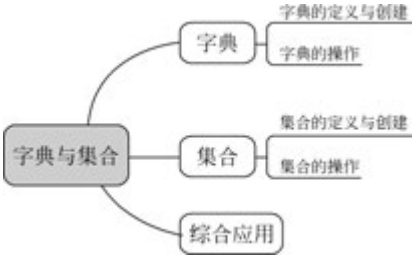
第5章 字典与集合

学习目标

- 自主探索字典与集合的基本概念及其特性。
- 精通字典与集合的构建方法及其相关操作技巧。
- 运用字典与列表的嵌套结构，以解决更复杂的应用场景为目标，提升实践能力。
- 精通集合的并集、交集与差集等集合运算。

章节框架



与列表和元组相比，字典(dict)与集合(set)是两种核心的数据结构。字典负责存储键值对(key-value pair)，并利用哈希表实现数据的快速定位，适用于需要高效查找、插入和删除操作的场景。集合作为一种无序且不包含重复元素的数据结构，它便于消除重复项并执行集合运算，例如并集、交集、差集等，从而提供了一种简洁而强大的问题解决方法。

5.1 字　　典

在第4章中，为处理复杂数据，列表结构会进一步地嵌套。例如，列表ls_user=[['name','Amy'],['age',25],['city','Nan Jing']]。然而，字典通过键值对的直接映射，提供了更为便捷的访问方式，尤其适用于处理具有多个属性的复杂数据(例如用户信息、产品信息等)，如dict_user={'name': 'Amy','age': 25,'city': 'Nan Jing'}。

5.1.1 字典的定义与创建

1. 字典的定义

字典使用花括号“{}”表示，键和值之间用冒号“:”分隔，键值对之间用逗号“,”分隔。其语法格式为：

```
字典 = { 键1 : 值1, 键2 : 值2, 键3 : 值3... }
```

【例5-1】 定义一个字典。

```
>>> d_price = {'电视': 3990, '洗衣机': 2890}      #家电与价格构成的字典
```

```
>>> type(d_price)
<class 'dict'>                                    #字典的数据类型为 dict
```

2. 字典的创建

在字典中,键必须采用不可变的数据类型,如字符串、布尔值、整数、浮点数以及元组。键应当保持其唯一性,若出现重复的键,则先前的键值对将被新的键值对所替代。至于字典的值,它可以是任何类型的数据。创建字典的常见方法包括使用花括号“{}”或使用 dict()函数。

1) 使用花括号“{}”创建

在花括号“{}”内以冒号“:”分隔键和值,以逗号“,”分隔不同的键值对。

```
>>> d1 = {}                                       #创建空字典
>>> d1
{}
>>> d2 = {"王一":[67,78,98],"陈可":[87,93,80],"李月":[76,83,92]}   #创建成绩的字典
>>> d2
{'王一': [67, 78, 98], '陈可': [87, 93, 80], '李月': [76, 83, 92]}
>>> d3 = {"Amy":168,"Jack":183,"Paul":175,"Amy":165}                #创建身高的字典
>>> d3
{'Amy': 165, 'Jack': 183, 'Paul': 175}                  #键重复时,最后的键值对将覆盖原来的
>>> d4 = {["王可"]:68.9,["李小"]:73.4}                  #字典中的键不能是列表(可变类型)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    d4 = {["王可"]:68.9,["李小"]:73.4}
TypeError: unhashable type: 'list'
```

2) 使用 dict()函数创建

(1) dict()函数的参数为空,可以创建空字典。

```
>>> d = dict()                                    #创建一个空字典
>>> d
{}
```

(2) dict()函数的参数为一个含有键值对的双元素序列,即序列中每个元素均包含两个元素,分别代表键和值,也可以创建字典。

```
>>> ls = [ ("Amy","江苏"), ("Paul","山东") ]          #将列表的两个元素组转换为字典
>>> d2 = dict(ls)
>>> d2
{'Amy': '江苏', 'Paul': '山东'}
>>> d3 = dict( ( ("计算机",5980),("手机",3999) ) )   #将一个两元素组元组转换为字典
>>> d3
{'计算机': 5980, '手机': 3999}
>>> d3 = dict( [12,34],[65,89] )                   #不能将两个元素组的列表转换为字典
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    d3 = dict([12,34],[65,89])
TypeError: dict expected at most 1 argument, got 2
>>> d5 = dict((["Amy",98,72],["Paul",81,69]))      #不能将一个三元素组元组转换为字典
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
```

```
    d5 = dict(("Amy",98,72],["Paul",81,69]))
ValueError: dictionary update sequence element          # 0 has length 3; 2 is required
```

使用 dict()函数时,括号里只能是一个序列,而不能是两个列表元素[12,34],[65,89],更不能是一个三元素组的序列。

(3) dict()函数使用关键字参数创建字典,关键字参数的键是固定的字符串(不需要显式地加引号),但不能是常量或关键字。关键字参数的值可以是任意类型,语法格式为:

```
字典 = dict( 键 1 = 值 1,键 2 = 值 2,键 3 = 值 3... )
>>> course_num = dict(Java = 150, Python = 180, C = 190)   # 创建课程和选课人数的字典
>>> print(course_num)
{'Java': 150, 'Python': 180, 'C': 190}
>>> dic_num = dict(11 = 'AB', 22 = 'cd', 33 = 'ef')        # 关键字参数的键不能是数字常量
SyntaxError: expression cannot contain assignment, perhaps you meant " == "?
```

(4) dict()函数还允许合并两个或多个字典来创建一个新字典。其语法格式为:

```
新字典 = dict(字典, ** 字典)
```

在这个表达式中,"字典"指的是一个已存在的字典对象,而 ** 字典指的是另一个待合并的字典。合并时,键名冲突会导致后续字典的键值对会更新原字典中的对应项,即后者的值会覆盖前者的值。

```
>>> d1 = {"aaa":33,"bbb":56}
>>> d2 = {"ccc":12,"ddd":78,"aaa":99}
>>> d3 = dict(d1, ** d2)
>>> d3
{'aaa': 99, 'bbb': 56, 'ccc': 12, 'ddd': 78}
>>> d1
{'aaa': 33, 'bbb': 56}
>>> d2
{'ccc': 12, 'ddd': 78, 'aaa': 99}
```

由上述示例可知,通过执行语句 dict(d1, ** d2),可以将 d2 中的关键字参数展开,并与 d1 进行合并,从而形成新的字典 d3。在此过程中,d1 与 d2 均保持其原始状态不变。若在合并过程中发现 d2 含有与 d1 相同的键,则 d2 中的键值对将取代 d1 中相应的键值对。

3) 使用字典生成式创建

生成式是构建字典的一种方法,特别适用于从其他数据结构(例如列表或元组)中迅速生成字典。其语法格式为:

```
字典 = {键表达式: 值表达式 for 变量 in 一个包含两个元素的序列迭代器}
```

【例 5-2】 字典生成式创建的示例。

```
>>> fruit = ['苹果','香蕉','西瓜','桃子']
>>> price = [86.5,34.5,72.8,45.2]
>>> fruit_price = { k : v for k, v in zip(fruit,price) }   # 遍历 zip 构成字典
>>> fruit_price
{'苹果': 86.5, '香蕉': 34.5, '西瓜': 72.8, '桃子': 45.2}
>>> fruit_price1 = dict(zip(fruit, price))
>>> fruit_price1
{'苹果': 86.5, '香蕉': 34.5, '西瓜': 72.8, '桃子': 45.2}
```

zip()函数作为参数接收多个可迭代序列,将两个列表按位置配对,生成一个可迭代的元组序列,例如:('苹果',86.5),('香蕉',34.5),('西瓜',72.8),('桃子',45.2)。遍历 zip()函数生成的元组序列,将每个元组的第一个元素作为键 k,第二个元素作为值 v,构建键值对。

字典创建的几种方法在功能上是等价的，花括号的方式在代码中更为直观，而 dict()函数在某些动态生成字典的场景下可能更为方便。

4）访问字典中的值

字典是一种无序的集合类型，因此无法通过索引或切片的方式进行访问。由于字典是一种键值对映射的数据结构，每个键都具有唯一性，因此可以利用方括号“[]”来访问字典中与特定键相关联的值。其语法格式为：

```
字典[ 键 ]
```

【例 5-3】 字典 d={"王一":[67,78,98],"陈可":[87,93,80],"李月":[76,83,92]}，其中存放了三位学生的英语、数学以及语文的成绩，使用键访问字典中的值。

```
>>> d["陈可"]                          #访问"陈可"的三门成绩
[87, 93, 80]
>>> d["李月"][1]                       #访问"李月"的数学成绩
83
>>> d[0]                               #字典是无序的,不能用索引访问值
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    d[0]
KeyError: 0
>>> d[[67,78,98]]                      #访问字典的值时,[]中只能是键,不能是值
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    d[[67,78,98]]
TypeError: unhashable type: 'list'
>>> d["陈课"]                          #访问字典的值时,[]中的键必须存在,否则报错
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    d["陈课"]
KeyError: '陈课'
```

访问字典中的值是一个很常见的操作，特别是在需要根据键快速检索数据时。字典提供了非常高效的方式来存储和访问键值对，使得数据操作变得简单快捷。

5.1.2 字典的操作

字典是一种可变的数据结构，其创建后允许进行修改、添加、删除以及查询操作，而无须构建一个新的字典实例。这一特性使得字典在处理复杂数据结构方面极具价值，特别是在需要动态更新数据的场合中。

1. 字典键值对的添加和修改

字典允许通过键值对的方式存储和检索数据。在字典中，添加和修改键值对是极为常见的操作。

1）直接赋值

通过直接为字典的键赋值，可以添加新的键值对或更新已有的键值对，其语法格式为：

```
字典[键] = 值
```

（1）语句：字典[键]=值，可以在字典中插入新的键值对。

```
>>> d = {}
>>> d["数学"] = 89
>>> d
```

```
{'数学': 89}
```

(2) 语句：字典[键] = 值，也可以更新已经存在的键对应的值。

```
>>> d = {"apple":98,"pear":87}
>>> d["apple"] = 39
>>> d
{'apple': 39, 'pear': 87}
```

2) update()方法

字典的 update()方法可以向字典中添加或修改键值对，其语法格式为：

```
字典.update( 一个字典或一个两元素组的列表/元组 )
```

(1) 如果待添加的键值对在字典中不存在，那么它将会被添加到字典中。

```
>>> student = {"name":"Amy","age":20}
>>> student.update( ( ("sex","boy"), ) )  #update()方法中参数是一个两元素的元组
>>> student
{'name': 'Amy', 'age': 20, 'sex': 'boy'}
>>> student = {"name":"Amy","age":20}
>>> student.update( [ ("sex","boy")] )    #update()方法中参数是一个两元素的列表
>>> student
{'name': 'Amy', 'age': 20, 'sex': 'boy'}
>>> student = {"name":"Amy","age":20}
>>> sex = {"sex":"boy"}
>>> student.update(sex )                  #update()方法中参数是一个字典
>>> student
{'name': 'Amy', 'age': 20, 'sex': 'boy'}
```

(2) 如果待添加的键值对在字典中已经存在，那么原有的值将会被新的值所覆盖。

```
>>> student = {'name': 'Amy', 'age': 20}  #update()方法中参数是一个两元素的元组
>>> student.update( ( ("age",23), ) )
>>> student
{'name': 'Amy', 'age': 23}
>>> student = {'name': 'Amy', 'age': 20}  #update()方法中参数是一个两元素的列表
>>> student.update([("age",28)])
>>> student
{'name': 'Amy', 'age': 28}
>>> student = {'name': 'Amy', 'age': 20}  #update()方法中参数是一个字典
>>> age = { "age":24 }
>>> student.update(age)
>>> student
{'name': 'Amy', 'age': 24}
```

添加和修改字典中的元素是一个非常灵活且高效的过程，通过直接赋值和 update()方法，可以轻松地管理字典中的数据。

2. 与字典相关的删除操作

在处理字典时，删除字典或删除字典中的键值对是一个很常见的操作，具体选择哪种方法取决于具体的应用场景和需求。

1) del 命令

(1) del 命令可以删除字典中指定的键值对，其语法格式为：

```
del 字典[键]
```

【例 5-4】 字典 operators={'+': 'Add','-': 'Sub','*': 'Mul','/': 'Div'}，使用 del 命令删除键值对。

```
>>> del operators['+']
>>> operators
{'-': 'Sub', '*': 'Mul', '/': 'Div'}
>>> del operators['**']                    #使用 del 命令,若删除的键不存在则会报错
>>> operators
Traceback (most recent call last):
  File "<pyshell#87>", line 1, in <module>
    del operators['**']
KeyError: '**'
```

(2) del 命令还可以直接删除字典,其语法格式为:

```
del 字典
```

【例 5-5】 字典 d={3: 4,5: 6},使用 del 命令删除字典。

```
>>> d = {3:4,5:6}
>>> del d
>>> d                                      #删除的字典不存在,所以不能访问
Traceback (most recent call last):
  File "<pyshell#99>", line 1, in <module>
    d
NameError: name 'd' is not defined
```

2) pop()方法

除了使用 del 命令,也可以使用 pop()方法来删除键值对。pop()方法不仅可以删除指定的键,还可以返回该键对应的值。其语法格式为:

```
字典.pop(键)
```

【例 5-6】 字典 operators={'+': 'Add','-': 'Sub','*': 'Mul','/': 'Div'},使用 pop()方法删除 operators 四则运算字典中的乘法'*'。

```
>>> operators = {'+': 'Add','-': 'Sub','*': 'Mul','/': 'Div'}
>>> operators.pop('*')
'Mul'
>>> operators
{'+': 'Add', '-': 'Sub', '/': 'Div'}
>>> operators = {'+': 'Add','-': 'Sub','*': 'Mul','/': 'Div'}
>>> operators.pop('//')                    #若删除字典中的键不存在则会报错
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    operators.pop('//')
KeyError: '//'
```

当删除字典中的键不存在时,pop()方法会报错,而 pop()方法还有一种带默认值的删除方法,在删除的键不存在时,不会报错。其语法格式为:

```
字典.pop(键,默认值)
```

因此,在删除不存在的键'//'时,相应的删除语句应书写如下:

```
>>> operators = {'+': 'Add','-': 'Sub','*': 'Mul','/': 'Div'}
>>> operators.pop('//',"该键不存在!")    #pop(键,默认值)
'该键不存在!'
```

3) popitem()方法

除了 del 命令和 pop()方法,还可以使用 popitem()方法来删除字典中最后被插入的键值对。这种方法在实现某些特定的算法或数据处理逻辑时非常有用。其语法格式为:

```
字典.popitem()
```

【例 5-7】 字典 fruit={'apple': 12,'banana': 22,'pear': 38 },使用 popitem()方法删除键值对。

```
>>> fruit = {'apple': 12, 'banana': 22, 'pear': 38 }
>>> fruit.popitem()
('pear', 38)
>>> fruit.popitem()
('banana', 22)
```

自 Python 3.7 起,字典维持了元素的插入顺序(即有序性)。因此,popitem()方法将删除最后被插入的键值对。在该版本之前,字典不具备顺序性,导致 popitem()方法删除的键值对具有不确定性。

4) clear()方法

通过调用 clear()方法,能够有效地删除字典内所有键值对,从而将该字典转换为一个空字典。这一操作极为实用,特别是在需要重新初始化字典或清除字典内容的场合。其语法格式为:

```
字典.clear()
```

【例 5-8】 字典 number={1: 2,3: 4,5: 6},使用 clear()方法清空字典。

```
>>> number = {1:2,3:4,5:6}
>>> number.clear()
>>> number
{}
```

通过使用 del 关键字、pop()方法和 popitem()方法,可以根据具体需求灵活地删除字典中的元素; clear()方法是一个非常便捷的工具,可以快速清空字典中的内容,使其变成一个空的容器。

3. 字典键值对的查询

键值对的检索可通过字典[键]实现,但是当检索的键不存在时,会抛出 KeyError 异常。为避免此类问题,可采用 get()方法。该方法在检索字典键时允许指定一个默认值,若键存在,则返回对应的值; 若键不存在,则返回预设的默认值。其语法格式为:

```
字典.get(键,默认值 = None)
```

【例 5-9】 字典 animal={'cat': 3,'dog': 6,'pig': 5},使用 get()方法查询键值对。

```
>>> animal = {'cat': 3, 'dog': 6, 'pig': 5}
>>> k = animal.get('bird')                #get(键),键不存在时,返回值为 None
>>> print(k)
None
>>> animal.get('kangaroo','不存在')       #get(键,默认值),键不存在时,返回默认值
'不存在'
>>> animal.get('cat')
3
```

【例 5-10】 给定一个学生参与志愿服务的记录 volunteerRecords=[{"姓名": "赵一", "activity": "社区清洁","hours": 3},{"姓名": "李三","activity": "敬老院服务", "hours": 5},{"姓名": "赵一","activity": "义卖活动","hours": 2},{"姓名": "王四", "activity": "社区清洁","hours": 4},{"姓名": "李三","activity": "义卖活动","hours": 3}],统计每个学生的志愿服务总时长,并找出时长最长的学生。

分析：

输入：参与志愿服务的记录 volunteerRecords。

输出：志愿服务时长最长的学生 maxStudent 和最长时长 maxHours。

算法：

(1) 初始化一个空字典 total，用于存储每个学生的总时长。

(2) 遍历 volunteerRecords，对于每一条记录，提取学生的姓名和时长。如果学生姓名已经存在于 total 中，则累加时长；否则，初始化该学生的时长为当前时长，遍历 total，比较每个学生的总时长，更新 maxStudent 和 maxHours。输出每个学生的总时长和时长最长的学生。

程序：

```
volunteerRecords = [
    {"姓名": "赵一", "activity": "社区清洁", "hours": 3},
    {"姓名": "李三", "activity": "敬老院服务", "hours": 5},
    {"姓名": "赵一", "activity": "义卖活动", "hours": 2},
    {"姓名": "王四", "activity": "社区清洁", "hours": 4},
    {"姓名": "李三", "activity": "义卖活动", "hours": 3}]
total = {}
for record in volunteerRecords :
    name = record["姓名"]
    hours = record["hours"]
    if name in total:
        total[name] += hours
    else:
        total[name] = hours
#找出时长最长的学生
maxStudent = ''
maxHours = 0
for student, hours in total.items():
    if hours > maxHours:
        maxStudent = student
        maxHours = hours
print("学生志愿服务总时长：", total)
print("志愿服务时长最长的学生：", maxStudent)
```

运行结果：

```
学生志愿服务总时长：{'赵一': 5, '李三': 8, '王四': 4}
志愿服务时长最长的学生：李三
```

【例 5-11】 某社交平台上的用户评论数据中，存在一些负面情绪词汇(如“糟糕”“失望”“投诉”等)。用户评论数据 comments＝["这次购物体验真的很糟糕，物流太慢了。","产品质量太差，完全不符合描述，非常失望。","客服态度恶劣，根本不解决问题，我要投诉!","商品还不错，但包装太简陋了，有点失望。","完全没有达到预期，简直是浪费钱。","物流很快，但商品质量一般，有点失望。","服务态度差，投诉无门，太糟糕了。","这次体验还不错，没有大问题。"]，负面情绪词汇列表 negativeWords＝["糟糕","失望","投诉","差","恶劣","浪费","不符合"]，平台希望通过统计这些词汇的出现频率，分析用户情绪，从而优化服务和产品。

分析：

输入：用户评论数据 comments 和负面情绪词汇列表 negativeWords。

输出：负面情绪词汇出现次数的字典 count。

算法：遍历每条评论，统计每个负面词汇在所有评论中出现的次数，并输出每个词汇的出现次数。

程序：

```
comments = ["这次购物体验真的很糟糕，物流太慢了。","产品质量太差，完全不符合描述，非常失望。","客服态度恶劣，根本不解决问题，我要投诉!","商品还不错，但包装太简陋了，有点失望。","完全没有达到预期，简直是浪费钱。","物流很快，但商品质量一般，有点失望。","服务态度差，投诉无门，太糟糕了。","这次体验还不错，没有大问题。"]
negativeWords = ["糟糕", "失望", "投诉", "差", "恶劣", "浪费", "不符合"]
count = {}
for w in negativeWords:
    for s in comments:
        if w in s:
            if w in count:
                count[w] += 1
            else:
                count[w] = 1
for k,v in count.items():
    print(f"{k}出现{v}次")
```

运行结果：

```
糟糕出现 2 次
失望出现 3 次
投诉出现 2 次
差出现 2 次
恶劣出现 1 次
浪费出现 1 次
不符合出现 1 次
```

4. 字典的遍历

在 Python 中，字典的遍历是一个常见的操作。字典以键值对的形式存储数据，因此在执行字典的遍历过程中，可以选择对字典键的遍历、对字典值的遍历或者对字典中的键值对的联合遍历。每种遍历策略均具有其特定的应用场景和优势，具体如表 5-1 所示。

表 5-1　字典遍历方法对比

遍历方法	语　法	返回值	适用场景
遍历字典键	for key in dict：	字典的键	只需要访问字典的键时使用
	for key in dict.keys()：	字典的键	同上，更明确地表示遍历键
遍历字典值	for value in dict.values()：	字典的值	只需要访问字典的值时使用
遍历字典键值对	for key,value in dict.items()：	键值对（元组形式）	需要同时访问键和值时使用

1）遍历字典键

遍历字典键的操作可以使用字典的 keys()方法或者直接遍历字典对象来获取键。

【例 5-12】 遍历字典键的示例。

程序：

```
#第一种直接遍历字典键的方式
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for k in fruit:                              #字典名作为迭代器
    print(k)
```

```
#第二种使用 keys()方法遍历字典键的方式
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for k in fruit.keys():                    #字典的 keys()方法作为迭代器
    print(k)
```

运行结果：

```
apple
pear
orange
```

```
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for k in fruit.keys():                    #字典的 keys()方法作为迭代器
    print(f"{k:<8}{ fruit[k]:<8 }")
```

运行结果：

```
apple   red
pear    yellow
orange  orange
```

2）遍历字典值

遍历字典值的操作可以使用字典的 values()方法来获取字典中所有的值。

【例 5-13】 遍历字典值的示例。

程序：

```
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for v in fruit.values():                  #字典的 values()方法作为迭代器
    print(v)
```

运行结果：

```
red
yellow
orange
```

3）遍历字典键值对

遍历字典键值对的操作可以使用字典的 items()方法同时遍历键和值。

【例 5-14】 遍历字典键值对的示例。

程序：

```
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for k,v in fruit.items():                 #字典的 items()方法作为迭代器
    print(f"{k:<8}{v:<8}")
```

运行结果：

```
apple   red
pear    yellow
orange  orange
```

程序：

```
#遍历字典键值对,item 为键和值的双元素元组
fruit = {"apple":"red","pear":"yellow","orange":"orange"}
for item in fruit.items():
    print(f"{item}")
```

运行结果：

```
('apple', 'red')
('pear', 'yellow')
('orange', 'orange')
```

在 Python 中进行字典的遍历操作时，应当依据具体需求选择合适的遍历方法，从而提升编程的效率与代码的可读性。

5. 字典的排序

在特定情境下，可能需要对字典内的元素执行排序操作，以便于数据的检索或展示。排序操作可以依据键或值来进行，鉴于字典本质上是无序的，可以通过将字典转换为列表来实现间接排序，或者利用内置的 sorted()函数直接对字典进行排序。

1）直接排序

sorted()函数接收一个字典名作为参数，它将返回一个按键排序的列表。此外，sorted()函数也可依据键值对对字典进行排序，并返回一个包含键值对的列表。其语法格式为：

```
列表 = sorted(字典) 或 列表 = sorted(字典.items())
```

【例 5-15】 字典 dicScore={"John":[79,67,88],"Amy":[87,95,73],"Paul":[65,82,90]}中存储了每个学生的英语、数学、计算机三门成绩，根据键对字典排序。

程序一：

```
#仅对字典中的键排序
ls_stuScore = sorted(dicScore)
print(ls_stuScore)
```

运行结果：

```
['Amy', 'John', 'Paul']
```

程序二：

```
#根据键对字典中的键值对排序
ls_stuScore = sorted(dicScore.items())
print(ls_stuScore)
```

运行结果：

```
[('Amy', [87, 95, 73]), ('John', [79, 67, 88]), ('Paul', [65, 82, 90])]
```

2）间接排序

字典由键值对构成，而列表则为有序元素的集合。通过转换字典为列表，能够借助列表的排序功能，对字典内的元素进行排序。

【例 5-16】 字典 dicScore={"John":[69,67,88],"Amy":[87,95,73],"Paul":[75,82,90]}中存储了每个学生的英语、数学、计算机成绩，根据学生的英语成绩进行升序排序。

程序：

```
dicScore = {"John":[69,67,88],"Amy":[87,95,73],"Paul":[75,82,90]}
ls_tmp = [ (v,k) for k,v in dicScore.items() ]                    #列表生成式交换键和值
ls_tmp.sort()                                                     #列表排序
ls_stuScore = [(v,k) for k,v in ls_tmp]                           #排序后变成键值顺序
print(dict(ls_stuScore))                                          #转换为字典输出排序结果
```

运行结果：

```
{'John': [69, 67, 88], 'Paul': [75, 82, 90], 'Amy': [87, 95, 73]}
```

6. 字典的运算

在 Python 中，字典可以使用一些运算符来进行运算。表 5-2 是一些常用的运算符。

表 5-2　字典相关的运算符

运　算　符	用　　途	示　　例
|	合并 Python 3.9 及以上版本	d1={1：2,3：4,5：6} d2={7：8,9：10} d1|d2 {**1：2,3：4,5：6,7：8,9：10**}
in/not in	键在/不在字典中	d={1：2,3：4,5：6} '1' not in d **True** 2 in d **False**

5.2　集　　合

在 Python 中，集合这一数据类型具备无序性和元素唯一性的特点。这意味着集合内的元素不会重复出现，并且集合中元素的排列顺序是不确定的。在需要消除重复元素以及进行集合运算的场景中，集合这一数据结构显得特别适用。

5.2.1　集合的定义与创建

1. 集合的定义

集合是一种无序且不包含重复元素的数据类型，它使用花括号“{}”表示，每个元素之间用逗号“,”分隔。

```
>>> s = { 1,2,3,4,5 }
>>> s
{1, 2, 3, 4, 5}
>>> type(s)
<class 'set'>
```

2. 集合的创建与访问

1）直接使用花括号创建

（1）集合中的元素是无序的，不能通过索引或切片的方式访问。

```
>>> s = {2,9,0,6,3}
>>> s
{0, 2, 3, 6, 9}                                    #集合的元素是无序的
>>> s[1]                                           #不能通过索引访问
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s[1]
TypeError: 'set' object is not subscriptable
```

(2) 集合中的元素不能是可变类型,例如,列表、字典或集合。

```
>>> s1 = { [1,2],6,"ab"}
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    s1 = {[1,2],6,"ab"}
TypeError: unhashable type: 'list'
>>> s2 = {{1:2},3,'ab'}
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    s2 = {{1:2},3,'ab'}
TypeError: unhashable type: 'dict'
>>> s3 = {{1,2,3},3,4}
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    s3 = {{1,2,3},3,4}
TypeError: unhashable type: 'set'
```

2) 使用 set()函数创建

使用 set()函数可以创建一个空集合,也可以将一系列元素作为参数传递给 set()函数。其语法格式为:

```
集合 = set([序列])
```

其中,序列为可选参数,可以是一个可迭代对象(如列表、元组、字符串等)。若参数为空,则创建一个空集合。

```
>>> s = set()
>>> s
set()
>>> type(s)
<class 'set'>
>>> s1 = set("Python")                    #将字符串转换为集合
>>> s1
{'P', 't', 'o', 'n', 'h', 'y'}
>>> s2 = set(['a',"bc","def"])            #将列表转换为集合
>>> s2
{'a', 'bc', 'def'}
>>> s3 = set(range(5))                    #将 range 转换为集合
>>> s3
{0, 1, 2, 3, 4}
>>> s4 = set((1,))                        #将元组转换为集合
>>> s4
{1}
>>> s5 = set(7654)                        #数字不是可迭代对象
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    s5 = set(7654)
TypeError: 'int' object is not iterable
>>> s6 = set([1,2,1,3,2,4,5])
>>> s6
{1, 2, 3, 4, 5}                           #去除重复元素
```

集合中允许出现重复元素,但是集合的结果必将去除重复项。

5.2.2 集合的操作

1. 集合元素的添加

1) add()方法

在 Python 中,可以使用集合来存储无序且不重复的元素。要向集合中添加元素,通过

调用 add()方法,可以将单个元素添加到集合中。其语法格式为:

```
集合.add(元素)
```

【例 5-17】 已知集合 s={1,2,3},使用 add()方法添加元素的示例。

```
>>> s = {1,2,3}
>>> s.add("abc")
>>> s
{1, 2, 3, 'abc'}
>>> s.add(456)
>>> s
{1, 2, 3, 'abc', 456}
>>> s.add([6,7])
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    s.add([6,7])
TypeError: unhashable type: 'list'
```

如果尝试添加一个已经存在于集合中的元素,则集合将不会进行任何操作,因为集合不允许有重复的元素。

```
>>> s1 = {1,2,3}
>>> s1.add(2)                          #已经存在于集合中的元素,集合将不会进行任何操作
>>> s1
{1, 2, 3}
```

2) update()方法

集合还可以通过 update()方法一次性地加入多个元素。它能接收一个可迭代对象作为其参数,并将该对象内的元素逐一添加至集合中,重复元素不添加。其语法格式为:

```
集合.update(可迭代对象)
```

【例 5-18】 集合 s = {2,3,4},使用 update()方法向集合中添加元素。

```
>>> s = {2,3,4}
>>> s.update([3,4,5])                  #重复元素不能添加
>>> s
{2, 3, 4, 5}
>>> s.update("abcd")
>>> s
{'d', 2, 3, 4, 5, 'c', 'b', 'a'}
>>> s.update(4567)                     #只有可迭代对象才能添加
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    s.update(4567)
TypeError: 'int' object is not iterable
>>> s.update({6:7,8:9})
>>> s
{'d', 1, 2, 3, 4, 5, 6, 8, 'c', 'b', 'a'}
```

使用 update()方法传入一个字典时,字典的键会被添加到集合中,而字典的值会被忽略。这是因为集合只存储单个元素,而字典是键值对的映射结构。

2. 集合元素的删除

集合包含多种方法,用于从其内部删除元素。根据不同的需求,这些方法能够从集合中删除相应的元素。

1）remove()方法

remove()方法可以从集合中删除一个指定的元素。如果该元素不存在于集合中，则会抛出一个 KeyError 异常。其语法格式为：

```
集合.remove(元素)
```

```
>>> s = {1, 2, 3, 4}
>>> s.remove(2)                          # 元素不存在，则会抛出一个 KeyError 异常
>>> s
{1, 3, 4}
>>> s = {1,2,3,4}
>>> s.remove('2')
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    s.remove('2')
KeyError: '2'
```

2）discard()方法

discard()方法可以有效地从集合中删除一个元素，而不会抛出任何异常，即使该元素不存在于集合中也不会抛出异常。

```
>>> s = {1, 2, 3, 4}
>>> s.discard(3)                         # 删除元素 3
>>> s
{1, 2, 4}
>>> s.discard(5)                         # 尝试删除不存在的元素 5，不会抛出异常
>>> s
{1, 2, 4}
```

此方法在处理集合元素时极为有效，尤其适用于元素是否存在于集合中尚不明确的情形。

3）pop()方法

pop()方法会从集合中随机删除并返回一个元素。如果集合为空，则调用 pop()方法会抛出 TypeError 异常。其语法格式为：

```
集合.pop()
```

```
>>> s = {1, 2, 3, 4}
>>> s.pop()                              # 删除并返回一个任意元素
1
>>> s
{2, 3, 4}
```

4）clear()方法

集合中的 clear()方法是一个非常实用的功能，它用于清空集合中的所有元素，使得集合变成一个空集合。其语法格式为：

```
集合.clear()
```

```
>>> s = {1, 2, 3, 4}
>>> s.clear()                            # 清空集合
>>> s
set()
```

3. 集合的遍历

集合是一种用于存储不重复元素的数据类型。由于集合内的元素不具备顺序性，在遍历集合时，无法借助索引像操作列表或元组那样访问各个元素，因此，唯一可行的方法是通过 for 循环直接访问集合中的每个元素。例如：

```
s = {6, 2, 7, 4, 0, 6, 7, 3}
for i in s:
    print(i,end = ' ')
```

运行结果：

```
0 2 3 4 6 7
```

集合具有无序性，其元素在遍历时的顺序可能与添加顺序不同。此外，集合中的元素是唯一的，因此不会出现重复的值。

【例 5-19】 在科研过程中，产生的实验数据 data=[{"实验序号"：1,"实验结果"：0.92}，{"实验序号"：2,"实验结果"：0.82},{"实验序号"：3,"实验结果"：0.93},{"实验序号"：1,"实验结果"：0.91},{"实验序号"：4,"实验结果"：0.87},{"实验序号"：2,"实验结果"：0.85},{"实验序号"：5,"实验结果"：0.79},{"实验序号"：3,"实验结果"：0.94}]，为了确保数据的准确性，需要对实验数据进行去重处理，仅保留最后一次出现的实验数据。

分析：

输入：一个包含多个字典的列表 data，每个字典包含两个键值对。

中间值：用于记录已处理的实验序号 recordNum。

输出：一个去重后的列表 uniqueData。

算法：遍历实验数据 data，如果实验序号未出现过，则直接添加到 uniqueData，recordNum 记录该序号；如果序号已存在，则找到结果列表中的旧数据替换为当前数据。

程序：

```
data = [{"实验序号":1,"实验结果":0.92},{"实验序号":2,"实验结果":0.82},{"实验序号":3,"实验结果":0.93},{"实验序号":1,"实验结果":0.91},{"实验序号":4,"实验结果":0.87},{"实验序号":2,"实验结果":0.85},{"实验序号":5,"实验结果":0.79},{"实验序号":3,"实验结果":0.94}]
uniqueData = []                          #用于存储去重后的数据
recordNum = set()                        #用于记录已处理的实验序号
for i in data:
    if i["实验序号"] not in recordNum:
        uniqueData.append(i)             #添加到结果列表
        recordNum.add(i["实验序号"])     #记录实验序号
    else:
        for idx, item in enumerate(uniqueData):#如果序号已存在，则替换为新数据
            if item["实验序号"] == i["实验序号"]:
                uniqueData[idx] = i      #替换为新数据
                break
for i in uniqueData:
    print(i)
```

运行结果：

```
{'实验序号': 1, '实验结果': 0.91}
{'实验序号': 2, '实验结果': 0.85}
{'实验序号': 3, '实验结果': 0.94}
{'实验序号': 4, '实验结果': 0.87}
{'实验序号': 5, '实验结果': 0.79}
```

4. 集合的运算

集合是一种用于存储不重复元素的数据类型。它提供了多种运算，以便于进行集合间的操作。这些运算包括并集、交集、差集和对称差集等。

1）集合元素的判断

使用 in 和 not in 运算符判断某个元素是否属于该集合。

```
>>> s = {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> 5 in s                              #5 确实存在于集合 s 中
True
>>> '5' in s                            #'5' 确实不存在于集合 s 中
False
>>> [5] in s                            #集合中不可能有列表,这句会报错
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    [5] in s
TypeError: unhashable type: 'list'
```

使用 in 和 not in 运算符，能够在 Python 中高效地进行集合元素的归属判断，进而执行后续的逻辑操作和数据处理。

2）并集运算

并集运算可以将两个集合中的所有元素合并在一起，去除重复的元素。在 Python 中，使用竖线符号(|)或者 union()方法来实现并集运算，并集运算如图 5-1 所示。

```
>>> s1 = {7,8,9}
>>> s2 = {5,6,7}
>>> s1|s2
{5, 6, 7, 8, 9}
>>> s1.union(s2)
{5, 6, 7, 8, 9}
```

3）交集运算

交集运算用于找出两个集合中共同的元素，可以使用与符号(&)或者 intersection()方法来实现，交集运算如图 5-2 所示。

```
>>> s3 = {4,6,8}
>>> s4 = {5,7,8}
>>> s3&s4
{8}
>>> s3.intersection(s4)
{8}
```

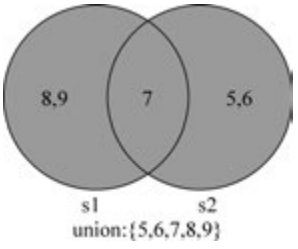


图 5-1　s1 与 s2 的并集

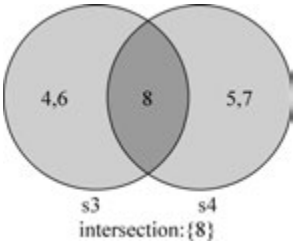


图 5-2　s3 与 s4 的交集

4）差集运算

差集运算用于找出在一个集合中但不在另一个集合中的元素，可以使用减号符号(－)或者 difference()方法来实现，差集运算如图 5-3 和图 5-4 所示。

```
>>> s5 = {6,7,8}
```

```
>>> s6 = {4,6,9}
>>> s5 - s6
{8, 7}
>>> s5.difference(s6)
{8, 7}
>>> s6 - s5
{9, 4}
```

注意：s5－s6 和 s6－s5 不一样。

5）对称差集运算

对称差集运算用于找出包含了两个集合中不共有的元素，也就是说，它包括了属于其中一个集合但不属于另一个集合的所有元素。可以使用异或符号(^)或者 symmetric_difference()方法来实现，对称差集运算如图 5-5 所示。

```
>>> s7 = {2,3,4,5}
>>> s8 = {3,5,7,8}
>>> s7^s8
{2, 4, 7, 8}
>>> s7.symmetric_difference(s8)
{2, 4, 7, 8}
```

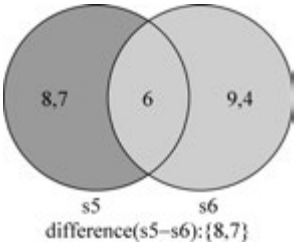


图 5-3　s5 与 s6 的差集

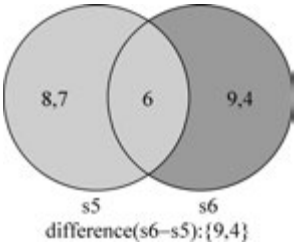


图 5-4　s6 与 s5 的差集

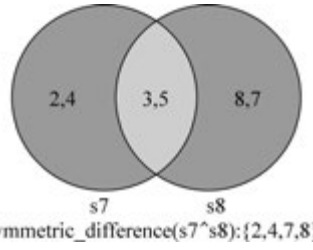


图 5-5　浅色为 s7 与 s8 的对称差集

【例 5-20】 为加强网络信息甄别，某平台记录了两天用户举报的“疑似谣言”词汇：d1＝{"AI 替代人类","芯片断供","全球停电","量子黑客","生物武器","货币崩盘"}，d2＝{"全球停电","气候武器","AI 替代人类","网络洗脑","5G 监控","量子计算崩溃"}。编程要求：

（1）输出连续两天均出现的疑似谣言。

（2）输出仅在第二天新出现的疑似谣言。

程序：

```
d1 = {"AI 替代人类", "芯片断供", "全球停电", "量子黑客", "生物武器", "货币崩盘"}
d2 = {"全球停电", "气候武器", "AI 替代人类", "网络洗脑", "5G 监控", "量子计算崩溃"}
commonRumors = d1 & d2
print("连续两天均出现的疑似谣言：", commonRumors)
newRumors = d2 - d1
print("仅在第二天新出现的疑似谣言：", newRumors)
```

运行结果：

```
连续两天均出现的疑似谣言：{'AI 替代人类', '全球停电'}
仅在第二天新出现的疑似谣言：{'5G 监控', '量子计算崩溃', '气候武器', '网络洗脑'}
```

5.3 综 合 应 用

Python 中的字典和集合是功能强大的数据结构，能高效解决多种问题。字典以键值对存储数据，优化了数据检索、添加和删除的效率，适合管理唯一数据项。集合是不重复元素的数据结构，提供并集、交集、差集等操作，便于处理复杂数据集。

【例 5-21】 已知英文句子为 s="The cat chased the mouse around the house，but the mouse was too quick for the cat to catch."，现需统计该句子中各单词出现的频次，不区分大小写，并将统计结果中频次最高的前三个单词和出现的频次输出。

分析：

输入：一个英文句子 s。

输出：单词频次最高的前三个单词和出现的频次。

算法：

(1) 移除标点符号，将输入句子转换为小写或大写，确保统计时不区分大小写。使用空格将句子拆分为单词列表。

(2) 遍历单词列表，如果单词已在字典中，则将其对应的频次加 1。如果单词不在字典中，则将其添加到字典并将频次设为 1。

(3) 将字典中的单词按照频次升序排序，输出频次最高的前三个单词和出现的频次。

程序一：

```
s = "The cat chased the mouse around the house, but the mouse was too quick for the cat to catch."
s = s.lower().replace('.', '').replace(',', '')
#将句子拆分为单词列表
words = s.split()
#创建一个空字典来存储单词出现的频次
word_count = {}
#统计每个单词出现的频次
for word in words:
    if word in word_count:
        word_count[word] += 1
    else:
        word_count[word] = 1
#将统计结果按照频次升序排列
ls = [(v,k) for k,v in word_count.items()]
sorted_word_count = sorted(ls)
#输出
for count,word in sorted_word_count[-1:-4:-1]:
    print(f"{word}: {count}")
```

运行结果：

```
the: 5
mouse: 2
cat: 2
```

将字典项转换为元组列表，先交换键值位置，然后使用 sorted()函数进行排序。它通过列表推导式 ls=[(v,k) for k,v in word_count.items()]将字典转换为以频次为第一元素的元组列表，然后进行排序。

程序二：

```
s = "The cat chased the mouse around the house, but the mouse was too quick for the cat to catch."
s = s.lower().replace('.', '').replace(',', '')
#将句子拆分为单词列表
words = s.split()
#创建一个空字典来存储单词出现的频次
word_count = {}
#统计每个单词出现的频次
for word in words:
    word_count[word] = word_count.get(word,0) + 1
#将统计结果按照频次升序排列
sorted_word_count = sorted(word_count.items(), key = lambda item: item[1])
#输出
for word, count in sorted_word_count[ - 1: - 4: - 1]:
    print(f"{word}: {count}")
```

直接在字典项上使用 sorted()函数，通过指定 key＝lambda item：item[1]按值（频次）排序更简洁，代码更直观。

【例 5-22】 设计一个商品库存管理系统，用字典存储商品信息。每个商品包含“名称”“库存数量”“单价”（单位：元）。完成以下要求。

（1）根据表 5-3 创建一个商品列表，每个商品用字典表示。

（2）计算并输出所有商品的总库存价值（库存数量×单价的总和）。

（3）按库存数量对商品从高到低排序，并输出排序后的商品列表。

表 5-3　商品信息

名　　称	库 存 数 量	单价/元
帽子	100	59
裤子	80	129
鞋子	150	299
袜子	200	19

分析：

输入：商品列表 products。

输出：商品的总库存价值 total_value，对 products 按照库存数量进行降序排序后的商品列表。

算法：遍历 products 列表，计算每个商品的库存价值，并累加到 total_value。使用 sorted()函数对 products 按照库存数量进行降序排序。

程序一：

```
products = [{"名称": "帽子", "库存数量": 100, "单价": 59},
    {"名称": "裤子", "库存数量": 80,  "单价": 129},
    {"名称": "鞋子", "库存数量": 150, "单价": 299},
    {"名称": "袜子", "库存数量": 200, "单价": 19}]
#计算并输出所有商品的总库存价值
total_value = 0
for product in products:
    total_value += product["库存数量"] * product["单价"]
print(f"所有商品的总库存价值: {total_value}元")
#按库存数量对商品按降序排序，并输出排序后的商品列表
products_new = []
```

```
for i in products:
    ls = [(v,k) for k,v in list(i.items())]
    products_new.append(ls)
products_new = [(b,a,c) for a,b,c in products_new]
products_new.sort(reverse = True)
products_new = [(b,a,c) for a,b,c in products_new]
print("排序后的商品列表:")
for item in products_new:
    print(f"名称: {item[0][0]:<4}库存: {item[1][0]:<5}单价: {item[2][0]:<5}")
```

运行结果：

```
所有商品的总库存价值: 64870 元
排序后的商品列表:
名称: 袜子   库存: 200   单价: 19
名称: 鞋子   库存: 150   单价: 299
名称: 帽子   库存: 100   单价: 59
名称: 裤子   库存: 80    单价: 129
```

这种方法将每个商品字典的键值对转换为一个元组列表，并将其添加到 products_new 列表中。重组元组列表，以便可以根据库存数量进行排序。使用 sorted(reverse＝True)对 products_new 列表进行降序排序，并输出排序后的商品信息。

程序二：

```
products = [{"名称": "帽子", "库存数量": 100, "单价": 59},
    {"名称": "裤子", "库存数量": 80,  "单价": 129},
    {"名称": "鞋子", "库存数量": 150, "单价": 299},
    {"名称": "袜子", "库存数量": 200, "单价": 19}]
#创建商品列表,每个商品用字典表示
total_value = 0
for product in products:
    total_value += product["库存数量"] * product["单价"]
print(f"所有商品的总库存价值: {total_value}元")
#按库存数量对商品按降序排序,并输出排序后的商品列表
sorted_products = sorted(products, key = lambda x: x["库存数量"], reverse = True)
print("排序后的商品列表:")
for product in sorted_products:
    print(f"名称: {product['名称']:<4}库存: {product['库存数量']:<5}单价: {product['单价']:<5}")
```

这里使用匿名函数，即 lambda 函数，具体为 lambda x：x["库存数量"]。该函数对于每一个传入的商品字典 x，将依据字典内的“库存数量”字段作为排序的主要依据，从而实现基于商品库存数量的排序。在后续的函数章节中，将继续探讨匿名函数的相关内容。

【例 5-23】 图书系统中存放了各类图书、图书借还状态以及用户借书还书的记录，图书馆的图书分类与图书数量在字典 library_books＝{"文科"：{"book1"：9,"book2"：0,"book3"：15},"理科"：{"book4"：10,"book5"：5,"book6"：0},"工科"：{"book7"：0,"book8"：7,"book9"：12}}中，用户借阅记录在字典 borrowed_records＝{"王一"：{"book2","book4"},"李月"：{"book9"},"陈静"：{"book7","book5"}}中。完成以下两个要求：

(1) 统计当前图书系统中每个类别的图书总量。

(2) 统计所有尚未被借过的图书清单。

分析：

输入：图书分类和图书数量字典 library_books 以及用户借阅记录字典 borrowed_records。

输出：各类别的图书总数量 category_count，所有尚未被借过的图书清单 unborrowed_book。

算法：遍历 library_books 字典，每个类别的图书总数等于字典中书籍条目数之和。从图书总集合中减去所有被借出的图书集合。

程序：

```
library_books = {"文科": {"book1":9, "book2":0, "book3":15},
    "理科": {"book4":10, "book5": 5, "book6":0},
    "工科": {"book7":0, "book8": 7, "book9": 12}}
borrowed_records = {"王一": {"book2", "book4"},
    "李月": {"book9"},
    "陈静": {"book7", "book5"} }
category_count = {}
for k,v in library_books.items():
    category_count[k] = sum(v.values())
print(f"各个类别图书总量:{category_count}")
all_books = set()
borrowed_books = set()
for k,v in borrowed_records.items():
    borrowed_books.update(v)
for k,v in library_books.items():
    all_books.update(v)
unborrowed_book = all_books - borrowed_books
print(f"从未被借出的书为:{unborrowed_book}")
```

运行结果：

```
各个类别图书总量:{'文科': 24, '理科': 15, '工科': 19}
从未被借出的书为:{'book1', 'book8', 'book3', 'book6'}
```

【例 5-24】 某互联网公司开发了一款社交应用，收集了大量用户信息 user_data={2001:{"has_sensitive": True,"sensitive_types": ["phone_number","id_card"]},2002:{"has_sensitive": False,"sensitive_types": []},2003:{"has_sensitive": True,"sensitive_types": ["email","address"]},2004:{"has_sensitive": False,"sensitive_types": []}}，包括用户 ID、是否含有敏感信息(has_sensitive)以及具体的敏感数据类型(sensitive_types，如电话号码、身份证号、电子邮件、地址等)。根据《个人信息保护法》的要求，公司需对用户数据进行合规性检测，识别含有敏感信息的用户，并输出违规详情，以便依法整改。设计并实现一个用户隐私合规性检测的程序，要求如下：

(1) 输入一组用户数据(以字典形式表示)，检测其中是否含有敏感信息。

(2) 输出违规用户数量、违规用户 ID 集合及每个违规用户的具体敏感数据类型。

(3) 返回检测结果，便于后续分析或处理。

分析：

输入：一个字典 user_data，键为用户 ID(整数)，值为一个嵌套字典，包含是否含敏感信息(布尔值)及具体的敏感数据类型(字符串列表)。

输出：违规用户数量、violation_users 集合内容(违规用户 ID)、violation_details 字典内容(具体敏感数据类型)。

算法：通过 for 循环遍历所有用户。使用 if 判断是否存在敏感信息。分别统计违规用

户数量、违规用户 ID 集合和违规用户详情字典。

程序：

```
user_data = {2001: {"has_sensitive": True, "sensitive_types": ["phone_number", "id_card"]},
2002: {"has_sensitive": False, "sensitive_types": []}, 2003: {"has_sensitive": True,
"sensitive_types": ["email", "address"]},2004: {"has_sensitive": False, "sensitive_types": []}}
violation_count = 0                    #违规用户数量
violation_users = set()                #违规用户 ID 集合
violation_details = {}                 #违规用户详情字典
for user_id in user_data:
    #检查当前用户是否含有敏感信息
    if user_data[user_id]["has_sensitive"] == True:
        #统计违规用户数量
        violation_count = violation_count + 1
        #添加违规用户 ID 到集合
        violation_users.add(user_id)
        #记录违规用户的敏感数据类型
        violation_details[user_id] = user_data[user_id]["sensitive_types"]
if violation_count > 0:
    print("违规用户数量:", violation_count)
    print("违规用户 ID:", violation_users)
    print("违规详情:")
    for user_id in violation_details:
        print(f"用户 {user_id}: 含敏感信息类型 - {violation_details[user_id]}")
    print("请依法处理违规用户数据,避免隐私泄露。")
else:
    print("未发现违规用户,数据合规")
```

运行结果：

```
违规用户数量: 2
违规用户 ID: {2001, 2003}
违规详情:
用户 2001: 含敏感信息类型 - ['phone_number', 'id_card']
用户 2003: 含敏感信息类型 - ['email', 'address']
请依法处理违规用户数据,避免隐私泄露。
```

5.4 本章小结

本章深入研究了两种至关重要的数据结构：字典和集合。

1. 字典是一种特殊的数据类型，它通过键值对的形式来存储数据。这种结构允许用户通过唯一的键来快速检索、插入和删除数据项。由于每个键都是唯一的，字典非常适合用来管理那些需要通过唯一标识符来访问的数据项，例如数据库记录的索引。

2. 集合是无序的数据元素构成的，它仅包含不重复的元素。集合的一个显著特点是它能够提供非常高效的成员资格测试，这意味着可以迅速判断某个元素是否已经存在于集合中。此外，集合还具有自动去除重复元素的功能，这使得它在处理需要去重的数据集时非常有用。

通过本章的学习，深入理解字典和集合的基本概念和操作，掌握如何利用这两种数据结构来解决实际问题。

5.5 练习与思考

习题答案

一、选择题

1. 以下_____方法可用于获取字典中所有键的列表。

A. values()　　B. items()　　C. keys()　　D. get()

2. 字典的键必须是_____。

A. 可变的　　B. 不可变的　　C. 字符串类型　　D. 整数类型

3. 字典的 pop()方法的功能是_____。

A. 删除指定键并返回其值　　B. 删除所有键值对

C. 返回所有键　　D. 添加键值对

4. 在字典中尝试获取不存在的键时,get()方法默认返回_____。

A. 0　　B. None　　C. KeyError　　D. False

5. 以下_____是字典的正确表示。

A. {1,2,3}　　B. [1,2,3]　　C. {'a': 1,'b': 2}　　D. (1,2,3)

6. 以下_____是集合的正确表示。

A. s={{1: 2},4,5}　　B. s=set({'a': 1,'b': 2})

C. s={[1,2,3],5}　　D. s=set(12345)

7. 以下_____操作能够向集合中添加元素。

A. append()　　B. extend()　　C. insert()　　D. add()

8. 执行代码 d={1: 'a',2: 'b'}; d[0]='c' 后,字典 d 的值变为_____。

A. {1: 'c',2: 'b'}　　B. { 0: 'c',1: 'a',2: 'b'}

C. {1: 'a',2: 'b',0: 'c'}　　D. 报错

9. 以下_____表达式能够准确判断集合 s 是否为空。

A. s==　　B. len(s)==0　　C. s.is_empty()　　D. s==None

10. 执行 s={1,2,2,3}后,集合 s 的值是_____。

A. {1,2,2,3}　　B. {1,3,2,2}　　C. {1,2,3}　　D. 报错

11. 字典 d={1: [1,2]},执行 d[1].append(3)后,字典 d 的值是_____。

A. {1: [1,2],3}　　B. {1: [1,2,3]}　　C. {1: 3,1: [1,2]}　　D. {1: 3}

12. 下列_____操作可以清空字典。

A. dict.clear()　　B. dict.pop()

C. del dict[key]　　D. dict.remove(key)

13. 集合中的元素是_____。

A. 有序的　　B. 可变的　　C. 唯一的　　D. 重复的

14. 以下不能够创建字典的语句是_____。

A. d=dict(a=3,b=4)　　B. d=dict((['e',5],['g',8]))

C. d=dict(zip(['d','t'],[4,7]))　　D. d=dict(('v',5),('m',9))

15. 已知集合 a={1,2,3},b={3,4,5},c={5,6,7},那么 a-b-c 的值是_____。

A. {1,2}　　B. {3}　　C. {5}　　D. set()

二、填空题

1. 集合 s={1,2,2,3},执行 s.remove(2) 后,s 的值是_____。

2. 字典 d={'x': 10},执行 d.update({'y': 20})后,d 的值是_____。

3. 已知 t={'a': 1,'b': 2,'c': 3},min(t.keys())的结果是_____。

4. ls=[('a',3),('b',5)],字典生成式 {k: v * 2 for k,v in ls} 的结果是_____。

5. d={'ab': 23,'de': 12},那么'a' in d 的结果是_____。

6. s={'a',2,(1,3)},执行 s.add((2,3))后,s 的值是_____。

7. 若 d={'a': 1,'b': 2,'c': 3},则当 ss=dict(sorted(d.items(),key=lambda x: x[1],reverse=True))时,ss 的值是_____。

8. 从集合 s 中删除'a'元素,不会引发任何异常的语句是_____。

三、判断题

1. 已知 d={[1,2]: 34},那么 d 是一个字典。(　　)

2. 集合中的元素不可以是列表。(　　)

3. clear()方法会删除集合中的所有元素。(　　)

4. 已知 d={1: 2,2: 4,1: 9},那么 d[1]的结果是 2。(　　)

5. 已知 s={},那么 s 是一个空集合。(　　)

6. 如果 s={2,3,4},那么执行 s.update('abc')后,s 的结果是{2,3,4,'abc'}。(　　)

7. 若 ss=set(1234),则 ss 的结果是{1,2,3,4}。(　　)

8. s1={1,2,3}和 s2={3,2,1}是相等的集合。(　　)

9. 在 Python 3.7 以上的版本中,dit.popitem()可以删除字典 dit 中任意一个键值对。(　　)

10. 若 a={'a': 3,'b': 6},b={'a': 0},则 dict(a, ** b)可合并两个字典,当键重复时,保留 a 的值。(　　)

四、程序阅读和填空

1. 阅读程序,写出运行结果_____。

```
d = {'a': 1, 'b': 2, 'c': 3}
s = sorted(d)
print(s)
```

2. 阅读程序,写出运行结果_____。

```
wr = ['apple', 'pear', 'cherry']
le = {i: len(i) for i in wr if len(i) % 2 == 0}
print(max(le.values()))
```

3. 阅读程序,写出运行结果_____。

```
text = "hello world"
count = {}
for char in text:
    if char in count:
        count[char] += 1
    else:
        count[char] = 1
print(sum(count.values()))
```

4. 阅读程序,写出运行结果_____。

```
data = {'a': [12,3,5], 'b': [4, 5], 'c': [-3,6]}
re = {}
for k,v in data.items():
```

```
    re[k] = min(v)
ls = sorted(re.items(),key = lambda x:x[1])
print(ls[1])
```

5. 阅读程序,写出运行结果_____。

```
d = {'m': {'x': 11, 'y': 2}, 'n': {'x': 3, 'y': 14}}
r = {}
for k, v in d.items():
    r[k] = v['x'] % v['y']
print(r)
```

6. 阅读程序,写出运行结果_____。

```
s = "Dream big, work hard, stay focused"
s.replace(',','')
w = s.split()
w_len = {}
for i in w:
    w_len[i] = len(i)
ls = sorted(w_len.items(),key = lambda x:x[1],reverse = True)
print(ls[:2])
```

7. 以下程序能够依据字典中值的升序排列结果,对字典的键进行排序,并输出排序后键的列表。请完善程序。

```
d = {'a': 5, 'b': 3, 'c': 7, 'd': 1}
ls = []
v = list(d.values())
k = list(d.keys())
while _____ > 0:
    min_v = min(v)
    inx = v._____
    ls.append(_____)
    v.pop(inx)
    k.pop(inx)
print(ls)
```

8. 以下程序可以利用集合去除列表中的重复元素并保持原有顺序输出不重复的元素。请完善程序。

```
lst = [1, 2, 3, 2, 1, 4, 3]
lst_diff = _____
s = set()
for i in lst:
    if i not in _____:
        lst_diff.append(i)
        s._____
print(lst_diff)
```

9. 以下程序可以从字典中删除值小于 10 的键值对,请完善程序。

```
dic = {'a': 5, 'b': 15, 'c': 8, 'd': 20}
k_del = []
for k, v in _____:
    if v < 10:
        k_del._____
for k in k_del:
    dic._____
print(dic)
```

10. 以下程序通过遍历列表中的字典,获取每个字典的键值对。对相同键的值进行累加操作,最后输出经过整合后的一个字典,请完善程序。

```
ls = [{'a': 1, 'b': 2}, {'b': 3, 'c': 4}, {'a': 5, 'c': 7}]
result = {}
for i in ls:
    for k,v in i.items():
        if k not in ________:
            result[k] = v
        else:
            result[k] = ________
print(result)
```

五、编程题

1. 字典存储的学生的兴趣标签 interests＝{"张三"：{"编程","音乐"},"李四"：{"运动","音乐"},"王五"：{"编程","阅读"},"赵六"：{"运动","编程"}},编写程序统计每个兴趣标签的人数,并输出最受欢迎的兴趣标签。

2. 字典存储的 AI 模型的训练数据集 dataset＝{"数据 1"："公平公正","数据 2"："性别歧视","数据 3"："种族平等","数据 4"："种族歧视"},编写程序检查数据集中是否存在集合 sensitive_keywords＝{"性别歧视","种族歧视"}中的敏感内容,并输出检查结果。

3. 某高校实行智慧门禁管理,不同年级学生可通行不同教学楼。现有如下权限数据字典 access＝{ "大一"：{"A 楼","B 楼"},"大二"：{"A 楼","B 楼","C 楼"},"大三"：{"A 楼","B 楼","C 楼","D 楼"},"大四"：{"A 楼","B 楼","D 楼"}}。编程要求：

(1) 输入一个年级字符串和一个教学楼名称。

(2) 判断是否有权限,输出“允许通行”或“禁止通行”。

4. 某高校开设人工智能选修课程,记录了学生的选课情况。每个学生选择的课程记录如下：students＝{ "张三"：{"机器学习","数据挖掘"},"李四"：{"数据挖掘","机器学习"},"王五"：{"自然语言处理","机器学习"},"赵六"：{"机器学习","数据挖掘","计算机视觉"}}。编程要求：

(1) 统计每门课程的选修人数。

(2) 找出所有学生共同选修的课程(使用集合求交集)。

5. 某高等教育机构希望建立一种更为安全的数据结构以存储学生资料。目前,学生信息列表包括学号、姓名及所属学院,具体如下所示：students＝[("2023002","张三","计算机学院"),("2023001","李四","文学院"),("2023004","王五","理学院"),("2023003","林六","农学院")]。编程要求：

(1) 请将上述列表按照学号进行升序排序,并展示排序后的结果。

(2) 基于排序后的列表,构建一个字典 studentDict,其中学号作为键,对应的值为一个嵌套字典,包含学生的姓名和学院信息。利用字典的 items()方法遍历 studentDict 字典,并输出其键和值。

6. 交警部门采集到一批车辆车牌信息 plates＝["京 A12345","沪 B23456","粤 C34567","京 D45678","苏 E12341","沪 E56789","京 A06782","苏 A12341"],为了了解不同省份车辆数量分布,编写程序按车牌首字符分类统计,并找出车辆最多的省份代码。

7. 随着智慧交通系统的发展,为实现道路资源的高效配置与城市通行效率的提升,某地交通局正在采集城市主要路段的交通路径信息,并建立路径权重模型。每条路径包含起点、终点、路径距离(单位：千米)和平均通行时间(单位：分钟)。路径信息 routes＝[{"起点"："A","终点"："B","距离"：10,"时间"：15},{"起点"："B","终点"："C","距离"：8,"时

间"：10},{"起点"："A","终点"："C","距离"：15,"时间"：20},{"起点"："C","终点"："D","距离"：12,"时间"：18}]。编程要求：

(1) 构建路径权重字典 routeWeightDict,按照 ("起点","终点") 作为键,{"权重"：距离 + 时间} 作为值,以便后续路径调度使用。例如：routeWeightDict={("A","B")：{"权重"：25},...}。

(2) 统计路径总数,输出当前共统计了多少条有效路径。

(3) 遍历字典找出权重最大的路径信息,并输出其起点、终点及权重值。

8. 学校计划采购网络安全设备,不同供应商的报价 devices={"防火墙"：[12500,13800,14200],"入侵检测系统"：[8500,9200,7800],"VPN 设备"：[6800,7200,7500],"日志分析系统"：[10500,9800,11200],"终端安全管理"：[4500,5200,4800],"网络审计设备"：[9500,8900,10200]}。编程要求：

(1) 计算每种设备的平均价格,保留小数点后 1 位数字,将平均价格追加到字典相应的值中,输出修改后的字典。

(2) 将字典按照平均价格升序排列后输出。

9. 学校对智慧教室中不同设备的周能耗数据(单位：度)进行了记录：ec={"智能黑板"：[12.5,13.2,11.8,12.7,12.0],"投影仪"：[8.5,9.2,8.8,9.0,8.7],"空调系统"：[35.2,33.8,36.5,34.2,35.0],"照明系统"：[15.8,16.5,15.2,16.0,15.5],"计算机终端"：[22.3,23.5,21.8,22.7,22.0],"网络设备"：[5.2,5.5,5.0,5.3,5.1]}。请编写程序,找出能耗最高且波动最大(标准差最大)的设备。注：标准差是衡量数据离散程度的一个统计量,其计算公式为 $\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}}$,其中,$n$ 代表数据点的数量,μ 代表数据的平均值。

10. 某高校开展网络安全与金融普法教育,各院系组织了班级学习和线下活动,各院系开设的普法班级 dic_Class={ "计算机学院"：["AI 安全班","区块链班"],"金融学院"：["反诈宣传班","金融合规班"],"法学院"：["数据保护班","网络法规班"]},各班级普法知识测试平均分 dic_Score={ "AI 安全班"：85,"区块链班"：78,"反诈宣传班"：92,"金融合规班"：88,"数据保护班"：95,"网络法规班"：90},各院系线下普法活动参与人数 dic_Event={"计算机学院"：120,"金融学院"：150,"法学院"：80}。编程要求：

(1) 计算各院系普法教育综合得分,计算结果存入字典 dic_Total,格式如 {"计算机学院"：98.5,"金融学院"：105.2,...},计算公式如下：

班级得分=该院系所有班级的平均分之和×0.6

活动得分=该院系线下活动参与人数×0.4

综合得分=班级得分+活动得分

(2) 找出普法教育综合得分最高的院系。

在线作业