程序设计（Python）

元组、字典和集合

主讲：数据与目标工程学院 胡瑞娟 副教授

知识回顾

【案例】无人机侦察数据管理。在现代化军事行动中，无人机侦察系统负责收集敌方区域的各种情报数据。无人机每次执行侦察任务后，会传回大量不同类型的数据至指挥中心，包括**目标坐标位置、敌方装备数量、地形海拔高度、侦察时间戳、目标类型标识**等；指挥中心需要对这些数据进行存储、管理和分析，为指挥决策提供支持。

【任务】编写程序来**管理**这些数据，并实现以下功能：

1.**存储**侦察到的目标数据
2.**添加**新的侦察数据
3.**修改**目标数据
4.**删除**已确认处理的目标数据
5.**统计**不同类型目标的数量
6.**找出**某个区域的目标
7.对目标按坐标进行**排序**

组合数据类型
——列表

知识回顾

- Python数据类型分为：
- **基本数据类型**：
  - 数字类型（整数（int）、浮点数（float））
  - 布尔类型（bool）
  - 字符串类型(str)
- **组合数据类型**：
  - 集合、元组、列表、字典

类型	示例
数字	1234, 3.14, 3+4j
字符串	'swfu', "I'm student", '"Python "'
布尔型	True, False
集合	set('abc'), {'a', 'b', 'c'}
元组	(2, -5, 6, '3')
列表	[1, 2, 3, '3']
字典	{1:'food' ,2:'taste', '3':'import'}
文件	f=open('data.dat', 'r')
空类型	None
编程单元类型	函数、模块、类

案例引入

在之前的无人机侦察案例中，我们使用**多个列表**分别存储不同类型的数据。

```
# 创建侦察数据列表
target_coordinates = [[120, 35], [125, 38], [118, 32]]  # 目标坐标(x,y)
enemy_equipment = [5, 3, 8, 2, 6]  # 各区域敌方装备数量
terrain_elevation = [5039, 5127, 5238, 5259, 5299]  # 地形海拔高度(米)
recon_time = ['08:30', '10:15', '13:45', '16:20']  # 侦察时间
target_types = ['radar', 'missile', 'tank', 'barracks']  # 目标类型
```

【问题1】目标类型应该是**固定**的几种，不应该被随意修改！

元组

```
# 使用元组定义固定目标类型
target_types_tuple=('radar','missile','tank','barracks')
```

01

元组

元组

- 元组是一种特殊的序列类型，一旦创建不能修改，使用逗号和圆括号来表示。
- 元组的一般使用与列表类似

```
>>> bTuple =([Monday',1],2,3)
>>> bTuple
(['Monday',1], 2,3)
>>>
1
>>> len(bTuple)
>>>
(2,3)
```

```
>>>2024
2024
>>> 2014,
(2014,)
```

1. 元组

- 单元素元组：定义只包含一个元素的元组时，必须在元素后面加逗号","，否则Python会将其识别为其他数据类型。

```
single_tuple = (5,)  # 正确定义单元素元组
not_a_tuple = (5)    # 这不是元组，而是一个数字
```

访问元组元素

```
# 定义固定目标类型（使用元组）
target_types_tuple = ('radar', 'missile', 'tank', 'barracks')
print("可用目标类型:", target_types_tuple)

# 访问元组元素
print("第一个目标类型:", )
print("最后一个目标类型:", )

# 元组切片
print("前两个目标类型:", )
```

- ✓ 元组和列表一样，可以使用下标索引来访问元组中的值

修改元组元素

- ✓ 列表元素可以改变
- ✓ 元组元素不可以改变

```
# 列表可以修改
target_types_list = ['radar', 'missile', 'tank', 'barracks']
target_types_list[0] = 'radar_station'
print("修改后的列表:", target_types_list)
```

```
修改后的列表: ['radar_station', 'missile', 'tank', 'barracks']
```

```
target_types_tuple = ('radar', 'missile', 'tank', 'barracks')
target_types_tuple[0]='radar_station'
```

```
TypeError: 'tuple' object does not support item assignment
```

常用方法

- ✓ **tuple(seq)**：将列表转换为元组。
- ✓ **len(tuple)**：计算元组中元素个数。
- ✓ **max(tuple)**：返回元组中元素的最大值。
- ✓ **min(tuple)**：返回元组中元素的最小值。
- ✓ **count(x)**: 统计元组中某个元素 x 出现的次数。
- ✓ **index(x)**：返回元组中第一个值为 x 的元素的索引，如果不存在则报错。

元组中的元素是不能改变的，因此，**没有append()，insert()**这样的方法。

【示例】**将列表转为元组**，并求出元组长度、最大值、最小值。

target_types_list = ['radar', 'missile', 'tank', 'barracks']

```
target_types_tuple1 = tuple(target_types_list)
print(len(target_types_tuple1))
tu_max = max(target_types_tuple1)
tu_min = min(target_types_tuple1)
print(tu_max)
print(tu_min)
```

```
4
tank
barracks
```

- 元组的解包（Unpacking）：元组可以通过解包将其中的元素分别赋值给多个变量。

```
target_types_list = ['radar', 'missile', 'tank', 'barracks']
target_types_tuple1 = tuple(target_types_list)
a,b,c,d = target_types_tuple1
print(f'a={a},b={b},c={c},d={d}')
a,*b,c = target_types_tuple1
print(f'a={a},b={b},c={c}')
```

```
a=radar_station,b=missile,c=tank,d=barracks
a=radar_station,b=['missile', 'tank'],c=barracks
```

通过星号 * 操作符将多个元素解包为一个列表。

【思考】交换两个变量的值?

元组与列表比较

特性	元组(Tuple)	列表(List)
可变性		
数据组成		
语义含义		
性能		
数据保护		
语法		
内存占用		
适用场景		
方法数量		

案例分析

在之前的无人机侦察案例中，我们使用**多个列表**分别存储不同类型的数据。

```
# 列表案例-问题1：创建侦察数据列表
target_coordinates = [[120, 35], [125, 38], [118, 32],[122,40]]  # 目标坐标(x,y)
enemy_equipment = [5, 3, 8, 6]  # 各区域敌方装备数量
terrain_elevation = [5039, 5127, 5238, 5299]  # 地形海拔高度(米)
recon_time = ['08:30', '10:15', '13:45', '16:20']  # 侦察时间
target_types = ['radar', 'missile', 'tank', 'barracks']  # 目标类型
```

问题2：目标坐标、设备数量和目标类型有对应关系，比如目标1对应的数据是[120,35]、5、'rader'，使用列表数据分散，不易管理，如何更好地组织管理这些数据?

 # 类比：学生（学号、姓名、年龄、性别），如果学号、姓名、年龄、专业单独用列表或元组存储，管理、操作数据需要找到对应关系，效率低。

```
# 创建目标字典
target1 = {
    'id': 1,
    'coordinates': (120, 35),
    'type': 'radar',
    'equipment_count': 5,
}

target2 = {
    'id': 2,
    'coordinates': (125, 38),
    'type': 'missile',
    'equipment_count': 3,
}

print("目标1信息:", target1)
print("目标1类型:", target1['type'])
print("目标1坐标:", target1['coordinates'])
```

字典

02

字典

字典的概念

冒号都在哪里出现过，各自表示什么？

字典是映射类型，是另外一种可变组合数据类型，且可以存储任意类型对象。

- 字典的创建

使用{}，每个键值对（key=>value）用冒号（：）分割，每对之间用逗号（,）分割。

```
字典名={key1:value1,key2:value2}
```

- 示例

```
students={'name':'Tom','age':18,'sex':'男',18:19}
```

字典（dict）

- “键--值”数据项的组合，每个元素是一个键值对

  如：身份证号（键）--个人信息（值）。

- 字典类型数据**通过映射**查找数据项，具体来说就是通过任意键查找字典中的值。
- 字典类型以键为索引，一个键对应一个值。
- 字典类型的键值对是无序的

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

字典的概念

- 字典中的键必须是唯一的不能重复

为什么不能重复?

```
dict={"北京":21,"北京":22}
print(dict)
```

```
{'北京': 22}
```

- 数字和字符串可以当键，注意不能重复

```
dict={1:22,1.3:33,"nianling":22}
print(dict)
```

```
{1: 22, 1.3: 33, 'nianling': 22}
```

- 字典中的键必须是不可变类型

列表可以当键吗?

```
l=[0,1,2]
dict={l:22}
print(dict)
```

```
TypeError: unhashable type: 'list'
```

字典的操作

创建字典

使用大括号就可以创建字典，只用方括号可以添加新的元素。

```
students={'name':'tom','age':18,'sex‘:’男'}
print(students)

st2={}
st2['name']=‘wang' #字典名[键名]=值
```

建一个存储全班学生信息的变量，可以快速地通过学号查找该学号学生的信息 如何实现?

建一个存储全班学生信息的变量，可以快速地通过学号查找该学号学生的信息，如何实现?

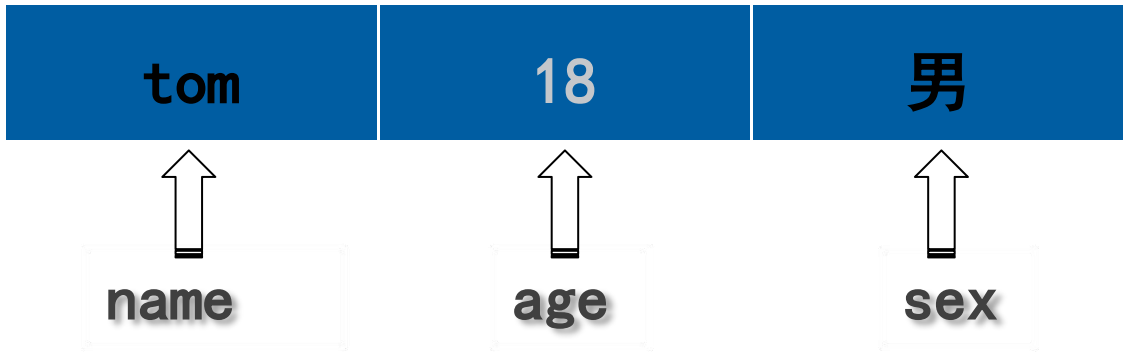
```
students_dict = {}
students_dict[1001] = {"姓名": "赵一", "年龄": 16, "性别": "女"}
students_dict[1002] = {"姓名": "钱二", "年龄": 17, "性别": "男"}
students_dict[1003] = {"姓名": "孙三", "年龄": 16, "性别": "女"}
target_id = eval(input())
if target_id in students_dict:
    print(students_dict[target_id])
else:
    print("未找到对应学号的学生信息")
```

字典的操作

访问字典

字典中根据键访问值，可以指定放在方括号内的键。

```
students={'name':'tom','age':18,'sex ':' 男'}
```



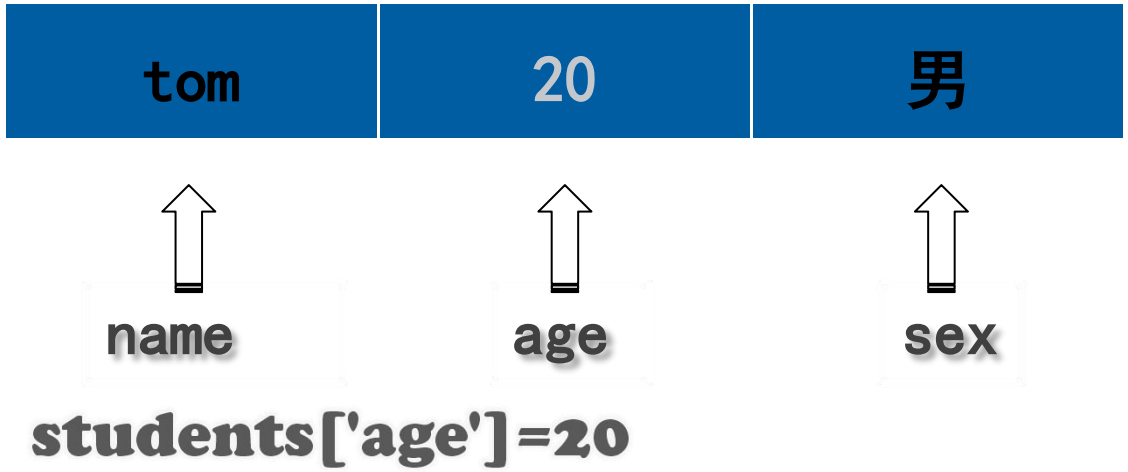
列表数据类型都是通过 数字索引 来获得值，字典是通过 键 来获得值

字典的操作

修改字典元素

字典元素也是可以修改的，通过key找到具体元素之后，给一个新的元素值即可。如：将学员的年龄修改为20。

```
students={'name':'tom','age':18,'sex':'男'}
```



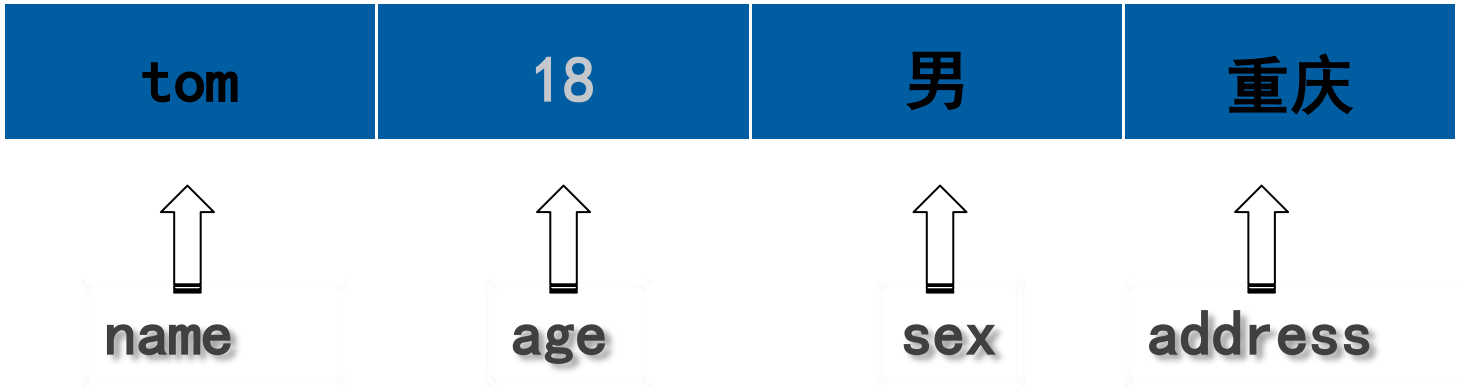

```
students['age']=20
```

字典的操作

添加字典元素

动态的向字典中添加元素的时候，只要添加的键在字典中不存在，就会新增这个元素。如：添加一个住址信息。

字典：students={'name':'tom','age':18,'sex':'男'}



students['address']='重庆'

字典的操作

删除字典元素

使用del语句删除元素：del既可以删除指定的字典元素，也可以删除整个字典，如果不指定key，代表删除整个字典。语法如下：

pop字典与pop列表的区别?

```
del 字典名[key]
```

```
字典名.pop(key)
```

del的使用注意：
准确的说是删除变量名引用

使用clear()方法清空整个字典：被清空的字典最后会剩下一个空的字典在，而用del删除的字典在程序当中就不存在了。语法如下：

```
字典名.clear()
```

字典的操作

删除字典元素

```
d ={"北京"：22，"上海"：24}
#查
print( d["北京"])  #   22
#改
d["北京"]= 20  #   {"北京”：20，"上海"：24}
#增
d["广州"]= 23  #   {"北京"：20，"上海"：24，“广州"：23}
#删
d.pop(“北京“）  #  {"上海"：24，“广州"：23}
```

字典的函数和方法

函数	描述	示例
len(dict)	计算字典中元素的个数	dict1={'name':'Tom','age':18,'sex':'男',18:19} print(len(dict1)) 输出结果：4
type(variable)	返回输入变量的数据类型，如果变量是字典就返回<class ‘dict’ >	dict1={'name':'Tom','age':18,'sex':'男',18:19} print(type(dict1)) 输出结果： <class 'dict'>

字典的函数和方法

方法	描述	示例
dict.fromkeys(seq[,value])	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	seq=('name','age','sex') dict1=dict.fromkeys(seq) print("新字典为：",dict1) dict2=dict.fromkeys(seq,'jack') print("新字典为：",dict2) #新字典为： #新字典为： 'jack'}

key in dict	如果键在字典dict里返回true,否则返回false	dict1={'name':['tom','jack'],'age':18,'sex':'男',18:19}
		if 'name' in dict1:
		print("键name在字典中存在")
		else:
		print("键name在字典中不存在")
		#键name在字典中存在

【举一反三】程序阅读

```
student_ids = [1001, 1002, 1003]
students_dict = dict.fromkeys(student_ids)
student_info_list = [
    {'姓名': '赵一', 'age': 16,'性别':"女"},
    {'姓名': '钱二', 'age': 17,"性别":"男"},
    {'姓名': '孙三', 'age': 16,"性别":"女"}]
for i in range(len(student_ids)):
    students_dict[student_ids[i]] = student_info_list[i]
target_id = eval(input())
print(students_dict[target_id])
```

字典的函数和方法

不是列表，而是 **“动态视图”**：它不存储键的副本，只是指向字典的键，字典的键发生变化时，视图会自动同步，无需重新生成，内存占用极少。

方法	描述	示例
dict.keys()	以列表返回一个字典所有的键	dict1={'name':['tom','jack'],'age':18,'sex':'男'} print(dict1.keys()) #
dict.values()	以列表返回一个字典中的所有值	dict1={'name':'tom','age':18} print(dict1.values()) #
dict.items()	以列表返回一个字典中所有的键值对	dict1={'name':'tom','age':18} print(dict1.items())

【注意】

- ✓ dict.keys()返回dict_keys而非列表，是因为它是动态视图对象，目的是节省内存、实时同步字典变化；
- ✓ dict_keys可遍历、可判断成员（如'name' in dict1.keys()），用法和列表接近；
- ✓ 若需要普通列表格式，只需用list()函数转换（list(dict1.keys())）；
- ✓ 不只是keys()，字典的values()（值视图）、items()（键值对视图）返回的都是类似的视图对象（dict_values、dict_items），逻辑完全一致。

字典的函数和方法

方法	描述	示例
dict.get(key,default=None)	返回指定键的值，如果值不在字典中返回default值	dict1={'name':'tom','age':18} print ("age键的值为 :" ,dict1.get('age', 9)) print ("sex键的值为 :" ,dict1.get('sex', '男'))
dict.setdefault(key, default=None)	和get类似，但如果键不存在于字典中，将会添加键并将值设为default的值	dict1={'name':'tom','age':18} print ("age键的值为 : " ,dict1.setdefault('age', 9)) print ("sex键的值为 : " ,dict1.setdefault('sex', '男')) print ("新字典为： ",dict1)

【举一反三】

```
>>> DNobelPrize={"物理学":"杨振宁","文学奖":"莫言"}
>>> DNobelPrize.keys()
    dict_keys(['物理学', '文学奖'])
>>> list(DNobelPrize.values())
    ['杨振宁', '莫言']
>>> DNobelPrize.items()
    dict_items([('物理学', '杨振宁'), ('文学奖', '莫言')])
>>> '伍连德' in DNobelPrize
     False
>>> DNobelPrize.get('文学奖', '川端康成')
     '莫言'
>>> DNobelPrize.setdefault("医学奖","屠呦呦")
     '屠呦呦'
>>> print(DNobelPrize)
    {'物理学': '杨振宁', '文学奖': '莫言', '医学奖': '屠呦呦'}
```

- 字典也可以通过for...in语句对其元素进行遍历，基本语法结构如下：

```
for <任意变量名> in <字典名>:
    语句块
```

```
Dcountry={"中国":"北京","美国":"华盛顿","法国":"巴黎"}
for key in Dcountry:
    print(key)
```

```
中国
美国
法国
```

```
d={"张三": 86, "李四": 92, "王五": 85}
for i in d:                    #遍历字典中所有的键 Key
   print(i)
   print(d[i])
#遍历字典中所有的键 Key
for k in d.keys():
   print(k)
#遍历字典中所有的值 Value
for v in d.values():
   print(v)
#遍历字典中所有的键和值 Key-Value
for k, v in d.items():
   print(k,v)
```

2. 字典

```
stu1 ={"name":"张三","chinese":86,"math": 92}
stu2 ={"name":"李四","chinese": 96,"math": 99}
stu3 = {"name":"王五","chinese":72,"math": 84}
d ={"小张":stu1, "小李":stu2,"小王":stu3}
print(d ["李四"]["chinese"])
```

```
96
```

```
l = [stu1, stu2, stu3]
print(l[1]["chinese"])
```

```
96
```

【举一反三】 提示用户输入一个 1 - 12 之间的整数值表示月份

然后在控制台显示用户输入的这个月份有多少天

比如用户输入的数是 7，就显示 "7 月份有 31 天"

```
m=eval(input("请输入1-12之间的月份值："))
day={1:31,2:29,3:31,4:30,5:31,6:30,7:31,8:31,9:30,10:31,11:30,12:31}
if m==2:
    n = eval(input("请输入年份："))
    print(f"{m}月份有{29 if n%4==0 and n%100!=0 or n%400==0 else 28}天")
else:
    print("{}月份有{}天".format(m, day[m]))
```

还能用什么结构？列表可以吗?

【举一反三】

```
m=eval(input("请输入1-12之间的月份值："))
days=[1,3,5,7,8,10,12]
if m==2:
    n = eval(input("请输入年份："))
    print(f"{m}月份有{29 if n%4==0 and n%100!=0 or n%400==0 else 28}天")
elif m in days:
    print("{}月份有{}天".format(m,31))
else:
    print("{}月份有{}天".format(m,30))
```

字典推导式

字典推导式的格式、用法与列表推导式类似，区别在于字典推导式外侧为大括号“{}”，且内部需包含键和值两部分。

```
{new_key:new_value for key,value in dict.items()}
```

```
old_dict = {'name':'Jack','age':23,'height':185}
new_dict = {value:key for key,value in old_dict.items()}
print(new_dict)
```

```
{'Jack': 'name',
23: 'age',
185: 'height'}
```

03

集合

例

人事部门的一份工资信息表登记时由于工作人员的疏忽有部分姓名重复登记了，如何快速解决这个问题？

```
names=[“Bob”,”Alice”,”Amy”,“Bob”,”Alice”,”Amy”]
namesSet=set(names)
print(namesSet)
```

集合

列表、字典？

- 集合中的元素不能重复
- 元素类型是不可变数据类型，例如：整数、浮点数、字符串等。
- 集合是无序组合，没有索引和位置的概念
- set（）函数用于集合的生成，返回结果是一个无重复且排序任意的集合。
- 集合通常用于表示成员间的关系、元素去重等。

集合的创建与删除

- 直接将集合赋值给变量

```
>>> a = {3, 5}
>>> a.add(7)#向集合中添加元素
>>> a
{3, 5, 7}
```

- 使用set将其他类型数据转换为集合

```
>>> a_set = set(range(8,14))
>>> a_set
{8, 9, 10, 11, 12, 13}
>>> b_set = set([0, 1, 2, 3, 0, 1,
2, 3, 7, 8])
>>> b_set
{0, 1, 2, 3, 7, 8}
>>> c_set = set()
>>> c_set
set()
>>> del b_set
>>> ###print(b_set)?
```

```
NameError: name 'b_set' is not defined
```

- 使用del删除整个集合

集合的创建与删除

- 当不再使用某个集合时，可以使用del命令删除整个集合
- 集合对象的pop()方法随机弹出并删除其中一个元素
- remove()方法直接删除指定元素
- clear()方法清空集合

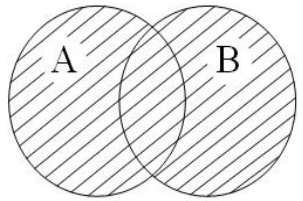
```
>>> a = {1, 4, 2, 3}
>>> a.pop()
1
>>> a.pop()
2
>>> a
{3, 4}
```

```
>>> a.add(2)
>>> a
{2, 3, 4}
>>> a.remove(3)
>>> a
{2, 4}
```

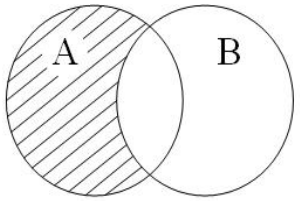
集合操作

- Python集合支持交集、并集、差集等运算

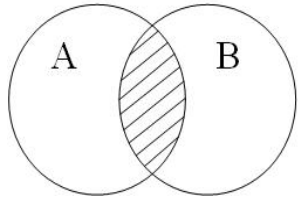
```
#并集
>>> a_set = set([8, 9, 10, 11, 12, 13])
>>> b_set = {0, 1, 2, 3, 7, 8}
>>> a_set | b_set / (a_set.union(b_set)
{0, 1, 2, 3, 7, 8, 9, 10, 11, 12, 13}
```



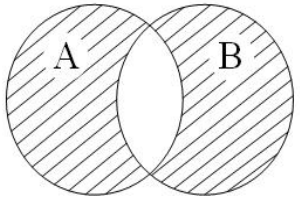
A|B



A-B



A&B



A^B

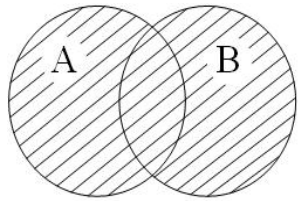
```
#交集
>>> a_set & b_set
{8}
>>> a_set.intersection(b_set)
{8}
```

```
#差集
>>> a_set.difference(b_set)
{9, 10, 11, 12, 13}
>>> a_set - b_set
{9, 10, 11, 12, 13}
```

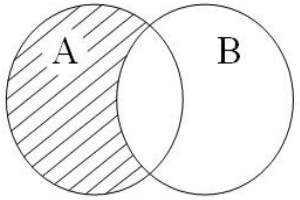
3. 集合

```
#对称差集
>>> a_set.symmetric_difference(b_set)
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
>>> a_set ^ b_set
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
```

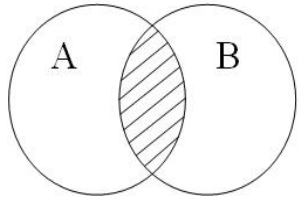
找出两个集合中 “只属于其中一个集合，不同时属于两个集合” 的所有元素



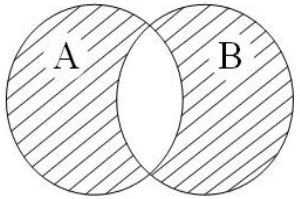
A|B



A-B



A&B



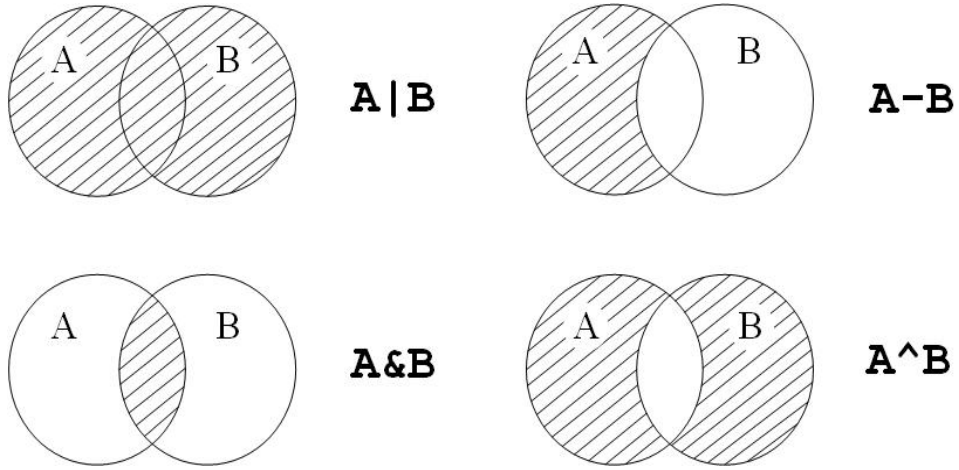
A^B

```
#测试子集
>>> x = {1, 2, 3}
>>> y = {1, 2, 5}
>>> z = {1, 2, 3, 4}
>>> x.issubset(y)
False
>>> x.issubset(z)
True
```

```
#判断交集是否为空
>>> {3} & {4}
set()
>>> {3}.isdisjoint({4})
True
```

【举一反三】

```
#包含关系测试
>>> x = {1, 2, 3}
>>> y = {1, 2, 5}
>>> z = {1, 2, 3, 4}
>>> x < y   #<运算符表示真子集关系
False
>>> x < z
#真子集
True
>>> y < z
False
>>> {1, 2, 3} <= {1, 2, 3}
#子集
True
```



习题

- 给定一个字符串s = "aabbccddeeff"，找出去除字符串中的重复字符后的字符串,去重之后要按照字母顺序排序。

```
s = "aabbccddeeff"
char_list = list(s)
char_set = set(char_list)
print(''.join(char_set))
```

''.join(list(set(char_list)).sort())对吗?

集合推导式

集合也可以利用推导式创建，集合推导式的格式与列表推导式相似，集合推导式外侧为大括号“{}”

```
{exp for x in set if cond}
```

```
nums = {1, 2, 3, 4, 5}
square_evens = {num ** 2 for num in nums if num % 2 == 0}
print(square_evens)
```

```
{16, 4}
```

总结

- 元组的有序性 → 人生需要规划与秩序
- 字典的结构性 → 做事需要条理与系统
- 集合的确定性 → 目标需要明确与专注

人生启示

- 像元组一样，在原则问题上坚定不移，在人生坐标中找准定位
- 人生如字典，每个选择都是键，每个结果都是值，键值对应方能成就精彩
- 在集体中保持个性，在合作中实现共赢，求同存异方能行稳致远

作业

【编程实现】设计一个Python程序，实现以下功能：

1. 接收用户输入的一段文本（包含多个单词）
2. 将文本中的单词按照某种方式分离并保存，以便后续处理
3. 找出文本中所有不同的单词（排除重复项）
4. 统计每个不同单词在文本中出现的次数

【提示】

思考使用合适的数据结构来分别实现以下功能：

- 存储原始文本中的所有单词（允许重复）
- 记录文本中出现过的不同单词（不允许重复）
- 存储每个不同单词与其出现次数的对应关系

作业

【手机通讯录】

在通讯录中**通过姓名查看相关联系人的联系方式等信息**，也可以在其中新增联系人，或修改、删除联系人信息。

编写程序，实现具备添加、查看、修改以及删除联系人信息功能的手机通讯录。

```
contacts = {}
while True:
    print("请选择操作：")
    print("1. 添加联系人")
    print("2. 查看联系人")
    print("3. 修改联系人信息")
    print("4. 删除联系人")
    print("5. 退出")
    choice = input("请输入选项对应的数字：")
    if choice == "1":
        name = input("请输入联系人姓名：")
        phone_number = input("请输入联系人电话号码：")
        contacts[name] = phone_number
        print(f"{name} 已成功添加到通讯录。")
.....
```

下课并不代表思考的终止
期待我们下次的思想碰撞