
mpl.rcParams['font.size'] = 13

y1, y2, y3 = [], [], []
x = [i for i in range(1000, 100000, 1000)]
for num in range(1000, 100000, 1000):
    original = [i for i in range(num)]

    t1 = timeit.default_timer()
    res1 = []
    for eve in original:
        res1.append(1 / ((math.exp(-eve)) + 1))
    t2 = timeit.default_timer()

    y1.append(t2-t1)

    t3 = timeit.default_timer()
    # print("\n", "正常循环所用时间:{}".format(t2-t1), "\n", "----------------------------------------------------------------------")
    res2 = [1 / ((math.exp(-eve)) + 1) for eve in original]
    t4 = timeit.default_timer()

    y2.append(t4-t3)

    t5 = timeit.default_timer()
    # print("\n", "列表推导式时间:{}".format(t3-t2), "\n", "----------------------------------------------------------------------")
    res3 = list(map(lambda  x: 1/((math.exp(-x))+1), original))
    t6 = timeit.default_timer()

    y3.append(t6-t5)
    # print("\n", "map函数式编程时间:{}".format(t4-t3), "\n", "----------------------------------------------------------------------")

print(y1)
print(y2)
print(y3)

plt.figure(1)
plt.plot(x, y1, label="正常循环所用时间")
plt.plot(x, y2, label="列表推导式时间")
plt.plot(x, y3, label="map函数式编程时间")
plt.xlabel("循环次数")
plt.ylabel("所用时间（秒）")
plt.legend()
plt.show()
```

序列的拷贝

直接赋值： 其实就是对象的引用（别名）

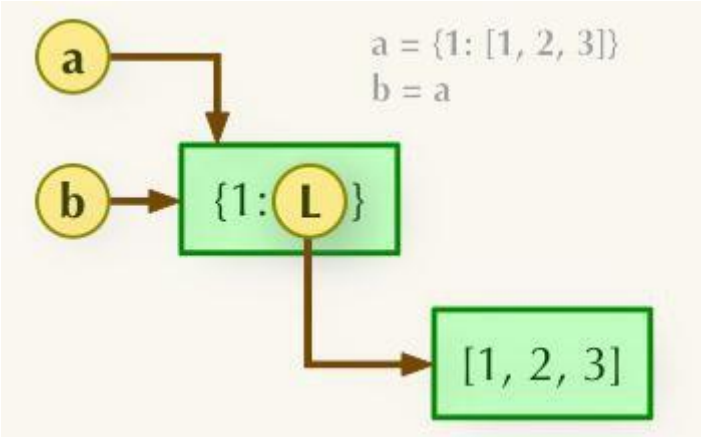
浅拷贝(copy)： 拷贝父对象， 不会拷贝对象的内部的子对象

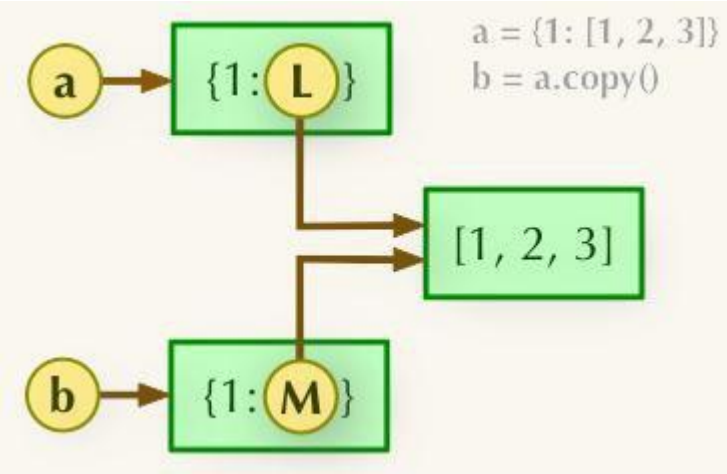
深拷贝(deepcopy)： copy 模块的 deepcopy 方法， 完全拷贝了父对象及其子对象

序列的拷贝

b = a: 赋值引用， a 和 b 都指向同一个对象。

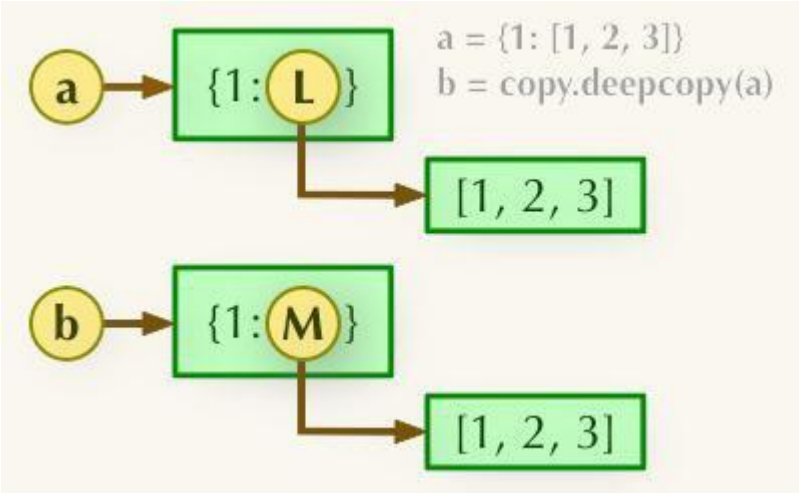
b = a.copy(): 浅拷贝, a 和 b 是独立的对象， 但他们的子对象还是指向统一对象（是引用）





序列的拷贝

b = copy.deepcopy(a): 深度拷贝, a 和 b 完全拷贝了父对象及其子对象， 两者是完全独立的



序列的拷贝

```
import copy
a = [1 , 2, 3, 4, ['a', 'b']] #原始对象
b = a #赋值， 传对象的引用
c = copy.copy(a) #对象拷贝， 浅拷贝
d = copy.deepcopy(a) #对象拷贝， 深拷贝
a.append(5) #修改对象a
a[4].append('c') #修改对象a中的['a', 'b']数组对象
print( 'a = ', a )
print( 'b = ', b )
print( 'c = ', c )
print( 'd = ', d )
```

示例及执行结果

```
a =  [1, 2, 3, 4, ['a', 'b', 'c'], 5]
b =  [1, 2, 3, 4, ['a', 'b', 'c'], 5]
c =  [1, 2, 3, 4, ['a', 'b', 'c']]
d =  [1, 2, 3, 4, ['a', 'b']]
```

集合 – 基本定义与创建

- ✓ 集合。是一个无序的不重复元素序列。
- ✓ **集合的各数据项必须为可哈希对象**（即，不可变数据类型，如数字类型、字符串、元组等），且不可重复。
- ✓ 分为可变集合和不可变集合两种。

  可以用{ }或set()函数、frozenset()创建集合。

  注意：创建一个空集合必须用**set()而不是{}**，后者用来创建一个空字典。
- ✓ 通过list(集合)、tuple(集合)，可以将集合转化为列表和元组。

```
a={1,2,3} #创建可变集合a

b=set([3,2,1])#创建可变集合b

c=frozenset([2,3,1])#创建不可变集合c

type(a);type(b);type(c)#<class 'set'>、<class 'set'>、 <class 'frozenset'>

a==b==c #True，说明集合无序
```

集合的基本运算

- ✓ S1|S2，求并集。返回集合S1∪S2。
- ✓ S1&S2，求交集。返回集合S1∩S2。
- ✓ S1-S2，求差集。返回存在于S1中，但不存在于S2中所有元素构成的集合。
- ✓ S1^S2，求补集。返回集合S1▵S2，就是(S1-S2)∪(S2- S1)。

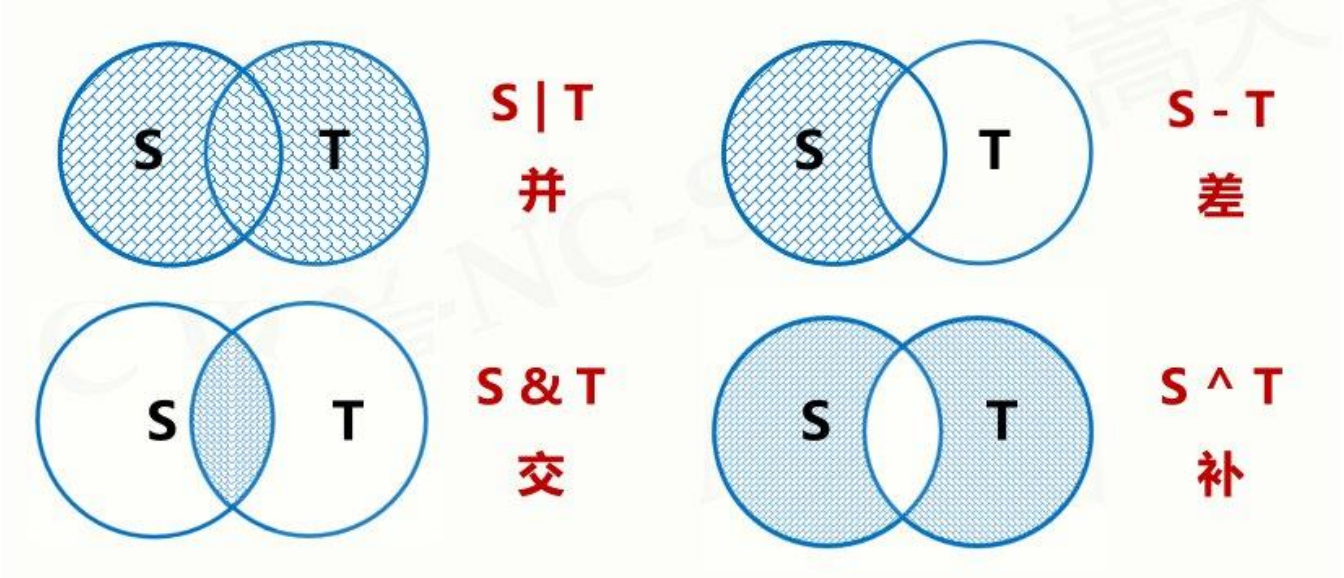
S1={1,2,3}，S2={2,4,5}

并集：***S1\|S2={1,2,3,4,5}***

交集：***S1&S2={2}***

差集：***S1-S2={1,3}***

补集：***S1^S2={1,3,4,5}***



集合的比较

s1==s2	若s1和s2完全相等，元素完全相同，返回真，否则假。
s1!=s2	若s1和s2元素不完全相同，返回真，否则假。
s1>s2	若s1为s2的纯超集，则返回真，否则假。
s1<s2	若s1为s2的纯子集，则返回真，否则假。
s1>=s2	若s1为s2的超集，则返回真，否则假。
s1<=s2	若s1为s2的子集，则返回真，否则假。

集合的其他运算

- ✓ ***s.copy()*** 返回集合s的浅拷贝。
- ✓ ***s1.isdisjoint(s2)*** s1和s2交集是否为空。
- ✓ ***s1.issubset(s2)*** s1是否是s2的子集。
- ✓ ***s1.issuperset(s2)*** s1是否是s2的超集。
- ✓ 并、交、差、补，等价运算方法

  s1.union(s2,...) ***s1.intersection(s2,...)***

  s1.difference(s2,...)，即s1-s2

  s1.symmetric_difference(s2)，即(s1-s2)∪(s2-s1)

```
a={1,2,3}          #创建可变集合a
b=a.copy()         #浅拷贝得到集合b
a==b               #True
c={1,2,3,4}        #创建可变集合c
a.issubset(a);a.issubset(c)  #True,a是a本身的子集，a也是c的子集
c.issuperset(c);c.issuperset(a)  #True,c是c本身的超集，c也是a的超集
```

集合的其他运算

- ✓ ***s.add(x)*** 将数据项x添加到s中，如果s中尚未有x。*注：x可以是不可变集合，但不能是可变集合。*
- ✓ ***s.clear()*** 清空集合。
- ✓ ***s.remove(x)*** 从s中删除x，若x不存在，导致错误。
- ✓ ***s.discard(x)*** 若s中存在x，从s中删除x。
- ✓ ***s.pop()*** 随机返回s中的一个元素，并且该元素从s中删除，若s为空，发生错误。

```
a.add(4)       #a变为{1,2,3,4}
a.remove(4)    #a变为{1,2,3}
a.remove(5);a.discard(5)   #集合里面没有5，remove操作出错，但是discard操作不出错
a.pop()    #由于集合无序，所以只能随机返回和删除，不像列表返回和删除的是最后一个元素
a.clear()    #a变为空集合——set()

s = {  'Python' ,  'C' ,  'C++'  }
fs = frozenset(['Java', 'Shell'])
s_sub = {'PHP', 'C#'}
#向set集合中添加frozenset
s.add(fs)
print('s =', s)
#向set集合添加子set集合
s.add(s_sub)   #报错
```

字典 基本概念和特性

- ✓ 映射：通过“名称”而非序号，来访 问“值”的数据结构，称为映射 （mapping）。
- ✓ 字典，是Python内置的映射类型。键应是可哈希的对象（不可变类型）。一 个key:value的键值对，称为一个item。

```
{'name': '张三', 'gender': '男', 'phone': '62282222', 'province': '北京'}
{'张三': '02', '李四': '03', '钱一': '01', '王五': '08'}
{1: '北京', 2: '上海', 3: '广州', 4: '深圳'}
{(0, 0): '北京', (0, 1): '上海', (0, 2): '广州', (0, 3): '深圳', (1, 0): '天津'}
# 分别用英文/中文字符串、数字、元组做“键”
```

- ✓ 保存的数据不是按照添加进去的顺序保存的，是 按照 hash 表的结构保存的，所以没有索引。只能通过 key 来获取字典中的数据。
- ✓ 通过key直接查找元素，不需要遍历，效率高。
- ✓ “键”必须是唯一的，重复赋值后者会覆盖前者。
- ✓ 典型场景——“通过键找值” ，如电话本、班级 成员信息、个人简历、文件属性等。
- ✓ 字典支持复杂的多层结构。

```
people = {'Aron': {'phone': '2765', 'addr': 'Queen Ave. 25'}, 'Bryan': {'phone': '7233', 'addr': 'King Ave. 19’}}
```

#第一层的“键”是名字，“值”也是字典 第二层的“键”是电话和地址

字典与列表的对比

字典 vs 列表

	列表	字典
存储顺序	任意对象的有序集合	任意对象的无序集合
读取方式	通过索引号读取	通过键读取
长度与成员	可变长度，异构及任意嵌套	可变长度，异构及任意嵌套
查找效率	查找和插入时间随着元素的增加而增加（注意不是append）	查找和插入速度快，不随键值对增加而显著增加
内存占用	占用空间小，浪费少	占用空间较大（hash表特性）

字典的创建

- ✓ 可以通过“键值对”创建字典。“键值对”为空时候，创建空字典；也可以通过***dict()***函数创建字典。
- ✓ 可以通过***fromkeys***方法创建只有“键值”，而“值”为默认值的字典，该参数空缺时所有字典项的值为*None*。
- ✓ 可以直接通过“键值”，更新或增加“键值对”。
- ✓ 可以通过***update()***方法，更新或增加“键值对”。

```
d1 = {} # 创建一个空字典
d2 = {'name': '张三', 'gender': '男', 'phone': '62282222'} # 创建三个键值对的字典
a = [(1, '北京'), (2, '上海'), (3, '广州'), (4, '深圳')]
d3 = dict(a) # 使用dict函数创建字典
d4 = dict.fromkeys(['name', 'age', 'addr'], '(默认值)')
d2['name'] = '李四' # 将把name的值改为李四
d2['year'] = '2000' # 由于d2中没有year，将新增一项
d2.update({'year': '1999'}) # 更新已有的值；如果没有则新增创建一项
```

字典的拷贝

- ***copy***。返回一个新字典，其包含的键值对与原来的字典相同（该方法执行的是“浅拷贝”）。
- ***deepcopy***。深拷贝，即同时复制值及其包含的所有值。

```
import copy

d = {}
d['names'] = ['张三', '李四']  # {'names':['张三', '李四']}
c = d.copy()  # 字典c用cpoy方法复制d；即，值是“引用”
dc = copy.deepcopy(d)  # 用deepcopy方法复制；值也是新的
d['names'].append('王五')  # 给d追加一个元素
print(c)  # {'names': ['张三', '李四', '王五']}，c和d一样——“浅拷贝”的效果
print(dc)  # {'names': ['张三', '李四']}，dc中并没有d中追加的新元素——“深拷贝”的效果
```

字典的读取

- ✓ 按“键”读取。***d[k]***
- ✓ get方法读取。***d.get(k)***
  - 没有该键时，返回None。
  - 可以指定一个自己想要的返回值 ***d.get(k,v)***。
- ✓ ***setdefault*** 有点像get，它也获取与指定键相关联的值，但除此之外，setdefault还在字典不包含指定的键时，在字典中添加指定的键值对。

```
d = {}
d = {'name': '张三', 'gender': '男', 'phone': '62282222'}
d.get('name')  # '张三'
d.get('addr')  # None
d.get('addr', '没有该元素')  # 返回设定的值——'没有该元素'

d.setdefault('wish', 'N/A')  # 无该元素，则在d中增加该元素，且同时返回该方法设定的默认值'N/A'

d['wish'] = '科学家'  # 更新该键下面的值为“科学家”

d.setdefault('wish', 'N/A')  # 再度执行本操作，则返回值为“科学家”

# setdefault的另一个例子
girls = ['alice', 'bernice', 'clarice']
boys = ['chris', 'arnold', 'bob']
letterGirls = {}
for girl in girls:
    letterGirls.setdefault(girl[0], []).append(girl)
print([b + '+' + g for b in boys for g in letterGirls[b[0]]])
```

字典的解析

和列表解析的方法类似，也可以通过对可迭代对象的遍历产生字典。

语法： ***{keyexpression:valueexpression for key,value in iterable if condition}***

示例：

```
filesize = {filename: os.path.getsize(filename) for filename in os.listdir('.') if
os.path.getsize(filename) > 1024}
```

{'sample1.html': 3632, 'scratch.py': 12402, 'scratch_1.py': 2145, 'scratch_10.py': 1425, 'scratch_4.py': 7780, 'scratch_6.py': 1933}

颠倒字典的键和值：

当字典的值也各不相同时，我们可以写：　　***{k:v for v,k in dict.items()}***

字典视图

d.keys()	返回字典所有key构成的列表
d.values()	返回字典所有value构成的列表
d.items()	返回字典所有(key,value)对构成的列表

- ✓ **字典视图**，是一组特殊类型*(dict_xxx)*的对象。字典视图的优点是不复制，它们始终是底层字典的反映（类似数据库中的视图和表的关系）。修改了字典的值，视图中的值同样随着改变。

```
d = {'d1': 1, 'd2': 2, 'd3': 3, 'd4': 2}
a = d.keys()  # dict_keys(['d1','d2','d3','d4']),返回所有keys
b = d.values()  # dict_values([1, 2, 3, 2]),返回所有values；注意会有重复，因为值本身就有重复
c = d.items()  # dict_items([('d1', 1), ('d2', 2), ('d3', 3), ('d4', 2)]) ,返回所有键值对
d['d5'] = 5
print(a)
print(b)
print(c)  # a，b，c的值也自动变化
# dict_keys(['d1', 'd2', 'd3', 'd4', 'd5'])
# dict_values([1, 2, 3, 2, 5])
# dict_items([('d1',1),('d2',2),('d3',3),('d4',2),('d5',5)])
```

字典的删除

- ✓ ***del()***。使用del函数删除指定元素。
- ✓ ***d.pop(k,v)***。使用pop方法， 删除指定键值的元素。如k不存在，返回v。
- ✓ ***d.popitem()***。使用popitem方法，随机删除一个元素。
- ✓ ***d.clear()***。使用clear方法，清空字典。

```
d = {'d1': 1, 'd2': 2, 'd3': 3, 'd4': 2}

del d['d1']  # {'d2':2,'d3':3,'d4':2}

d.pop('d2')  # {'d3':3,'d4':2}

d.popitem()  # {'d4':2}

d.clear()  # {}
```

字典的其他函数

- ✓ ***len()***。返回字典元素的个数。
- ✓ 比较运算。 ***== ， !=***。
- ✓ 集合运算。
  - *k in d*，检查是否字典的键中有k。
  - *v in d.values()*，检查是否字典的值 中有v。
  - *(k,v) in d.items()*，检查键值对是否在字典中

```
d1 = {'d1': 1, 'd2': 2, 'd3': 3, 'd4': 2}
d2 = {'d1': 1, 'd2': 2, 'd3': 3}
len(d1)  # 4
d1 == d2  # False
d1 != d2  # True
'd4' in d1  # True，检查键是否存在；等同in d1.keys()
2 in d1.values()  # True，检查值是否存在
('d4', 2) in d1.items()  # True，检查键值对是否存在
```

字典：利用字典产生多分支

Python不存在switch语句，可以利用字典模拟。

```
print('请输入1-4之间的数字')
x = int(input())
if x == 1:
    print('1的英文是：one')
elif x == 2:
    print('2的英文是：two')
elif x == 3:
    print('3的英文是：three')
elif x == 4:
    print('4的英文是：four')
else:
    print('输入的数字，我不会翻译。')
```

常规方法

```
d = {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
print('请输入1-4之间的数字')
x = int(input())
t = d.get(x, 'N/A')  # 直接一下子取出来对应的英文
if t == 'N/A':  # 字典中没有的元素，get返回特定字符
        print('输入的数字，我不会翻译。')
else:
    print(str(x) + '的英文是：' + t)


# 设想一下，如果分支数量很多。。。
```

字典方法

字典示例　模拟抽奖

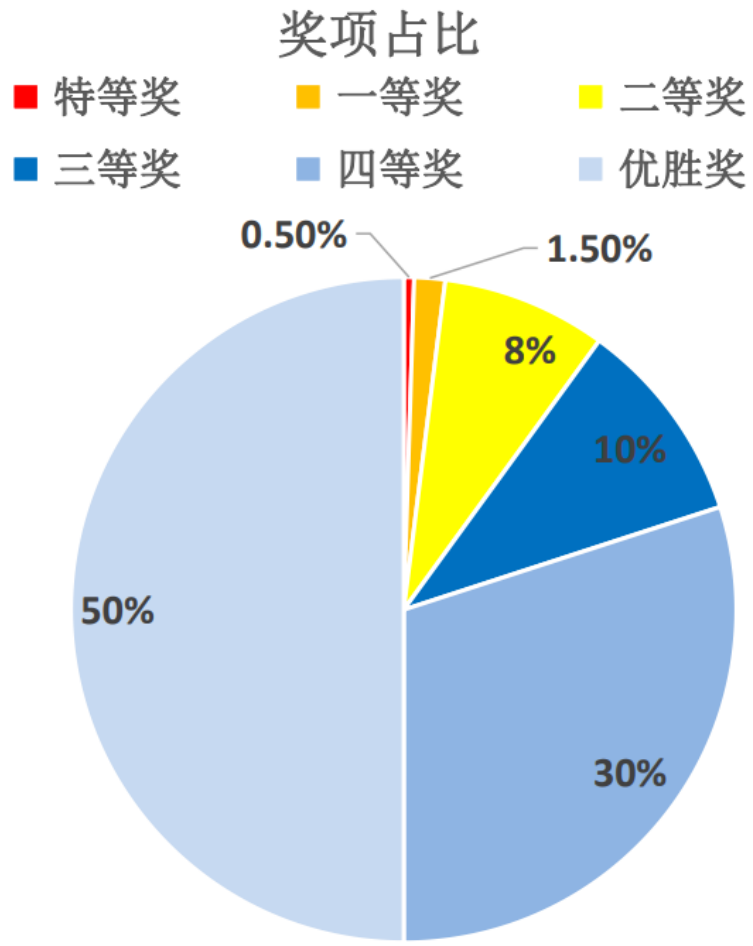
模拟轮盘抽奖游戏

轮盘分为：特等奖、一等奖、二等奖、三等 奖、四等奖、五等奖

轮盘转的时候是随机的：

- ✓ 如果范围在[0,0.005]之间，代表特等奖。
- ✓ 如果范围在[0.005,0.02]之间，代表一等奖。
- ✓ 如果范围在[0.02,0.1]之间，代表二等奖。
- ✓ 如果范围在[0.1,0.2]之间，代表三等奖。
- ✓ 如果范围在[0.2,0.5]之间，代表四等奖。
- ✓ 如果范围在[0.5,1.0]之间，代表优胜奖（五等奖）。

计算1000次抽奖的话，各级别奖项的人数。



字典示例　模拟抽奖

```
import random

# 定义一个字典，模拟转盘奖项类别和对应概率
reward_d = {'特等奖': (0, 0.005), '一等奖': (0.005, 0.02), '二等奖': (0.02, 0.1), '三等奖': (0.1, 0.2), '四等奖’:
          (0.2, 0.5), ‘五等奖’: (0.5, 1.0)}
result_d = dict.fromkeys([‘特等奖’, ‘一等奖’, ‘二等奖’, ‘三等奖’, ‘四等奖’, ‘五等奖’], 0)
# 利用fromkeys创建一个存储计数结果的字典
for i in range(1000):
    num = random.random()
    for k, v in reward_d.items():  # 利用items()方法，获取奖项和概率元素，进行比对
            if v[0] < num <= v[1]:  # 下开上闭，随机数字落在哪个里面，就认为中了几等奖
                reward_level = k  # 取出该概率对应的奖励等级名字
                result_d[reward_level] += 1  # 把结果字典中，该奖励等级的计数加1

print(result_d)
```

某次执行结果：{'特等奖': 8, '一等奖': 15, '二等奖': 81, '三等奖': 95, '四等奖': 293, '五等奖': 508}

综合实例　问题描述

- ✓ 有一份文档包含10万条记录，每条记录有姓名、性别、出生年份共3个字段。
- ✓ 给出一个1万人的姓名名单，要求统计这1万人中，2000年出生的女性有多少人。

name	gender	year
name156	f	2001
name241	m	2003
name037	m	1997
name891	m	1995
……		

name
name011
name891
name451
name037
…..

综合实例-方法一 使用列表

```
# list_demo
import random
import time
#step1.产生10万个随机数数据，加入列表random10
random20=list() #初始化一个空的列表
for x in range(0,200000): #产生20万个数
    i = random.randint(1,10000000) #数的范围，1-1千万
    random20.append(i) #把产生的数添加到random20中
print(len(random20))
myset=set(random20)
print(len(myset))
random10=random.sample(random20,100000) #从random20中选取10万个数
print(len(random10))
myset=set(random10)
print(len(myset))
```

```
#step2.产生10万个姓名、性别和出生年份，分别加入列表namelist、genderlist、yearlist
namelist=[] #初始化空的列表
genderlist=[]
yearlist=[]
start=time.time()
print("step2 go...")
for x in range(0,100000): #10万次循环
    name= “name” +str(random10[x]) #name加上某个数字编号，保证唯一吗?
    year=random.randint(1990,2010) #年份范围
    gender= "f" if x%3==1 else "m"
    namelist.append(name)
    yearlist.append(year)
    genderlist.append(gender)
end=time.time()
print("step2 done:",end-start)
print(len(namelist))
```

综合实例-方法一 使用列表

```
#step3.产生1万个名字数据，加入namelist10k
num10k=random.sample(random10,10000)
#从random10这个列表中取出1万个数
num10k=random.sample(random20,10000)
#从random20这个列表中取出1万个数
namelist10k=[]
for i in num10k:
    namelist10k.append("name"+str(i))
```

```
#step4.取出1万个名字，依次与10万个姓名一一对照，求得位置，然后到另外两个数组中找到对应的值进行判断
start=time.time()
print("step4 go-------------")
count=0
for name in namelist10k: #取一万次
    if name in namelist: #在10万条记录中搜索，首先需要判断name是否在namelist中，如果不在，下一步使用index会报错。
        ind=namelist.index(name) #取出name在列表中的序号，在10万条记录中定位
        if yearlist[ind]==2000 and
        genderlist[ind]=="f":
            count+=1
end=time.time()
print("step4 done. list time:",end-start)
print("count is",count)
```

综合实例-方法二 使用字典

```
#step1和step3与前面相同，只需要修改step2和step4
#step2.产生10万个姓名、性别和出生年份，加入字典mydict
mydict = dict() #初始化一个空的字典
start = time.time()
print("step2 go...")
for x in range(0,100000): #10万次循环
    name = "name" + str(random10[x]) #name加上某个数字编号
    year = random.randint(1990,2010) #年份范围
    gender = "f" if x % 3 == 1 else "m"

    v = [year, gender] #将year和gender放在v列表中
    mydict.setdefault(name, v) #添加到字典中，并以name作为key

end=time.time()
print("step2 done:",end-start)
print(len(mydict))
```

```
#step4.使用字典查找
count = 0 #初始化计数器
print("step4 go-----------------")
start=time.time()
for x in namelist10k: #取一万次
    if mydict.get(x): #hash查找,O(1)的时间复杂度
        if mydict[x][0]==2000 and
        mydict[x][1]=="f": #出生年份是2000年的女生
            count += 1

end=time.time()
print("step4 done. dict time:",end-start)
print("count is",count)
```

综合实例

方法一： 列表

时间消耗的差距有4千倍

用1万条记录去10万记录中查找：

step2 go...
step2 done: 0.416759729385376
100000
step4 go--------------
step4 done. list time: *32.43141961097717*
count is 135

用1万条记录去100万记录中查找：

线性增长

step2 go...
step2 done: 4.04068398475647
1000000
step4 go--------------
step4 done. list time: *313.44585824012756*
count is 163

方法二： 字典

用1万条记录去10万记录中查找：

step2 go...
step2 done: 0.425747871398925
899484
step4 go----------------
step4 done. dict time: *0.008013963699340 82*
count is 168

用1万条记录去100万记录中查找：

几乎不变

step2 go...
step2 done: 5.160044193267822
951751
step4 go----------------
step4 done. dict time: *0.009993076324462 89*
count is 140

单选题 1分

设置

```
>>>numbers=[1,2,3,4,5,6,7,8,9,10]
>>>numbers[1:4]=[]
>>>numbers
```

A [1,6,7,8,9,10]

B [5,6,7,8,9,10]

C [4,5,6,7,8,9,10]

D 以上均不对

提交

多选题 1分

设置

下列说法哪些正确?

A sorted排序，不改变原来列表

B sort排序，也不改变原来列表

C 使用del时，可以以切片的方式一次删除多个元素

D pop方法，返回元素，同时从列表中删除该元素

提交

多选题 1分

设置

下列说法哪些正确?

A 列表解析生成列表，比Python循环方法快

B 不可变类型的赋值相当于“传值”，可变数据类型的赋值相当于“传址”

C 对于单层列表，浅拷贝和深拷贝效果一样

D 浅拷贝无论哪一层，遇到可变类型则创建新的内存空间

提交

多选题 1分

设置

下列说法哪些正确?

A 序列类型，是可变数据类型

B 元组，是可变数据类型

C 字符串，属于序列类型

D 序列的双向索引，最末尾元素小标是-1

提交

多选题 1分

设置

下列说法哪些正确?

A 元组中的元素，数据类型必须一致

B 创建元组时，t=(0,1,2)与t=0,1,2等价

C a[0:5:2]，相当于从0-4，隔1个取1个

D 两个列表可以进行比大小的操作

提交

多选题 1分

设置

下列说法错误的有?

A 集合是有序的不重复元素序列

B 集合的元素可以是可变数据类型

C 创建空集合的命令为set{}

D 空集合等于"{}"

提交

多选题 1分

设置

下列说法哪些正确?

A 一个集合也是自己的子集

B 一个集合也是自己的超集

C 集合的copy方法是“浅拷贝”

D 两个集合的元素不完全相同即是该两集合不相等

提交

多选题 1分

设置

下列说法哪些正确?

A 集合的pop方法，返回最后一个元素，且删除该元素

B 集合的remove和discard方法，两者效果完全相同

C set()函数创建集合时，支持自动去掉重复元素

D tuple()函数可以将集合转化为元组

提交

多选题 1分

设置

下列说法哪些正确?

A 字典的键应为可哈希对象

B 相同的键，重复赋值将只保留后一次的值

C 字典的值，可以是嵌套的复杂结构

D 字典的元素是有顺序的

提交

多选题 1分

设置

下列说法哪些正确?

A 列表的查找和插入时间，随着元素增加而增加

B 字典的查找和插入时间，不会随着key增加而增加

C 同样的内容，字典占用空间比列表多

D 通过键赋值时，如果没有该元素，则自动增加

提交

多选题 1分

设置

下列说法哪些正确？

A 字典的copy方法，是“浅拷贝”

B 可以用get方法，在没有某key时，返回自己想要的值

C setdefault方法，没有某key时，将自动添加新元素

D 修改字典的值，字典视图的值不会跟着改变

提交

多选题 1分

设置

下列说法哪些正确?

A pop方法和popitem方法的效果一样

B del函数与clear方法的效果一样

C 字典与字典之间可以比较是否相等

D 可以用字典模拟switch多分支语句

提交

四 组合数据类型

- 组合数据类型概述
- 序列
- 集合
- 字典
- 综合实例

谢谢