PYTHON程序设计

张树兵编

20230308改

励志诗（节选）

[魏晋] 张华

安心恬荡，栖志浮云。体之以质，彪之以文。
如彼南亩，力耒既勤。蘼蕤致功，必有丰殷。
水积成川，载澜载清。土积成山，歊蒸郁冥。
山不让尘，川不辞盈。勉尔含弘，以隆德声。
高以下基，洪由纤起。川广自源，成人在始。
累微以著，乃物之理。纆牵之长，实累千里。
复礼终明，天下归仁。若金受砺，若泥在钧。
进德修业，辉光日新。隰朋仰慕，予亦何人。

注释：

- biāo gǔn
- 穮蓧。耕耘和培育。薾，通'穮'。
- 耒读作lěi，耒指古代耕地用的农具，耜用于起土，耒是耜上的弯木柄，也用做农具的统称。
- 歊蒸郁冥：xiāo zhēng气升腾貌，树木郁郁葱葱。
- 山不让尘，川不辞盈。管子曰：海不辞水，故能成其大；山不辞土，故能成其高；
- 士不厌学，故能成其圣。
- 勉尔含弘，以隆德声。周易曰：含弘光大。蔡邕袁乔碑曰：于兹德声，发闻遐迩。

注释：

- 毛泽东年轻时题写的自励警句：“若金发砺，若陶在钧；进德修业，光辉日新”.
- 砺：磨石
- 钧：制陶器所用的转轮
- 就像要让金子发光就要有好的磨石，要做好陶就要有好的转轮。
- 隰朋：[xí péng]，姜姓，出身于齐国公族。著名春秋时期齐国大夫，朋氏鼻祖。

字典概念

- Python 字典是一种类似于字典书一样的数据结构。就像字典书里面有很多单词和它们的解释一样，Python 字典也包含了很多键值对，其中键是唯一的，而值可以重复。
- 比如说，我们可以创建一个 Python 字典来表示水果的价格：

```
fruit_prices = {"苹果": 5, "香蕉": 3, "橙子": 4}
```

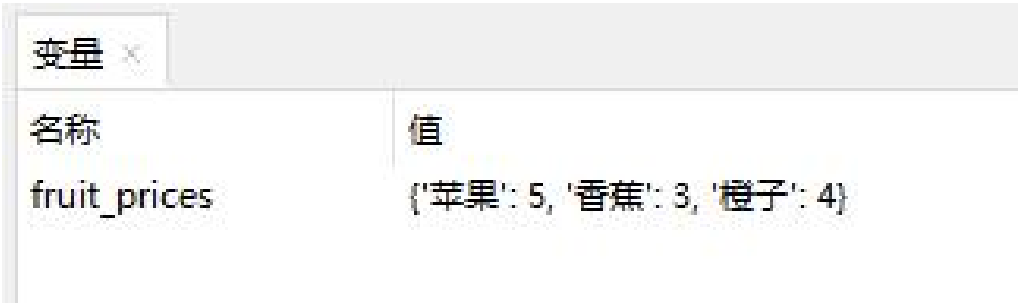
- 在这个字典里面，键是水果的名字，而值是它们的价格。如果我们想查找苹果的价格，只需要使用它的键：

```
print(fruit_prices["苹果"])
```

- 这个程序会输出 5，因为苹果的价格是 5 元。
- 总之，Python 字典就是一个用来存储键值对的工具，它可以帮助我们快速地查找和修改数据。

```
fruit_prices = {"苹果": 5, "香蕉": 3, "橙子": 4}
print(fruit_prices["苹果"])
```

```
>>> %Run '8_1字典.py'
5
```



字典概念

- Python 字典是一种数据结构，它可以用来存储和访问键-值对。就像是一个字典一样，你可以通过查找字典中的关键词来得到相应的定义。例如，如果你想知道“苹果”是什么，你可以查找字典中的“苹果”这个关键词并找到它的定义，也就是它的值。
- 这里有一个简单的例子，让我们创建一个水果字典，其中包含三种不同的水果及其价格：

```
fruits = {"apple": 2, "banana": 1, "orange": 3}
```

- 在这个例子中，我们使用大括号 `{}` 创建了一个字典，其中包含了三个键-值对。每个键和值之间用冒号 `:` 分隔开来，每个键-值对之间用逗号 `,` 分隔开来。 **(英文符号)**
- 现在，如果我们想知道苹果的价格，我们可以使用以下代码：

```
print(fruits["apple"])
```

- 这将输出 `2`，因为我们查找了键为 `"apple"` 的值。这就是 Python 字典的基本用法。

字典遍历 for

- Python 字典是一种可以存储键值对的数据类型。我们可以把它想象成一
- 本字典，里面有很多词语和对应的解释。比如说，我们可以创建一个字
- 典来存储英文单词和它们的中文翻译：

```
my_dict = {"hello": "你好", "world": "世界", "python": "蟒蛇"}
```

- 在这个字典中，"hello"、"world" 和 "python" 是键，而它们的值分别是 "
- 你好"、"世界" 和 "蟒蛇"。我们可以通过键来访问这些值，比如说：

```
print(my_dict["hello"])
```

- 我们也可以向字典中添加新的键值对，比如说：

```
my_dict["cauc"] = "中国民航大学"
```

#这段代码会依次输出字典中的每一个键的值，比如：

```
for key in my_dict:
    print(my_dict[key])
```

```
>>> %Run '8_5字典.py'
你好
世界
蟒蛇
中国民航大学
```

名称	值
key	'cauc'
my_dict	{'hello': '你好', 'world': '世界', 'python': '蟒蛇', 'cauc': '中国民航大学'}

字典遍历 使用items()方法

- Python 字典是一种可以存储键值对的数据类型。我们可以把它想象成一
- 本字典，里面有很多词语和对应的解释。比如说，我们可以创建一个字
- 典来存储英文单词和它们的中文翻译：
- 在这个字典中，"hello"、"world" 和 "python" 是键，而它们的值分别是 "
- 你好"、"世界" 和 "蟒蛇"。我们可以通过键来访问这些值，比如说：

```
print(my_dict["hello"])
```

- 我们也可以向字典中添加新的键值对，比如说：

```
my_dict["cauc"] = "中国民航大学"
```

这段代码会依次输出字典中的每一个键值对，比如：

```
for key, value in my_dict.items():
    print(key, value)
```

```
>>> %Run '8_2字典.py'
你好
hello 你好
world 世界
python 蟒蛇
cauc 中国民航大学
```

字典遍历 使用items()方法

```python
my_dict = {"hello": "你好", "world": "世界", "python": "蟒蛇"}
#通过键来访问这些值，
print(my_dict["hello"])  # 输出："你好"
#向字典中添加新的键值对
my_dict["cauc"] = "中国民航大学"
#这段代码会依次输出字典中的每一个键值对
for key, value in my_dict.items():
    print(key, value)
```

名称	值
key	'cauc'
my_dict	{'hello': '你好', 'world': '世界', 'python': '蟒蛇', 'cauc': '中国民航大学'}
value	'中国民航大学'

助手

The code in 8_2字典.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusin

Shell

```
>>> %Run '8_2字典.py'
你好
hello 你好
world 世界
python 蟒蛇
cauc 中国民航大学
```

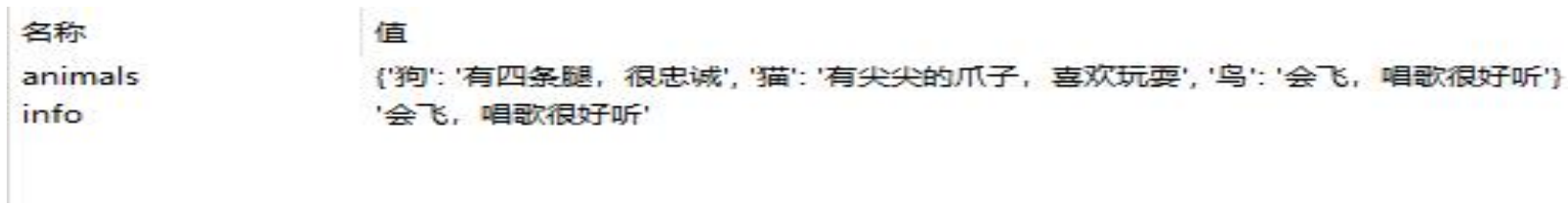
字典遍历 使用keys()方法

- Python的字典是一种非常有用的数据类型，就像一个字典一样，可以帮助我们查找和存储各种信息。字典非常方便，可以帮助我们存储和访问各种信息。
- 例如，我们可以创建一个字典来存储动物的信息：

```
animals = {"狗": "有四条腿，很忠诚", "猫": "有尖尖的爪子，喜欢玩耍", "鸟": "会飞，唱歌很好听"}
```

- 在这个字典中，我们可以通过键（例如“狗”、“猫”、“鸟”）来查找对应的值.
- 如果我们只需要遍历字典的键，可以使用 keys() 方法：

```
for animal in animals.keys():
    print(animal)
```




```
>>> %Run '8_3字典.py'
狗
猫
鸟

>>>
```

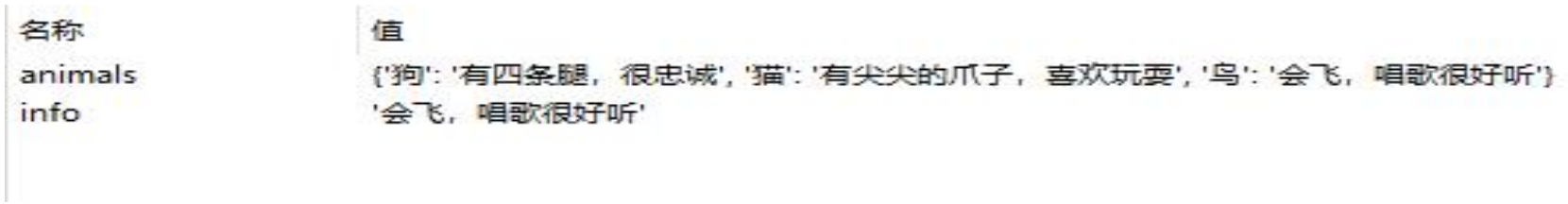
字典遍历 使用values()方法

- Python的字典是一种非常有用的数据类型，就像一个字典一样，可以帮助我们查找和存储各种信息。字典非常方便，可以帮助我们存储和访问各种信息。
- 例如，我们可以创建一个字典来存储动物的信息：

```
animals = {"狗": "有四条腿，很忠诚", "猫": "有尖尖的爪子，喜欢玩耍", "鸟": "会飞，唱歌很好听"}
```

- 在这个字典中，我们可以通过键（例如“狗”、“猫”、“鸟”）来查找对应的值.
- 如果我们只需要遍历字典的值，可以使用 values() 方法：

```
for info in animals.values():
    print(info)
```




```
>>> %Run '8_4字典.py'
有四条腿，很忠诚
有尖尖的爪子，喜欢玩耍
会飞，唱歌很好听
>>>
```

字典应用：成绩统计1

- Python中的“字典”是一个非常有用的工具，可以用来存储各种信息。例如，我们可以用字典来记录学生的成绩。
- 让我们假设有三个学生，它们的姓名和成绩如下：- 小明：90分- 小红：80分- 小刚：95分

我们可以用一个字典来记录他们的成绩，代码如下：

```
scores = {"小明": 90, "小红": 80, "小文": 95}
```

这个字典中，每个键（key）代表一个学生的姓名，每个值（value）代表他们的成绩。现在我们可以很容易地查找每个学生的成绩，只需要输入他们的姓名即可：

```
print(scores["小明"])  # 输出90
```

我们还可以用循环来遍历整个字典，计算所有学生的平均成绩：

```
total_score = 0
for name, score in scores.items():
    total_score += score
average_score = total_score / len(scores)
print("平均分 {:.2f}".format(average_score))  # 输出88.33
```

这段代码中，我们用`items()`方法获取字典中的所有键值对（key-value pairs），然后用循环逐一计算每个学生的成绩总和。最后，我们除以学生人数得到平均成绩。

字典应用：成绩统计1

```python
scores = {"小明": 90, "小红": 80, "小文": 95}
total_score = 0
for name, score in scores.items():
    total_score += score
average_score = total_score / len(scores)
print("平均分 {:.2f}".format(average_score))  # 输出88.33
```

名称	值
average_score	88.33333333333333
name	'小文'
score	95
scores	{'小明': 90, '小红': 80, '小文': 95}
total_score	265

助手

Shell

```
>>> %Run -c $EDITOR_CONTENT
 平均分 88.33
```

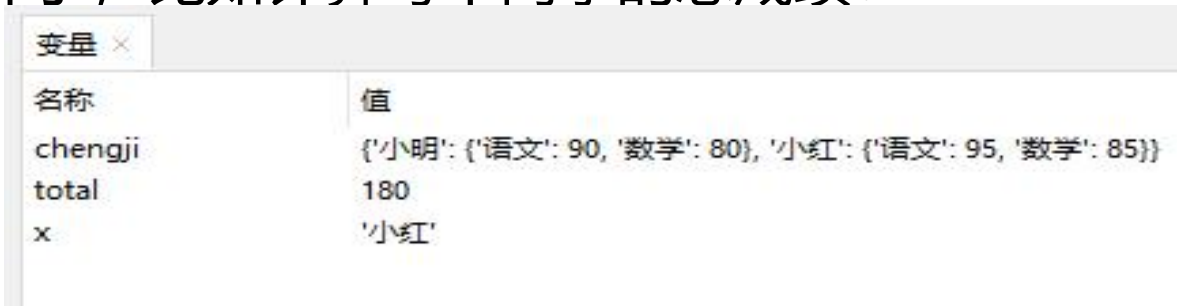
字典应用：成绩统计2“典中典”

- 如果我们有很多个同学的成绩，我们就可以用一个字典来存储所有同学的成绩：

```
chengji = {'小明': {'语文': 90, '数学': 80}, '小红': {'语文': 95, '数学': 85}}
```

- 这个字典中，'小明'和'小红'就是键，它们对应的值又是一个字典，表示他们各自的成绩。
- 有了这个字典，我们就可以方便地进行成绩统计了，比如计算每个同学的总成绩：

```
for x in chengji:
    total = chengji[x]['语文'] + chengji[x]['数学']
    print(x + '的总成绩是：' + str(total))
```



- 这段代码中，'for x in chengji'表示遍历字典中的每个键，也就是每个同学的名字。然后，我们用chengji[x]['语文']和chengji[x]['数学']分别取出这个同学的语文成绩和数学成绩，再把它们加起来得到总成绩。最后，用print函数输出结果。
- 这样，我们就成功地用Python的字典来统计同学们的成绩

字典应用：成绩统计2 “典中典”

7_9.py | 8_1字典.py | 8_2字典.py | 8_3字典.py | 8_4字典.py | 8_5字典.py * | 8_6字典chengji.py | 8_6字典chengji2.py

```
1 chengji = {'小明': {'语文': 90, '数学': 80}, '小红': {'语文': 95, '数学': 85}}
2 for x in chengji:
3     total = chengji[x]['语文'] + chengji[x]['数学']
4     print(x + '的总成绩是: ' + str(total))
```

变量

名称	值
chengji	{'小明': {'语文': 90, '数学': 80}, '小红': {'语文': 95, '
total	180
x	'小红'

Shell

```
>>> %Run '8_6字典chengji2.py'
小明的总成绩是：170
小红的总成绩是：180
>>>
```

助手

The code in 8_6字典chengji2.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

字典总结

理解“映射”

- 映射是一种键(索引)和值(数据)的对应

内部颜色：蓝色

外部颜色：红色

```
"streetAddr"  :  "中关村南大街5号"
"city"        :  "北京市"
"zipcode"     :  "100081"
```

字典总结

字典类型定义

字典类型是“映射”的体现

- **键值对： 键是数据索引的扩展**
- **字典是键值对的集合， 键值对之间无序**
- **采用大括号{}和dict()创建， 键值对用冒号: 表示**

{<键1>:<值1>， <键2>:<值2>， … ， <键n>:<值n>}

字典总结

字典类型的用法

在字典变量中，通过键获得值

<字典变量> = {<键1>:<值1>, … , <键n>:<值n>}

<值> = <字典变量>[<键>]　　<字典变量>[<键>] = <值>

[ ] 用来向字典变量中索引或增加元素

字典总结

字典类型定义和使用

```
>>> d = {"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>> d
{'中国': '北京', '美国': '华盛顿', '法国': '巴黎'}
>>> d["中国"]
'北京'
>>> de = {} ; type(de)
<class 'dict'>
```

type(x)

返回变量x的类型

字典总结

字典类型操作函数和方法

函数或方法	描述
del d[k]	删除字典d中键k对应的数据值
k in d	判断键k是否在字典d中，如果在返回True，否则False
d.keys()	返回字典d中所有的键信息
d.values()	返回字典d中所有的值信息
d.items()	返回字典d中所有的键值对信息

字典总结

字典类型操作

```
>>> d = {"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>> "中国" in d
True
>>> d.keys()
dict_keys(['中国', '美国', '法国'])
>>> d.values()
dict_values(['北京', '华盛顿', '巴黎'])
```

字典总结

字典类型操作函数和方法

函数或方法	描述
d.get(k, <default>)	键k存在，则返回相应值，不在则返回<default>值
d.pop(k, <default>)	键k存在，则取出相应值，不在则返回<default>值
d.popitem()	随机从字典d中取出一个键值对，以元组形式返回
d.clear()	删除所有的键值对
len(d)	返回字典d中元素的个数

字典总结

字典类型操作

```
>>> d = {"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>> d.get("中国","伊斯兰堡")
'北京'
>>> d.get("巴基斯坦","伊斯兰堡")
'伊斯兰堡'
>>> d.popitem()
('美国', '华盛顿')
```

字典 get用法

- Python中的字典是一种存储键值对的数据结构。其中，get()是一个用来获取字典中指定键对应的值的函数。

例如，我们可以创建一个字典，其中包含学生名字和对应的数学成绩：

```
scores = {'小明': 90, '小红': 95, '小文: 80}
```

- 然后，我们可以使用get()函数来获取小明的数学成绩：`

```
x = scores.get('小明')
print('小明',x)
```

```
>>> %Run 8_7get.py
小明 90
小李 99
>>>
```

这将返回90，因为小明对应的数学成绩是90。如果我们尝试获取一个不存在的键，例如：

```
y = scores.get('小李',99)
print('小李',y)
```

这将返回99，因为字典中没有名为“小李”的键,将返回备用值 99。

所以，get()函数的作用就是获取字典中指定键对应的值，如果键不存在，则返回None。

字典 get用法

```
1  scores = {'小明': 90, '小红': 95,'小文': 80}
2  x = scores.get('小明')
3  print('小明',x)
4  y = scores.get('小李',99)
5  print('小李',y)
```

Shell

```
>>> %Run 8_7get.py
 小明 90
 小李 99
>>>
```

set集合概念：

- Python中的set是一种数据类型，它类似于一个包含不同元素的列表。唯一的区别是，set不允许重复元素。例如，如果我们有一个set包含三个数字{1, 2, 3}，我们不能将另一个数字1添加到集合中，因为集合中已经有一个1。这是set的一个重要特性，因为它可以帮助我们避免重复的数据。
- 例如，如果我们想要创建一个set包含一些水果，我们可以这样做：

fruits = {"苹果", "梨", "香蕉","苹果"}

这将创建一个包含三个水果的set。我们可以访问这些元素，就像在列表中一样：

print(fruits)

- 我们还可以使用set的一些特殊函数来执行各种操作，例如添加新元素、删除元素、合并两个set等等。这些函数使set成为一种非常有用的数据类型

```
>>> %Run 8_8set.py
{'苹果', '梨', '香蕉'}
```

```
>>> %Run 8_8set.py
{'梨', '苹果', '香蕉'}
```

set集合：

- Python中的集合是一种数据结构，它可以存储一组不同的元素。例如，一个水果集
- 合可以包含苹果、梨子和香蕉，每个元素只出现一次。
- 以下是一个简单的Python代码示例，它创建了一个集合。我们可以使用“add”方
- 法向集合中添加元素，使用“remove”方法删除元素，并使用“len”方法获取集
- 合的大小。例如：

```
fruits = {"苹果", "梨", "香蕉"}
fruits.add("桔子")
fruits.remove("香蕉")
print(fruits)
print(len(fruits))
```

```
>>> %Run 8_9set.py
{'梨', '苹果', '桔子'}
3
```

这将输出以下内容：

- 这是因为集合中的元素是无序的，而且每个元素只出现一次。您可以使用集合来查
- 找不同的元素、删除重复项，或者检查两个集合之间的差异。

集合：set()函数

- Python中的set()函数是用来创建一个集合的函数。集合是一种无序的数据结构，其中每个元素都是唯一的。
- 例如，我们可以使用set方法来创建一个包含多个颜色的集合：

```
colors = set(['红色', '蓝色', '绿色','红色', '蓝色'])
```

- 这将创建一个名为colors的集合，其中包含三个元素：红色，蓝色和绿色。由于集合是无序的，它们可能以任何顺序出现。
- 我们可以使用一些其他的set方法来操作这个集合，比如添加新的元素：

```
colors.add('黄色')
print(colors)
```

```
>>> %Run 8_10set.py
{'蓝色', '绿色', '红色', '黄色'}
```

- 现在，colors集合中有四个元素：红色，蓝色，绿色和黄色。

集合：

我们可以使用set方法来创建一个包含多个颜色的集合：

```
colors = set(['红色', '蓝色', '绿色','红色', '蓝色'])
```

这将创建一个名为colors的集合，其中包含三个元素：红色，蓝色和绿色

我们还可以使用in方法来查找集合中的元素：

```
if "3" in colors:
    print("3在集合中")
else:
    print("3不在集合中")
```

```
>>> %Run 8_11set.py
{'黄色', '蓝色', '红色', '绿色'}
3不集合中

>>>
```

集合：运算

- Python中的set（集合）是一种数据类型，它可以存储一组唯一的元素。交集、并集和差集是集合运算中常用的概念。
- 假设一个集合A，A={1,2,3}，有一个集合B，：B={3，4，5}。
- 交集：交集是指两个集合中都包含的元素。他们的交集是{3}。
- 并集：并集是指两个集合中所有的元素。在这个例子中，他们的并集是{1,2,3 ,4,5}。
- 差集：差集是指在一个集合中出现但在另一个集合中没有出现的元素。因此A和B的差集是{1,2}。

集合：运算

```
1  A={1,2,3}
2  B={3,4,5}
3
4  #交集
5  print(A&B)
6  #并集
7  print(A|B)
8  #差集
9  print(A-B)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
 {3}
 {1, 2, 3, 4, 5}
 {1, 2}
>>>
```

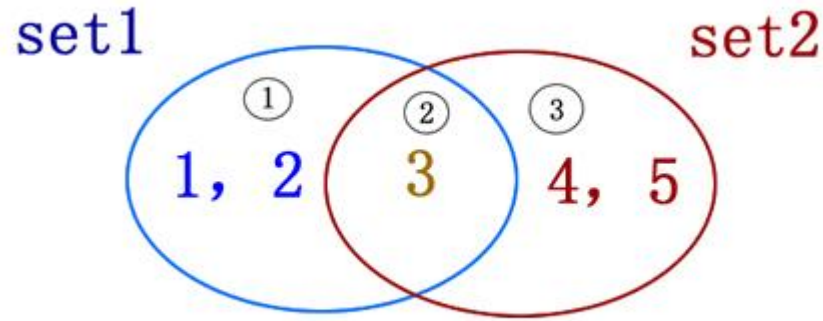


图 1 集合示意图

集合总结

集合类型的定义

集合是多个元素的无序组合

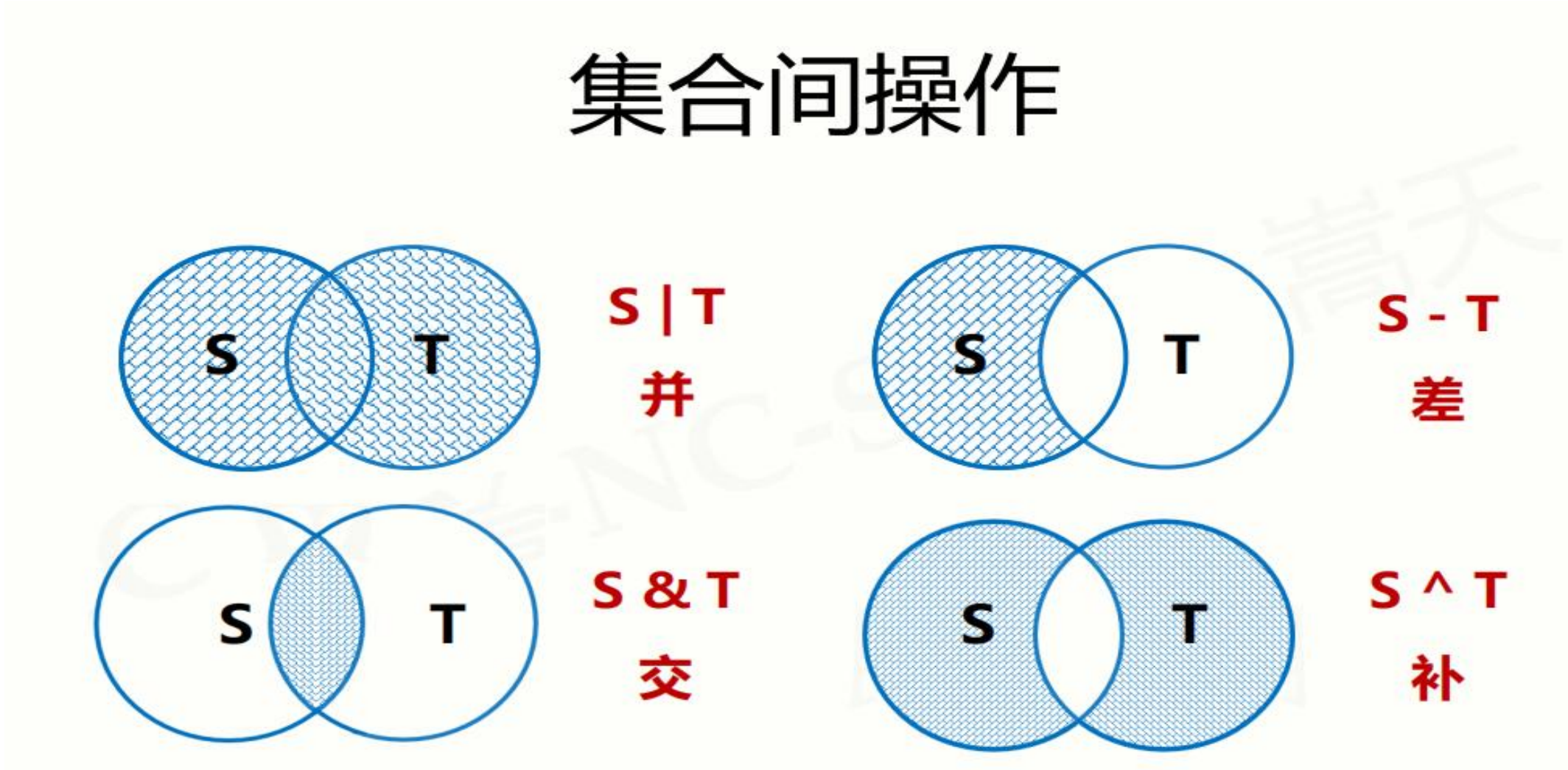
- **集合用大括号 {} 表示，元素间用逗号分隔**
- **建立集合类型用 {} 或 set()**
- **建立空集合类型，必须使用set()**

集合总结:

集合类型的定义

```
>>> A = {"python", 123, ("python",123)}   #使用{}建立集合
{123, 'python', ('python', 123)}
>>> B = set("pypy123")                    #使用set()建立集合
{'1', 'p', '2', '3', 'y'}
>>> C = {"python", 123, "python",123}
{'python', 123}
```

集合总结:



集合总结：

集合操作符

6个操作符

操作符及应用	描述
S | T	返回一个新集合，包括在集合S和T中的所有元素
S - T	返回一个新集合，包括在集合S但不在T中的元素
S & T	返回一个新集合，包括同时在集合S和T中的元素
S ^ T	返回一个新集合，包括集合S和T中的非相同元素
S <= T 或 S < T	返回True/False，判断S和T的子集关系
S >= T 或 S > T	返回True/False，判断S和T的包含关系

集合总结:

集合类型的定义

```
>>> A = {"p", "y" , 123}
>>> B = set("pypy123")
>>> A-B
{123}
>>> B-A
{'3', '1', '2'}
>>> A&B
{'p', 'y'}
>>> A|B
{'1', 'p', '2', 'y', '3', 123}
>>> A^B
{'2', 123, '3', '1'}
```

集合总结：

集合处理方法

操作函数或方法	描述
S.add(x)	如果x不在集合S中，将x增加到S
S.discard(x)	移除S中元素x，如果x不在集合S中，不报错
S.remove(x)	移除S中元素x，如果x不在集合S中，产生KeyError异常
S.clear()	移除S中所有元素
S.pop()	随机返回S的一个元素，更新S，若S为空产生KeyError异常

集合总结：

集合处理方法

操作函数或方法	描述
S.copy()	返回集合S的一个副本
len(S)	返回集合S的元素个数
x in S	判断S中元素x，x在集合S中，返回True，否则返回False
x not in S	判断S中元素x，x不在集合S中，返回False，否则返回True
set(x)	将其他类型变量x转变为集合类型

集合总结:

集合处理方法

```
>>> A = {"p", "y" , 123}
>>> for item in A:
        print(item, end="")
p123y
>>> A
{'p', 123, 'y'}
```

```
>>> try:
        while True:
            print(A.pop(), end=""))
    except:
        pass
p123y
>>> A
set()
```

元组概念：

- Python中的元组是一种有序的数据结构，它与列表非常相似，但不同之处在于元组不可更改。例如，如果你有一个元组包含你的生日和年龄，你可以使用以下代码创建它：

```
my_tuple = ('2008-01-01', 13)
```

这个元组有两个元素，一个是字符串类型的生日，一个是整数类型的年龄。你可以像这样通过索引访问元组中的元素：

```
print(my_tuple[0])
print(my_tuple[1])
```

```
>>> %Run 8_12turple.py
2008-01-01
13
Traceback (most recent call last):
  File "F:\2023春python0303\第八次课课件0308\examplecode\8_12turple.py", line 4, in <module>
    my_tuple[1] = 14
TypeError: 'tuple' object does not support item assignment
```

- 注意，由于元组是不可更改的，以下代码将会产生错误：

```
my_tuple[1] = 14  # 报错：TypeError: 'tuple' object does not support item assignment
```

- 因此，如果你需要一个**不可更改**的数据结构，你可以使用元组。

元组概念：

- Python元组是一种数据结构，它是一组有序的值的集合，这些值可以是数字、字符串或其他数据类型。**元组用小括号 () 表示**，其中的值用逗号分隔。
- 例如，我们可以创建一个包含三个元素的元组，分别是一个数字、一个字符串和一个布尔值：

```
my_tuple = (42, "Hello, World!", True)
```

- 这个元组包含一个数字 42，一个字符串 "Hello, World!" 和一个布尔值 True。我们可以通过索引来访问元组中的每个元素，例如：

```
print(my_tuple[0])  # 输出：42
print(my_tuple[1])  # 输出："Hello, World!"
print(my_tuple[2])  # 输出：True
```

- 元组是不可变的，也就是说，一旦创建后就**不能修改其**中的值。这使得元组在一些情况下比列表更加适用，例如用于表示一些常量或配置信息。

元组总结：

元组类型定义

元组是序列类型的一种扩展

- 元组是一种序列类型，一旦创建就不能被修改
- 使用小括号 () 或 tuple() 创建，元素间用逗号，分隔
- 可以使用或不使用小括号

```
def func():
    return 1,2
```

元组总结：

元组类型定义

```
>>> creature = "cat", "dog", "tiger", "human"
>>> creature
('cat', 'dog', 'tiger', 'human')
>>> color = (0x001100, "blue", creature)
>>> color
(4352, 'blue', ('cat', 'dog', 'tiger', 'human'))
```

元组总结：

元组类型操作

元组继承序列类型的全部通用操作

- 元组继承了序列类型的全部通用操作
- 元组因为创建后不能修改，因此没有特殊操作
- 使用或不使用小括号

问题：？

元组总结：