# 第3章 chapter 3

# 元组、字典和集合

本章主要介绍：元组、字典和集合三种数据类型，序列解包，日期格式和字符串格式相互转化，循环中的 break、continue、pass 和 else。

## 3.1 元　　组

元组与列表类似，也是由一系列按照特定顺序排列的元素组成的，只不过元组是用圆括号()把元素括起来的，且元组是不可变序列，不能增加、修改和删除元组的元素。正是由于这一特性，元组用来存储那些常用但不能更改的数据，以免数据被改而造成未知的错误。

元组比列表的访问速度快，如果只需要访问元素，而不需要修改的话，建议使用元组。此外，同列表相比，元组通常占用的内存更小，因为列表具有可变性，需要额外的内存开销。


创建元组

### 3.1.1 创建元组

#### 1. 通过赋值创建元组

只要为变量分配一个写在圆括号()之间、用逗号分隔开的元素序列即可创建一个元组。

```
>>> tup = ()                          #创建空元组
>>> tup = (1,)                        #创建只有一个元素的元组，在元素后面要加上逗号
>>> tup = (1,2,["a","b","c"],"a")     #创建含有多个元素的元组
```

#### 2. 通过元组构造函数 tuple()将列表、集合、字符串转换为元组

```
>>> tup1=tuple([1,2,3])               #将列表[1,2,3]转换为元组
>>> tup1
(1, 2, 3)
```

### 3. 以逗号隔开的一组无符号的对象默认为是一个元组

```
>>> A='a', 5.2e30, 8+6j, 'xyz'
>>> type(A)
<class 'tuple'>
>>> A
('a', 5.2e+30, (8+6j), 'xyz')
```

## 3.1.2 修改元组


修改元组

元组属于不可变序列，因此，元组没有提供 append()、extend()、insert()、remove()、pop()方法，也不支持对元组元素进行 del 操作，但能用 del 命令删除整个元组。

元组中的元素是不允许修改的，但可以对元组进行连接组合，得到一个新元组。

```
>>> tuple1 = ('hello', 18 , 2.23, 'world', 2+4j)
>>> tuple2 = ('best', 16)
>>>tuple3 = tuple1 + tuple2                    #连接元组
>>> print(tuple3)
('hello', 18, 2.23, 'world', (2+4j), 'best', 16)
```

元组中的元素是不允许修改的，指的是元组中元素是不可变对象时，该位置处的元素不可以修改；但元素是可变对象时，可改变该对象中的元素。

```
>>> tuple4 = ('a', 'b', ['A', 'B'])
>>> id(tuple4[2])
59252488
>>> tuple4[2][0] = 'X'
>>> tuple4[2][1] = 'Y'
>>> tuple4[2][2:]= 'Z'
>>> tuple4
('a', 'b', ['X', 'Y', 'Z'])
>>> id(tuple4[2])
59252488
```

表面上看，tuple4 的元素确实变了，但其实变的不是 tuple4 的元素，而是 tuple4 中的列表的元素，tuple4[2]仍旧指向原来的列表。元组所谓的“不变”是指：元组的每个元素，指向永远不变，即指向'a'，就不能改成指向'b'，指向一个列表，就不能改成指向其他列表，但指向的这个列表本身是可变的。

【例 3-1】 for 循环遍历二元组构成的列表。

```
test_tuple = [("a",1),("b",2),("c",3),("d",4)]
print("准备遍历的元组列表:", test_tuple)
print('遍历列表中的每一个元组')
for (i, j) in test_tuple:
  print(i, j)
```

运行上述代码构成的程序，运行结果如下。

```
准备遍历的元组列表:[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
遍历列表中的每一个元组
a 1
b 2
c 3
d 4
```

### 3.1.3 生成器推导式

生成器推导式的用法与列表推导式非常相似,但在形式上,生成器推导式使用圆括号作为定界符,而不是列表推导式所使用的方括号。与列表推导式最大的不同是,生成器推导式的结果是一个生成器对象,比列表推导式具有更高的效率,空间占用非常少,尤其适合大数据处理的场合。

使用生成器对象的元素时,可以根据需要将其转化为列表或元组,也可以使用生成器对象的__next__()方法或者内置函数 next()进行遍历,或者直接使用 for 循环来遍历其中的元素,但不支持使用下标访问其中的元素。不管用哪种方法访问生成器对象的元素,只能从前往后正向访问每个元素,不可再次访问已访问过的元素。当所有元素访问结束以后,如果需要重新访问其中的元素,必须重新创建该生成器对象,enumerate、filter、map、zip 等其他迭代器对象也具有同样的特点。

```
>>> g = ((i+2) * *2 for i in range(5))          #创建生成器对象
>>> type(g)
<class 'generator'>
>>> tuple(g)     #将生成器对象转换为元组
(4, 9, 16, 25, 36)
>>> list(g)      #上面 tuple(g)已访问完生成器对象的元素,再次访问发现没有元素了
[]
```

生成器是用来创建一个 Python 序列的一个对象。使用它可以迭代庞大序列,且不需要在内存创建和存储整个序列,这是因为它的工作方式是每次处理一个对象,而不是一口气处理和构造整个数据结构。在处理大量的数据时,最好考虑生成器推导式而不是列表推导式。

```
>>> c = (x for x in range(11) if x%2==1)
>>> c.__next__()  #使用生成器对象的__next__()方法获取元素
1
>>> 3 in c
True
>>> 5 in c
True
>>> 5 in c          #第二次执行 3 in c 后,c 中的所有元素都被访问了,c 中就没有元素了
False
>>> 7 in c
False
>>> 9 in c
False
```

# 3.2 字 典

字典是写在花括号{}之间、用逗号分隔开的“键:值”对集合,字典的元素之间是无序的,字典是可变对象。字典是一种映射类型,键值对“键:值”是一种二元关系,源于属性和值的映射关系,键表示一个属性,值表示属性的内容,键值对“键:值”整体而言表示一个属性和它对应的值。“键”必须使用不可变类型的对象,如整型、浮点型、复数型、布尔型、字符串、元组等,但不能使用诸如列表、字典、集合或其他可变类型作为字典的键。在同一个字典中,“键”必须是唯一的,但“值”是可以重复的。


创建字典

## 3.2.1 创建字典

(1) 使用赋值运算符将使用花括号{ }括起来的“键:值”对集合赋值给一个变量即可创建一个字典。

```
>>> dict1={'姓名':'芳芳','年龄':20,'身高':168,'体重':50}
```

(2) 使用 dict()函数将一个二元组序列转换为字典。

```
>>> items=[('one',1),('two',2),('three',3),('four',4)]
>>> dict2 = dict(items)
>>> print(dict2)
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

(3) 使用 dict()函数、zip()函数把两个序列创建为字典。

```
>>> city=['北京','上海','天津','重庆']
>>> university=['北京大学','上海大学','天津大学','重庆大学']
>>> dict3=dict(zip(city,university))
>>> dict3
{'北京': '北京大学', '上海': '上海大学', '天津': '天津大学', '重庆': '重庆大学'}
```

(4) 通过关键字创建字典。

```
>>> dict3 = dict(one=1,two=2,three=3)
>>> print(dict3)
{'one': 1, 'two': 2, 'three': 3}
```

(5) 用字典类型 dict 的 fromkeys(iterable[,value=None])方法创建一个字典,以可迭代对象字符串、列表、元组的元素,以及字典的元素的键作为字典中的键,以 value 为键对应的值,默认为 None。

```
>>> iterable2 = [1,2,3,4,5,6]                    #列表
>>> v2 = dict.fromkeys(iterable2,'列表')
>>> v2
{1: '列表', 2: '列表', 3: '列表', 4: '列表', 5: '列表', 6: '列表'}
>>> iterable3 = {1:'one', 2:'two', 3:'three'}    #字典
>>> v3 = dict.fromkeys(iterable3, '字典')
>>> v3
{1: '字典', 2: '字典', 3: '字典'}
```


访问字典

### 3.2.2 访问字典

```
>>> dict1 = {'Alice':18, 'Beth': 19, 'Cecil': 20}  #创建变量 dict1,引用字典对象
```

(1) 通过“dict1[key]”的方法返回字典 dict1 中键 key 对应的值 value。

```
>>> print (dict1['Beth'])                          #输出键为'Beth'的值
19
```

(2) 通过 dict1.get(key)返回字典 dict1 中键 key 的值。

get()方法的语法格式如下。

```
dict1.get(key, default=None)
```

功能：返回指定键的值,如果指定键的值不存在,则返回 default 指定的默认值。

参数说明如下。

key：要查找的键。

default：如果指定键的值不存在,返回该默认值。

```
>>> dict1.get('Alice')   #通过字典对象的 get()方法获取'Alice'对应的值
18
>>> dict1.get("a", 9)    #返回不存在的键"a"对应的值
9
```

(3) dict1.setdefault(key)返回字典 dict1 中键 key 的值,与 get()方法类似。

setdefault()方法的语法格式如下。

```
dict.setdefault(key, default=None)
```

功能：如果键不存在于字典中,将会添加该键并将 default 的值设为该键的默认值,如果键存在于字典中,将读出该键原来对应的值,default 的值不会覆盖原来已经存在的键的值。

key：要查找的键。

default：键不存在时,设置的默认键值。

```
>>> dict2 = {'Name': 'XiaoMing', 'Age': 17}
>>> print ("Age 键的值为 : %s" %  dict2.setdefault('Age', None))
Age 键的值为 : 17
>>> print ("Sex 键的值为 : %s" %  dict2.setdefault('Sex', None))
Sex 键的值为 : None
>>> print ("新字典为: ", dict2)
新字典为:  {'Name': 'XiaoMing', 'Age': 17, 'Sex': None}
```

(4) dict1.keys()返回字典 dict1 中所有的键组成的列表。

```
>>> dict1.keys()
dict_keys(['Alice', 'Beth', 'Cecil'])
```

(5) dict1.values()返回字典 dict1 中所有的值组成的列表。

```
>>> dict1.values()
dict_values([18, 19, 20])
```

(6) dict1.items()返回字典 dict1 的(键,值)二元组组成的列表。

```
>>> dict1.items()
dict_items([('Alice', 18), ('Beth', 19), ('Cecil', 20)])
```

【例 3-2】 遍历输出字典元素。

```
person={'姓名':'李明', '年龄':'26', '籍贯':'北京'}
#items()方法把字典中每对 key 和 value 组成一个元组,并把这些元组放在列表中返回。
for key,value in person.items():
    print('key=',key,',value=',value)
for x in person.items(): #只有一个控制变量时,返回每一对 key 和 value 对应的元组
    print(x)
for x in person:          #不使用 items(),只能取得每一对元素的 key 值
    print(x)
```

运行上述程序代码,所得的输出结果如下。

```
key = 姓名 ,value = 李明
key = 年龄 ,value = 26
key = 籍贯 ,value = 北京
('姓名', '李明')
('年龄', '26')
('籍贯', '北京')
姓名
年龄
籍贯
```

### 3.2.3 添加与修改字典元素


添加与修改字典元素

(1) 使用[]运算符添加字典元素。

```
>>> school={'class1': 60, 'class2': 56, 'class3': 68, 'class4': 48}
>>> school['class5']=70 #不存在键'class5',为字典 school 添加元素'class5': 70
>>> school
{'class1': 60, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 70}
>>> school['class1']=62 #存在键'class1',修改键 class1 所对应的值
>>> school
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 70}
```

由上可知,当以指定“键”为索引为字典元素赋值时,有两种含义:①若该“键”不存在,则表示为字典添加一个新元素,即一个“键:值”对;②若该“键”存在,则表示修改该“键”所对应的“值”。

(2) 使用字典对象 school1 的 update(school2) 方法可以将字典对象 school2 的元素一次性全部添加到 school1 字典对象中,如果两个字典中存在相同的“键”,则只保留字典对象 school2 中的“键:值”对,此时相当于实现了字典元素的修改。

```
>>> school1={'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5':
70}
>>> school2={ 'class5': 78,'class6': 38}
```

```
>>> school1.update(school2)
>>> school1      #'class5'所对应的值取 school2 中'class5'所对应的值 78
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 78, 'class6
': 38}
```

(3) 使用字典对象 school1 的 update(关键字)方法将关键字转化为“键:值”对并添加到字典 school1 中。

```
>>> school2.update(class9=60,class10=50)
>>> school2
{'class5': 78, 'class6': 38, 'class9': 60, 'class10': 50}
```

【例 3-3】 找出一句英文中出现次数最多的字符,并输出其出现的位置。

```
s = "Great works are performed not by strength but by perseverance."
s="".join(s.split())
letter_count_dict=dict()                    #创建空字典,用于记录字符出现的次数
for i in s:
    if i in letter_count_dict:              #判断 i 是否是字典的键,若是则次数加 1
        letter_count_dict[i]+=1
    else:                                   #没出现过就是 1
        letter_count_dict[i] = 1
print("字符串中各字符出现的次数是:",end="")
print(letter_count_dict)

max_letter_occurrence=max(letter_count_dict.values())
print("最多的字符出现次数是:"+str(max_letter_occurrence))

#创建空列表,存储出现次数最多的字符,因为有可能是 1 个或多个
max_occurrence_letters=[]
for k,v in letter_count_dict.items():
    if v==max_letter_occurrence:            #找到出现次数最多的字符,存到列表中
        max_occurrence_letters.append(k)
print("出现次数最多的字符是:"+str(max_occurrence_letters))

for i in max_occurrence_letters:
    #创建记录出现次数最多的字符的出现位置的变量
    max_occurrence_letter_positions = []
    start_postion=0                         #从 0 开始找
    while True:
        postion=s.find(i,start_postion)
        if postion!=-1:                     #! =-1 表示找到了
            max_occurrence_letter_positions.append(postion)
            start_postion=postion+1         #更新下次查找的起始位置
        else:                   #当查找不到 i 所表示的字母的位置时,说明位置都找到了
            print("%s 字符出现的位置:%s" %(i,max_occurrence_letter_positions))
            break                           #终止 while 循环的执行
```

运行上述程序代码,所得的输出结果如图 3-1 所示。

```
字符串中各字符出现的次数是:{'G': 1, 'r': 8, 'e': 9, 'a': 3, 't': 5,
'w': 1, 'o': 3, 'k': 1, 's': 3, 'p': 2, 'f': 1, 'm': 1, 'd': 1, 'n'
: 3, 'b': 3, 'y': 2, 'g': 1, 'h': 1, 'u': 1, 'v': 1, 'c': 1, '.': 1
}
最多的字符出现次数是:9
出现次数最多的字符是:['e']
e字符出现的位置:[2, 12, 14, 20, 30, 41, 44, 46, 51]
```

图 3-1 输出结果


删除字典元素

### 3.2.4 删除字典元素

(1) 使用 del 命令可以删除字典中指定键的字典元素,也可以删除整个字典。

```
>>> dict3 = dict([('one', 1), ('two', 2), ('three', 3),('four', 4)])  #创建字典
>>> dict3
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> del dict3[ 'four']    #删除键是 'four'的字典元素
>>> dict3
{'one': 1, 'two': 2, 'three': 3}
```

(2) 用字典对象的 pop()方法删除指定键的字典元素并返回该键所对应的值。

```
>>> x=dict3.pop('three') #删除键'three'所对应的字典元素,返回'three'所对应的值
>>> print(x)
3
>>> dict3
{'one': 1, 'two': 2}
```

(3) 用字典对象的 popitem()方法随机删除字典中的元素并返回该元素的键和值组成的二元组。一般删除末尾字典对象的末尾元素。

```
>>> dict3.popitem()
('two', 2)
```

(4) 利用字典对象的 clear()方法清空字典的所有元素。

```
>>> dict3.clear()
>>> dict3
{}
```

### 3.2.5 复制字典

```
>>> dict1={'Jack': 18, 'Mary': 16, 'John': 20}  #创建字典
```

(1) 浅复制。调用字典对象的 copy()方法返回字典的浅复制。

执行 dict2=dict1.copy()语句后,dict2 和 dict1 指向不同的内存空间,当对 dict1 进行增删改查操作时,dict2 不会改变,反之亦然。但是,当字典里包含列表时,修改列表中的值,对应字典中的列表值也会改变。

```
>>> dict2=dict1.copy()
>>> print("dict1 的 id 是%s,dict2 的 id 是%s"%(id(dict1),id(dict2)))
dict1 的 id 是 48563616,dict2 的 id 是 48751816
```

```
>>> dict1['Jack']=28
>>> dict1
{'Jack': 28, 'Mary': 16, 'John': 20}
>>> dict2
{'Jack': 18, 'Mary': 16, 'John': 20}
>>> dict3={'Jack': [18,19], 'Mary': 16, 'John': 20}
>>> dict4=dict3.copy()
>>> dict3["Jack"].append(20)
>>> dict3
{'Jack': [18, 19, 20], 'Mary': 16, 'John': 20}
>>> dict4
{'Jack': [18, 19, 20], 'Mary': 16, 'John': 20}
```

(2) 深复制。需导入 copy 模块,执行 dict4 = copy.deepcopy(dict3) 语句后,dict3、dict4 指向的不是同一内存空间,对 dict3 做任何修改,dict4 的值都不会变化。

```
>>> import copy
>>> dict3={'Jack': [18,19], 'Mary': 16, 'John': 20}
>>> dict4 = copy.deepcopy(dict3)
>>> dict4
{'Jack': [18, 19], 'Mary': 16, 'John': 20}
>>> dict3["Jack"].append(20)
>>> dict3
{'Jack': [18, 19, 20], 'Mary': 16, 'John': 20}
>>> dict4
{'Jack': [18, 19], 'Mary': 16, 'John': 20}
```

### 3.2.6 字典推导式

字典推导(生成)式和列表推导式的用法是类似的。

```
>>> dict6 = {'physics': 1, 'chemistry': 2, 'biology': 3, 'history': 4}
#把 dict6 的每个元素键的首字母大写、键值变为 2 倍
>>> dict7 = { key.capitalize(): value * 2 for key,value in dict6.items() }
>>> dict7
{'Physics': 2, 'Chemistry': 4, 'Biology': 6, 'History': 8}
```

## 3.3 字典实战:使用 jieba 库统计《兼葭》的词频

首先,执行“pip install jieba”命令安装 jieba 库。jieba 库是一款优秀的 Python 第三方中文分词库,jieba 支持三种分词模式:精确模式、全模式和搜索引擎模式,三种模式的特点如下。

精确模式:试图将语句做最精确的切分,不存在冗余数据,适合做文本分析。

全模式:将语句中所有可能是词的词语都切分出来,速度很快,但是存在冗余数据。

搜索引擎模式:在精确模式的基础上,对长词再次进行切分。

```
>>> import jieba
>>> s = "关山难越,谁悲失路之人?萍水相逢,尽是他乡之客。"
>>> print("/".join(jieba.lcut(s)))
#jieba.lcut(s)精简模式,返回一个列表类型的结果
关山/难越/,/谁/悲失路/之/人/?/萍水相逢/,/尽是/他/乡/之客/。
>>> print("/".join(jieba.lcut(s, cut_all=True)))
#全模式,使用 'cut_all=True' 指定
关山/山难/越/,/谁/悲/失/路/之人/?/萍水/萍水相逢/相逢/,/尽是/他/乡/之/客/。
>>> print("/".join(jieba.lcut_for_search(s)))        #搜索引擎模式
关山/难越/,/谁/悲失路/之/人/?/萍水/相逢/萍水相逢/,/尽是/他/乡/之客/。
```

【例 3-4】 蒹葭词频统计。

```
import jieba
txt='''蒹葭苍苍,白露为霜。所谓伊人,在水一方。溯洄从之,道阻且长。溯游从之,宛在水中央。蒹葭萋萋,白露未晞。所谓伊人,在水之湄。溯洄从之,道阻且跻。溯游从之,宛在水中坻。蒹葭采采,白露未已。所谓伊人,在水之涘。溯洄从之,道阻且右。溯游从之,宛在水中沚。'''
txt = txt.replace(",","")
txt = txt.replace("。","")
words=jieba.lcut(txt)                              #精准模式
a={}
for word in words:
    a[word]=a.get(word,0)+1
items=list(a.items())                              #将字典转换为记录列表
items.sort(key=lambda x:x[1],reverse=True)  #按词频排序
for i in range(8):                                 #输出词频最高的 8 个词
    word,count=items[i]
    print("{0:<10}{1:<5}".format(word,count))
```

运行上述程序代码,所得的输出结果如下。

```
从        6
之        5
在        4
蒹        3
葭        3
白露      3
所谓      3
伊人      3
```

## 3.4 集合数据类型

集合数据类型 set 是 Python 的一种内置数据类型。集合是写在花括号{}之间、用逗号分隔开的元素集,集合中的元素互不相同,元素类型只能是不可变数据类型,如数字、元组、字符串等。

### 3.4.1 创建集合


创建集合

(1) 使用赋值操作直接将一个集合赋值给变量来创建一个集合。

```
>>> student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}
```

(2) 使用集合的构造函数 set()将列表、元组等其他可迭代对象转换为集合,如果原来的数据中存在重复元素,则在转换为集合的时候只保留一个。

```
>>> set1 = set('cheeseshop')
>>> set1
{'s', 'o', 'p', 'c', 'e', 'h'}
```

注意:创建一个空集合必须用 set()而不是{ },因为{ }是用来创建一个空字典的。


集合添加元素

## 3.4.2 集合添加元素

### 1. 集合单个添加元素

虽然集合中不能有可变元素,但是集合本身是可变的。也就是说,可以添加或删除集合中的元素。可以使用集合对象的 add()方法向集合添加单个元素。

```
>>> set3 = {'a', 'b'}
>>> set3.add('c')                                    #添加一个元素
>>> set3
{'b', 'a', 'c'}
```

### 2. 集合批量添加元素

使用集合对象的 update()方法向集合批量添加元素。

```
>>> set3.update(['d', 'e', 'f'])                     #将列表中的元素添加到集合中
>>> set3
{'a', 'f', 'b', 'd', 'c', 'e'}
>>> set3.update(['o', 'p'], {'l', 'm', 'n'})         #一次将列表和集合中的元素添加到
                                                     #set3 集合中
>>> set3
{'l', 'a', 'f', 'o', 'p', 'b', 'm', 'd', 'c', 'e', 'n'}
```

【例 3-5】 遍历输出集合元素。

```
weekdays = {'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'}
#for 循环在遍历 set 时,遍历的顺序和 set 中元素书写的顺序很可能是不同的
for d in weekdays:
    print (d,end=' ')
```

运行上述程序代码,所得到的输出结果如下。

```
THU TUE MON FRI WED SAT SUN
```


集合元素删除

## 3.4.3 集合元素删除

### 1. 使用集合对象的 discard(x)方法删除集合中的元素 x

如果元素 x 不存在于集合中,discard(x)方法不会抛出 KeyError;如果 x 存在于集合

中，则会移除 x 并返回 None。

```
>>> set4 = {1, 2, 3, 4,5}
>>> set4.discard(4)
>>> set4
{1, 2, 3, 5}
```

**2. 使用集合对象的 remove(x)方法删除集合中的元素 x**

如果元素 x 不存在于集合中，则会抛出 KeyError；如果 x 存在于集合中，则会移除 x 并返回 None。

```
>>> set4.remove(6)
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    set4.remove(6)
KeyError: 6
```

**3. 使用集合对象的 pop()方法从左边删除集合中的元素并返回删除的元素**

```
>>> set4.pop()
1
```

**4. 使用集合对象的 clear()方法删除集合的所有元素**

```
>>> set4.clear()
>>> set4
set()
```

### 3.4.4 集合运算

Python 中的集合支持交集、并集、差集、对称差集等集合运算。

```
>>>A={1,2,3,4,6,7,8}
>>>B={0,3,4,5}
```

交集：两个集合 A 和 B 的交集是由所有既属于 A 又属于 B 的元素所组成的集合，使用“&”操作符执行交集操作，同样地，也可使用集合对象的方法 intersection()完成，如下所示。

```
>>>A&B            #求集合 A 和 B 的交集
{3, 4}
>>> A.intersection(B)
{3, 4}
```

并集：两个集合 A 和 B 的并集是由这两个集合的所有元素构成的集合，使用“|”操作符执行并集操作，也可使用集合对象的方法 union() 完成，如下所示。

```
>>> A | B
{0, 1, 2, 3, 4, 5, 6, 7, 8}
>>> A.union(B)
{0, 1, 2, 3, 4, 5, 6, 7, 8}
```

差集：集合 A 与集合 B 的差集是所有属于 A 且不属于 B 的元素构成的集合，使用“－”操作符执行差集操作，也可使用集合对象的方法 difference()完成，如下所示。

```
>>> A - B
{1, 2, 6, 7, 8}
>>> A.difference(B)
{1, 2, 6, 7, 8}
```

对称差：集合 A 与集合 B 的对称差集是由只属于其中一个集合，而不属于另一个集合的元素组成的集合，使用“^”操作符执行对称差集操作，也可使用集合对象的方法 symmetric_difference() 完成，如下所示。

```
>>> A ^ B
{0, 1, 2, 5, 6, 7, 8}
>>> A.symmetric_difference(B)
{0, 1, 2, 5, 6, 7, 8}
```

子集：由集合中一部分元素所组成的集合，使用“<”操作符判断“<”左边的集合是否是“<”右边的集合的子集，也可使用集合对象的方法 issubset()完成，如下所示。

```
>>> C={1,3,4}
>>> C < A          #C 集合是 A 集合的子集,返回 True
True
>>> C.issubset(A)
True
>>> C < B
False
```

### 3.4.5 集合推导式

集合推导式跟列表推导式差不多，区别在于：集合推导式不使用方括号，而使用花括号；集合推导式得到的集合中无重复元素。

```
>>> a = [1, 2, 3, 4, 5]
>>> squared = {i * * 2 for i in a}
>>> print(squared)
{1, 4, 9, 16, 25}
>>> strings = ['All','things','in','their','being','are','good','for','
something']
>>> {len(s) for s in strings}       #长度相同的只留一个
{2, 3, 4, 5, 6, 9}
>>> {s.upper() for s in strings}
{'THINGS', 'ALL', 'SOMETHING', 'THEIR', 'GOOD', 'FOR', 'IN', 'BEING', 'ARE'}
```

## 3.5 集合实战：统计公司的各类人才都有谁？

【例 3-6】 公司的董事成员有李强、刘涛和孙涛，经理有李强、刘涛、韩冰和王飞。回答以下问题。

(1) 既是董事也是经理的有谁?

```
>>> set01 = {"李强","刘涛","孙涛"}
>>> set02 = {"李强","刘涛","韩冰","王飞"}
>>> print(set01  &  set02)        #输出既是董事也是经理的成员
{'刘涛', '李强'}
```

(2) 是董事,但不是经理的有谁?

```
>>> print(set01 - set02)
{'孙涛'}
```

(3) 是经理,但不是董事的有谁?

```
>>> print(set02 - set01)
{'王飞', '韩冰'}
```

(4) 韩冰是董事吗?

```
>>> print("韩冰" in set01)
False
```

(5) 身兼一职的都有谁?

```
>>> print(set02  ^  set01)
{'王飞', '孙涛', '韩冰'}
```

(6) 董事和经理总共有多少人?

```
>>> print(len(set02  | set01))
5
```

## 3.6 序列解包

创建列表、元组、集合、字典以及其他可迭代对象,称为“序列打包”,因为值被“打包到序列中”。“序列解包”是指将多个值的序列解开,然后放到变量的序列中。序列解包由一个“*”和一个序列连接而成,Python 解释器自动将序列解包成多个元素。下面给出四种通过元组解包获取元组的元素的方法,这些方法对列表、集合同样适用。

### 1. 一对一解包

解包是将元组中的元素一个或多个剥离出来。一对一解包,就是用与元组元素数量相同个数的变量,将元组中的元素分别赋给这些变量。例如:

```
>>> user = ("李白", "唐朝", "伟大的浪漫主义诗人")
>>> name, dynasty, feature = user     #一对一解包
>>> print("姓名:", name)
姓名: 李白
>>> print("朝代:", dynasty)
朝代: 唐朝
>>> print("特征:", feature)
特征: 伟大的浪漫主义诗人
```

### 2. 用下画线表示解包时不需要的元素

有时我们并不需要元组中所有的元素。比如之前的 user,只想获取姓名和特征。

```
>>> user = ("李白", "唐朝", "伟大的浪漫主义诗人")
>>> name, _, feature = user     #用下画线放在不想要朝代数据的位置
>>> print(f"姓名: {name}; 特征: {feature}")
姓名: 李白; 特征: 伟大的浪漫主义诗人
```

**注意**:下画线实际上是一个有效的 Python 变量名。实际上,它保存了朝代数据,如下所示。

```
>>> print(f"朝代: {_}")
朝代: 唐朝
```

不过与使用显示变量名相比,建议使用此用法,因为它清楚地表明不需要在此特定位置的值。

### 3. 使用星号(*)获取元组连续的若干元素

当元组包含多个元素时,有时我们可能想要获取连续的几个元素,这种情况就可以使用星号(*)来实现这一要求。

```
>>> nums = (1, 2, 3, 4, 5)
'''first_num 和 last_num 用于指定元组中的第一项和最后一项, *middle_nums 以列表的形式获取除元组中的第一个和最后一个元素以外的其余所有元素,即使该列表包含一个或零个元素'''
>>> first_num, *middle_nums, last_num = nums          #使用星号(*)捕获全部
>>> print("第一个数字:", first_num)
第一个数字: 1
>>> print("最后一个数字:", last_num)
最后一个数字: 5
>>> print("剩下的数字:", middle_nums)
剩下的数字: [2, 3, 4]
```

### 4. 使用星号解包整个元组

有时候,想创建一个包含现有元组的新元组,这种情况下我们可以在元组前面使用星号来解开元组中的所有元素。例如:

```
>>> group_factors1 = ("王勃", "杨炯")
>>> group_factors2 = (*group_factors1, "卢照邻", "骆宾王")
>>> print("group_factors2:", group_factors2)
group_factors2: ('王勃', '杨炯', '卢照邻', '骆宾王')
```

**注意**:解包字典的语法类似于解包元组、列表,不同之处在于使用两个星号(**)将字典解包为关键字参数。以下是一个简单的示例:

```
>>> person = {'name': 'Alice', 'age': 25, 'city': 'Beijing'}
>>> print("{name} is {age} years old and lives in {city}.".format(**person))
Alice is 25 years old and lives in Beijing.
```

```
>>> x,y,z = person          #字典解包默认是解包字典的键
>>> print(x,y,x)
name age name
```

## 3.7 日期格式和字符串格式相互转化

datetime是一个模块，它提供了日期格式和字符串格式相互转化的函数，datetime模块还包含一个datetime类。

### 3.7.1 字符串格式转化为日期格式

由字符串格式转化为日期格式的函数为datetime.datetime.strptime()。如果输入的日期和时间是字符串，要处理日期和时间，首先必须把str转换为datetime。转换方法是通过strptime()函数实现的，需要一个日期和要转换成的时间的格式化字符串，具体有如下两种实现方法。

**1. 时间.strftime(日期格式)**

```
>>> import datetime
#获取当前日期并转换成指定的日期格式
>>> datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
'2022-05-07 13:37:27'
```

**2. datetime.datetime.strptime(字符串,日期格式)**

```
>>> day = datetime.datetime.strptime('2022-5-7 13:26:45', '%Y-%m-%d %H:%M:%S
')
>>> day
datetime.datetime(2022, 5, 7, 13, 26, 45)
>>> print(day)
2022-05-07 13:26:45
>>> type(day)
<class 'datetime.datetime'>
```

Python中时间日期格式化符号如下。

%y：两位数的年份表示(00～99)。

%Y：四位数的年份表示(000～9999)。

%m：月份(01～12)。

%d：月内中的一天(0～31)。

%H：24小时制小时数(0～23)。

%I：12小时制小时数(01～12)。

%M：分钟数(00～59)。

%S：秒数(00～59)。

%a：本地简化的星期名称。
%A：本地完整的星期名称。
%b：本地简化的月份名称。
%B：本地完整的月份名称。
%c：本地相应的日期表示和时间表示。
%j：年内的一天(001～366)。
%p：本地 A.M.或 P.M.的等价符。
%U：一年中的星期数(00～53)星期天为星期的开始。
%w：星期(0～6),星期天为星期的开始。
%W：一年中的星期数(00～53)星期一为星期的开始。
%x：本地相应的日期表示。
%X：本地相应的时间表示。
%Z：当前时区的名称。

### 3.7.2 日期格式转化为字符串格式

由日期格式转化为字符串格式的函数为 datetime.datetime.strftime()。后台提取到 datetime 对象后,要想把它格式化为字符串显示给用户,就需要转换为 str,转换方法是通过 strftime()实现的,同样需要一个日期和时间的格式化字符串。

```
>>> s= datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d %H:%M
')
>>> s
'2022-05-07 13:49'
>>> type(s)
<class 'str'>
>>> print(s)
2022-05-07 13:49
```

## 3.8 循环中的 break、continue、pass 和 else

break 语句和 continue 语句提供了另一种控制循环的方式。break 语句用来终止循环语句,即循环条件没有 False 或者序列还没被完全遍历完,也会停止执行循环语句。如果使用嵌套循环,break 语句将停止执行最深层的循环,并开始执行下一行代码。continue 语句终止当前迭代而进行循环的下一次迭代。Python 的循环语句可以带有 else 子句,else 子句在序列遍历结束(for 语句)或循环条件为假(while 语句)时执行,但循环被 break 终止时不执行。

### 3.8.1 用 break 语句提前终止循环

可以使用 break 语句跳出最近的 for 或 while 循环。下面的 TestBreak.py 程序演示

了在循环中使用 break 的效果。

```
TestBreak.py
1.    sum=0
2.    for k in range(1, 30):
3.        sum=sum + k
4.        if sum>=200:
5.            break
6.
7.    print('k 的值为', k)
8.    print('sum 的值为', sum)
```

TestBreak.py 程序执行的结果如下所示。

```
k 的值为 20
sum 的值为 210
```

这个程序从 1 开始，把相邻的整数依次加到 sum 上，直到 sum 大于或等于 200。如果没有第 4 和第 5 行，这个程序将会计算 1～29 的所有数的和。但有了第 4 和第 5 行，循环会在 sum 大于或等于 200 时终止，跳出 for 循环。没有了第 4 和第 5 行，输出将会是：

```
k 的值为 29
sum 的值为 435
```

### 3.8.2 用 continue 语句提前结束本次循环

有时我们并不希望操作终止整个循环，而只希望提前结束本次循环，接着执行下次循环，这时可以用 continue 语句。当 continue 语句在循环结构中执行时，并不会退出循环结构，而是立即结束本次循环，重新开始下一轮循环，也就是说，跳过循环体中在 continue 语句之后的所有语句，继续下一轮循环。换句话说，continue 退出一次循环而 break 退出整个循环。下面通过例子来说明循环中使用 continue 的效果。

**【例 3-7】** 要求输出 100～200 之间的不能被 7 整除的数以及不能被 7 整除的数的个数。

**分析：** 本题需要对 100～200 之间的每一个整数进行遍历，这可通过一个循环来实现；对遍历中的每个整数，判断其能否被 7 整除，如果不能被 7 整除，就将其输出。

```
1.    n = 0
2.    for k in range(100, 201):
3.        if k%7==0:
4.            continue
5.        print(k, end=' ')
6.        n+=1
7.
8.    print('\n100～200 之间不能被 7 整除的整数一共有%d 个'%(n))
```

运行上述程序代码,所得的输出结果如下。

```
100 101 102 103 104 106 107 108 109 110 111 113 114 115 116 117 118 120 121 122 123
124 125 127 128 129 130 131 132 134 135 136 137 138 139 141 142 143 144 145 146 148
149 150 151 152 153 155 156 157 158 159 160 162 163 164 165 166 167 169 170 171 172
173 174 176 177 178 179 180 181 183 184 185 186 187 188 190 191 192 193 194 195 197
198 199 200
```

100～200 之间不能被 7 整除的整数一共有 87 个。

**程序分析:** 有了第 3 到第 4 行,当 k 能被 7 整除时,执行 continue 语句,流程跳转到表示循环体结束的第 7 行,第 5 到第 6 行不再执行。

### 3.8.3 pass 子句

有时候程序需要占一个位置(当前还没想好放什么语句,将来想好了再放),或者放一条语句,但又不希望这条语句做任何事情,此时就可以通过 pass 语句来实现。pass 语句是一个空操作语句,表示什么也不做。在执行到 pass 语句时,程序不会有任何操作,直接跳过并继续执行下一条语句。

【例 3-8】 条件语句中 pass 语句的使用举例。

```
age = int(input("请输入您的年龄: "))
if age > 18:
    pass
else:
    print("您还未成年!")
```

运行上述程序代码,所得的输出结果如下。

```
请输入您的年龄: 20
```

【例 3-9】 循环语句中 pass 语句的使用举例。

```
for i in range(5):
    if i == 3:
        pass
    else:
        print(i,end=",")
```

运行上述程序代码,所得的输出结果如下。

```
0,1,2,4,
```

### 3.8.4 循环语句的 else 子句

Python 的循环语句中可以带有 else 子句。在循环语句中使用 else 子句时,else 子句只有在序列遍历结束(for 语句)或循环条件为假(while 语句)时才执行,但在循环被 break 语句终止时不执行。带有 else 子句的 while 循环语句的语法格式如下所示。

```
while 循环继续条件:
    循环体
else:
    语句体
```

当 while 语句带有 else 子句时，如果 while 子句内嵌的“循环体”在整个循环过程中没有执行 break 语句(“循环体”中没有 break 语句，或者“循环体”中有 break 语句但始终未执行)，那么循环过程结束后，就会执行 else 子句中的语句体。否则，如果 while 子句内嵌的“循环体”在循环过程中一旦执行 break 语句，那么程序的流程将跳出循环结构，因为这里的 else 子句也是该循环结构的组成部分，所以 else 子句内嵌的“语句体”也就不会被执行了。

下面是带有 else 子句的 for 语句的语法格式。

```
for 控制变量 in 可遍历序列:
    循环体
else:
    语句体
```

与 while 语句类似，如果 for 语句在遍历所有元素的过程中，从未执行 break 语句，那么在 for 语句结束后，else 子句内嵌的“语句体”将得以执行；否则，一旦执行 break 语句，程序流程将连带 else 子句一并跳过。下面通过例子来说明循环中使用 else 子句的效果。

【例 3-10】 判断给定的自然数是否为素数。

```
import math
number = int(input('请输入一个大于 1 的自然数: '))
#math.sqrt(number)返回 number 的平方根
for i in range(2, int(math.sqrt(number))+1):
   if number % i == 0:
      print(number, '具有因子', i, ',所以', number,'不是素数')
      break                          #跳出循环,包括 else 子句
else:                                 #如果循环正常退出,则执行该子句
   print(number, '是素数')
```

运行上述程序代码，所得的输出结果如下。

```
请输入一个大于 1 的自然数: 28
28 具有因子 2 ,所以 28 不是素数
```

【例 3-11】 for 循环正常结束执行 else 子句。

```
for i in range(2, 11):
    print(i)
else:
    print('for statement is over.')
```

运行上述程序代码，所得的输出结果如下。

```
2,3,4,5,6,7,8,9,10,
for statement is over.
```

【例 3-12】 for 循环执行过程中被 break 语句终止时不会执行 else 子句。

```
for i in range(10):
    if(i == 5):
        break
    else:
        print(i, end=' ')
else:
    print('for statement is over')
```