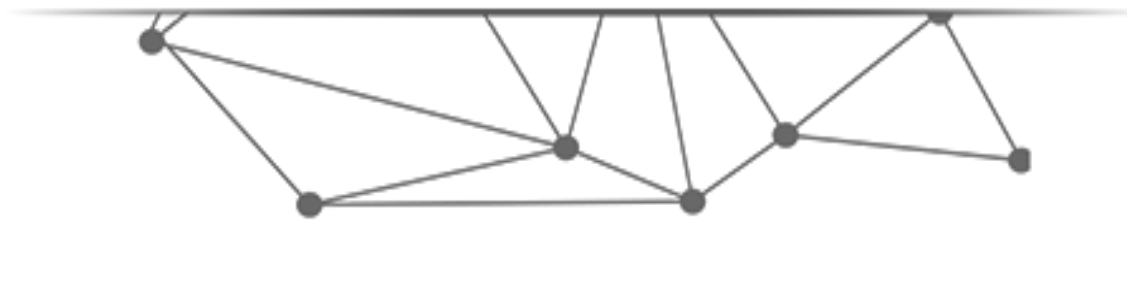
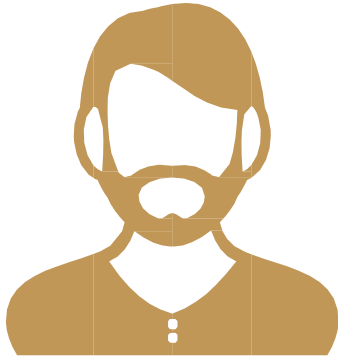


3.3 组合类型（集合、字典）





集合类型及操作



- 集合类型定义
- 集合操作符
- 集合处理方法
- 集合类型应用场景



集合类型的定义

集合是多个元素的无序组合

- **集合类型与数学中的集合概念一致**
- **集合元素之间无序，每个元素唯一，不存在相同元素**
- **集合元素不可更改，不能是可变数据类型** **为什么？**



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}
```



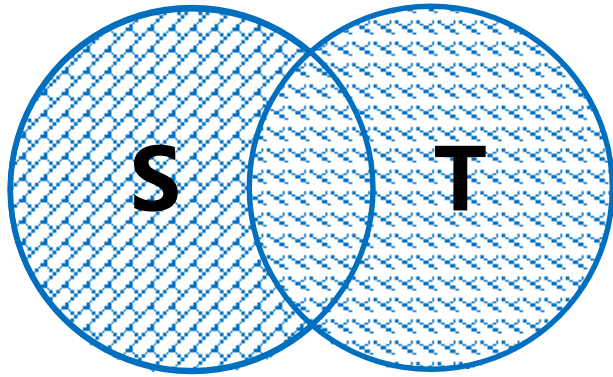
集合操作符

6个操作符

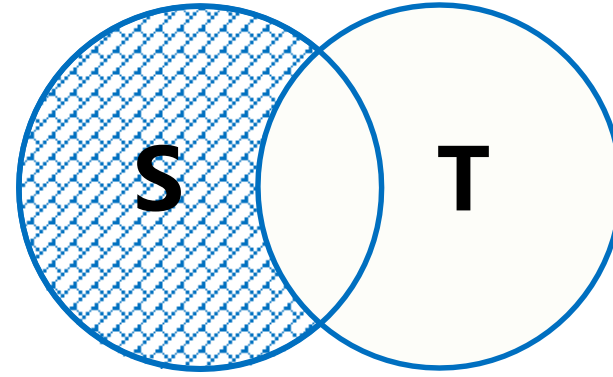
操作符及应用	描述
$S \mid T$	并，返回一个新集合，包括在集合S和T中的所有元素
$S - T$	差，返回一个新集合，包括在集合S但不在T中的元素
$S \& T$	交，返回一个新集合，包括同时在集合S和T中的元素
$S \wedge T$	补，返回一个新集合，包括集合S和T中的非相同元素
$S \leq T$ 或 $S < T$	返回True/False，判断S和T的子集关系
$S \geq T$ 或 $S > T$	返回True/False，判断S和T的包含关系



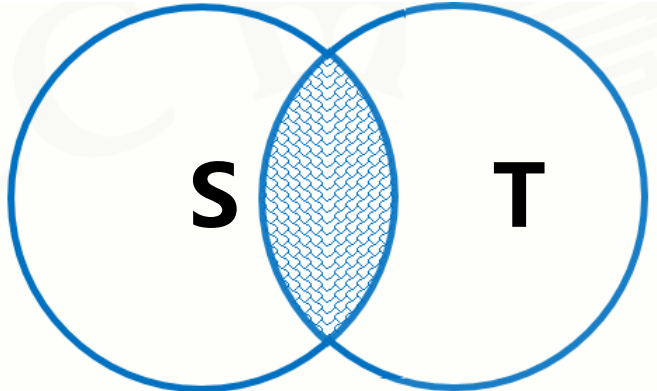
集合操作符



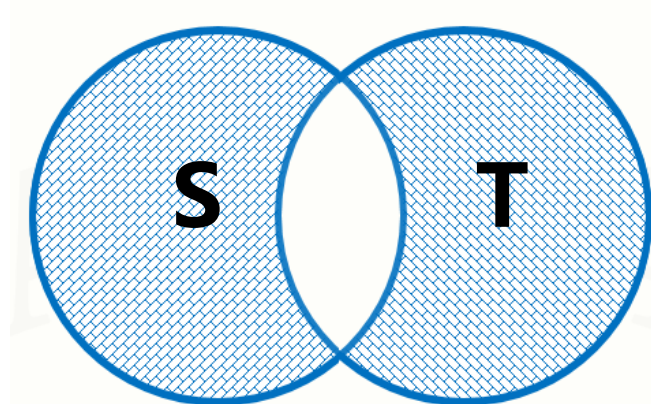
$S \cup T$
并



$S - T$
差



$S \cap T$
交



$S \Delta T$
补



集合操作符

4个增强操作符

操作符及应用	描述
$S = T$	并，更新集合S，包括在集合S和T中的所有元素
$S -= T$	差，更新集合S，包括在集合S但不在T中的元素
$S \&= T$	交，更新集合S，包括同时在集合S和T中的元素
$S \wedge= T$	补，更新集合S，包括集合S和T中的非相同元素



集合类型操作

```
>>> 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'}
```



集合处理方法

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



集合处理方法

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



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()`



PY01B26 博采



集合类型应用场景

包含关系比较

```
>>> "p" in {"p", "y", 123}
```

```
True
```

```
>>> {"p", "y"} >= {"p", "y", 123}
```

```
False
```



集合类型应用场景

数据去重：集合类型所有元素无重复

```
>>> ls = ["p", "p", "y", "y", 123]
```

```
>>> s = set(ls)      # 利用了集合无重复元素的特点
```

```
{'p', 'y', 123}
```

```
>>> lt = list(s)      # 还可以将集合转换为列表
```

```
['p', 'y', 123]
```

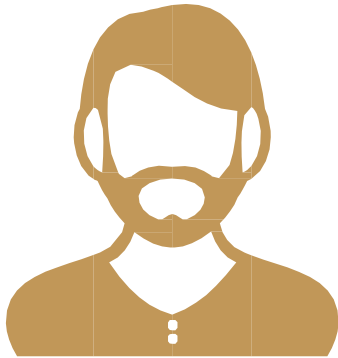


集合类型及操作

- 集合使用{}和set()函数创建
- - 集合间操作：交(&)、并(|)、差(-)、补(^)、比较(>=<)
- 集合类型方法：.add()、.discard()、.pop()等
- 集合类型主要应用于：包含关系比较、数据去重



字典类型及操作



- 字典类型定义
- 字典处理函数及方法
- 字典类型应用场景



字典类型定义



字典类型定义

理解“映射”

- 映射是一种键(索引)和值(数据)的对应





字典类型定义

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"



字典类型定义

理解“映射”

- 映射是一种键(索引)和值(数据)的对应

["python", 123, ".io"]



0

1

2



内部颜色: 蓝色

外部颜色: 红色

序列类型由0..N整数作为数据的默认索引 映射类型则由用户为数据定义索引



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



字典处理函数及方法

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



字典处理操作

```
>>> 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'])
```



字典类型操作函数和方法

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



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')
```



字典功能默写

- 定义空字典d
- 向d新增2个键值对元素
- 修改第2个元素
- 判断字符"c"是否是d的键
- 计算d的长度
- 清空d

- `>>> d = {}`
- `>>> d["a"] = 1; d["b"] = 2`
- `>>> d["b"] = 3`
- `>>> "c" in d`
- `>>> len(d)`
- `>>> d.clear()`



字典类型应用场景

映射的表达



PY01B29 无所不包

- 映射无处不在，键值对无处不在
- 例如：统计数据出现的次数，数据是键，次数是值
- 最主要作用：表达键值对数据，进而操作它们



字典类型应用场景

元素遍历

for k *in* d :

<语句块>



字典类型及操作

- 映射关系采用键值对表达
- 字典类型使用{}和dict()创建，键值对之间用:分隔
- d[key] 方式既可以索引，也可以赋值
- 字典类型有一批操作方法和函数，最重要的是.get()