目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

14 这么多的数据结构（二）：字典、集合

更新时间：2019-09-16 10:21:49

> 天才就是长期劳动的结果。
>
> ——牛顿

四、字典

字典是一种用来存放若干键值对的数据类型。

什么是键值对呢？**键值对**就是两个对象，其中一个是用来做定位的数据，叫做**键**（Key），另一个是要存放的数据，叫做**值**（Value）。在字典中，键值对作为一个整体被存放，我们可以通过键来快速获取到对应的值。

在 Python 中字典用花括号（ `{}` ）来表示，键值对以 `键:值` 的方式写入花括号中，有多个键值对时用逗号分隔。

如 `{'010': 'beijing', '021': 'shanghai'}` 便是一个字典，其中包含两个键值对

- `'010': 'beijing'`
- `'021': 'shanghai'`

使用时，我们可以通过 `'010'` 快速查询到其对应的值是 `'beijing'` 。这就好比现实中的一本字典一样，通过拼音或偏旁来映射一个具体的字词，以此来实现字词的快速查找，而这个拼音或偏旁就相当于 Python 字典的键，而字词就相当于 Python 字典的值，它们在字典中是映射关系。

Python 字典中的键是有要求的，需要是不可变的类型，如元组、字符串、数字。而字典中的值可以是任意类型。字典本身是可变的，我们可以向其中添加、删除、修改键值对。

因为字典不是序列，更不是有序的，所有它没有列表那样的索引，也不能保证每个键值对的存放次序。

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

1. 创建空的字典

```
字典 = {}
```

```
>>> empty_dict = {}
>>> empty_dict
{}
```

2. 创建包含键值对的字典

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

如城市和其对应的电话区号：

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes
{‘beijing’：‘010’，‘shanghai’：‘021’}
```

字典键值对的添加

1. 向字典中增加键值对

```
字典[键] = 值
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes[‘tianjin’] = '022’
>>> codes
{‘beijing’：‘010’，‘shanghai’：‘021’，‘tianjin’：‘022’}
```

使用这种方式时，若字典中没有这个键，则会创建这个键值对；若字典中原本已有这个键，则是修改键所对应的值。

键值对的获取

1. 通过键获取值

```
值 = 字典[键]
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes[‘beijing’]
‘010’
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes[‘a’]
Traceback (most recent call last):
    File “”, line 1, in
KeyError: ‘a’
```

2. 通过键获取值（ get 方法）

如果通过键获取值时不希望 KeyError 异常抛出，可以使用 get 方法，若键不存在，则直接返回 None 。

```
值 = 字典.get(键)
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.get(‘a’)
>>>
```

返回的 None 代表什么都没有，所以没有任何值显示。

也可以给 get 方法传递第二个参数作为默认值，使得键不存在时直接返回默认值。

```
值 = 字典.get(键, 默认值)
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.get(‘a’，‘000’)
>>> ‘000’
```

3. 判断字典中是否包含某个键

```
布尔值 = 键 in 字典
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> ‘beijing’ in codes
True
>>> ‘guangzhou’ in codes
False
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
键的列表 = 字典.keys()
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.keys()
dict_keys([‘beijing’，‘shanghai’])
```

获取到的所有键是以迭代器的形式存在，至于什么是迭代器我们将在之后的章节中介绍。在这里我们可以用 `list()` 函数将迭代器转换为列表。如下：

```
>>> list(codes.keys())
[‘beijing’，‘shanghai’]
```

5. 获取所有值

```
值的列表 = 字典.values()
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.values()
dict_values([‘010’，‘021’])
```

获取到的所有值是以迭代器的形式存在，我们用 `list()` 函数将迭代器转换为列表。如下：

```
>>> list(codes.values())
[‘010’，‘021’]
```

6. 获取所有键值对的列表

```
值的列表 = 字典.items()
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.items()
dict_items([(‘beijing’，‘010’)，(‘shanghai’，‘021’)])
```

获取到的所有键值对是以迭代器的形式存在，我们用 `list()` 函数将迭代器转换为列表。如下：

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

> [(‘beijing’，‘010’), (‘shanghai’，‘021’)]

列表中的每一个元素是都是二元组（即包含两个元素的元组），每个二元组的第一个元素是键，第二个元素是值。

字典键值对的删除

1. 通过键删除键值对

可以使用 pop 方法删除一个键值对，并将值返回。

```
值 = 字典.pop(键)
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.pop(‘beijing’)
’010’
>>> codes
{‘shanghai’：‘021’}
```

如果 pop 一个不存在的键，则会抛出 KeyError 异常：

```
>>> codes.pop(‘a’)
Traceback (most recent call last):
  File “”, line 1, in
KeyError: ‘a’
```

如果你不希望异常抛出，可以传递 pop 方法的第二个参数作为默认值。默认值仅在键不存在时生效，此时方法将直接返回这个默认值，且跳过删除操作：

```
值 = 字典.pop(键, 默认值)
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.pop(‘guangzhou’，‘000’)
’000’
```

2. 通过键删除键值对（ del 方法）

也可以通过关键字 del 来删除键值对。

```
del 字典[键]
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> codes
{‘shanghai’：‘021’}
```

3. 随机删除一个键值对

使用 popitem 随机删除一个键值对，并返回这个键值对的二元组，二元组的第一个元素是键，第二个元素是值。

```
键值二元组 = 字典.popitem()
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.popitem()
(‘shanghai’，‘021’)
>>> codes
{‘beijing’：‘010’}
```

4. 清空所有键值对

```
键值二元组 = 字典.clear()
```

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> codes.clear()
>>> codes
{}
```

字典中键值对修改

1. 修改键对应的值

```
字典[键] = 值
```

```
>>> codes = {‘beijing’：‘010’}
>>> codes[‘beijing’] = '021’
>>> codes
{‘beijing’：‘021’}
```

如果键不存在，则创建键值对。

2. 用字典批量更新键值对

```
字典.update(另一字典)
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> codes.update({‘guangzhou’：‘020’，‘shanghai’：‘000’})
>>> codes
{‘beijing’：‘010’，‘shanghai’：‘000’，‘guangzhou’：‘020’}
```

可以看到字典中新增了 `'guangzhou': '020'` 这个键值对，同时将 `'shanghai': '021'` 修改为 `'shanghai': '000'` 。

什么时候用字典

字典的显著优势是可以通过键快速地查询数据。字典中的元素以键值对的形式存在，使用时通过键来获取和修改值，由于字典内部的特殊实现，字典通过键获取值的效率非常高。

如果我们希望将批量的数据存放起来，并且在需要时能以很高的执行效率来获取其中某个指定的数据，这时就可以使用字典。除此之外，如果我们想在程序中暂时维护一个映射关系，也可以使用字典，因为字典本质上就是一个映射关系。

如，我们可以将城市名和对应的区号保存在字典中，这样就可以通过城市名快速地查询到其区号，而不需要进行遍历。

```
area_codes = {
    '北京': '010',
    '上海': '021',
    '天津': '022',
    '重庆': '023',
    '沈阳': '024',
    '南京': '025',
    '武汉': '027',
    '成都': '028',
}
```

```
>>> area_codes[‘成都’]
‘028’
>>> area_codes[‘南京’]
‘025’
```

五、集合

集合是一个用于存放批量元素的数据类型，它不是有序的，其中的元素**没有顺序**关系。集合中的元素**没有重复**，重复的元素将被自动剔除最终只留下一个。

集合也是用花括号（ `{}` ）来表示，不同于字典的是，花括号中放的是一个个数据，而不是键值对。

集合是可变的，我们可以向其中添加、删除、修改元素。

创建集合

1. 创建包含元素的集合

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> numbers = {1, 2, 3}
>>> numbers
{1, 2, 3}
```

2. 创建空集合

```
集合 = set()
```

注意创建空集合不能直接使用 `{}`，那样是表示空字典，而是使用 `set()`，这才表示空集合。

```
>>> empty_set = set()
>>> empty_set
set()
```

集合元素的添加

1. 向集合中添加一个元素

```
集合.add(元素)
```

```
>>> numbers = {1, 2}
>>> numbers.add(3)
>>> numbers
{1, 2, 3}
```

向集合中添加重复元素时，会被去重处理。

```
>>> numbers = {1, 2}
>>> numbers.add(2)
>>> numbers
{1, 2}
```

2. 从另一集合中批量添加元素

```
集合.update(另一集合)
```

```
>>> numbers_1 = {1, 2}
>>> numbers_2 = {2, 3, 4}
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
{1, 2, 3, 4}
```

可以看到，集合 numbers_2 中的所有元素被添加到了集合 numbers_1 中，并且其中重复的元素被剔除仅保留一份。

集合元素的获取

集合不能像列表那样通过索引来获取元素，也不能像字典那样通过键来获取值，集合没法直接获取到某个指定的元素。想要获取元素，只能通过遍历的方式。

虽然集合不能直接获取到元素，但是我们依然可以用 in 关键字来判断元素是否存在于集合中。

1. 查看元素是否存在于集合中

```
布尔值 = 元素 in 集合
```

```
>>> letters = {‘a’, ‘b’, ‘c’}
>>> ‘a’ in letters
True
>>> ‘z’ in letters
False
```

集合元素的删除

1. 随机删除一个元素，并返回这个元素

```
元素 = 集合.pop()
```

使用 pop 方法随机删除一个元素的时候，这个元素会被返回。

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

2. 删除一个指定的元素

```
集合.remove(元素)
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> numbers.remove(1)
>>> numbers
{2, 3}
```

如果要删除的元素不存在，则抛出 KeyError 异常：

```
>>> numbers = {1, 2, 3}
>>> numbers.remove(4)
Traceback (most recent call last):
    File “ ”, line 1, in
KeyError: 4
```

3. 删除一个指定的元素，且不抛出 KeyError 异常

使用 remove 方法删除一个不存在的元素时，会抛出 KeyError 异常，如果我们不想让异常抛出，可以使用 discard 方法。

```
集合.discard(元素)
```

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

4. 清空所有元素

```
集合.clear()
```

与列表和字典一样，想要清空所有元素，可以使用 clear 方法。

```
>>> numbers = {1, 2, 3}
>>> numbers.clear()
>>> numbers
set()
```

顺便考大家一个问题，为什么元组没有这个方法？因为元组是不可变的！我们不能删除元组的元素，也不能添加和修改元素。

集合的运算

看到这里你可能会想，集合不就是阉割版的列表嘛？不是的，集合的功能不止于此。

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

1. 求交集

可以通过 intersection 方法求多个集合的交集。

```
交集 = 集合1.intersection(集合2, 集合3, 集合N)
```

```
>>> numbers_1 = {1, 2, 3}
>>> numbers_2 = {2, 3, 4}
>>> numbers_3 = {3, 4, 5}
>>> numbers_1.intersection(numbers_2, numbers_3)
{3}
```

也可以直接使用**与运算符** & 来代替，完全等效：

```
交集 = 集合1 & 集合2 & 集合N
```

```
>>> numbers_1 & numbers_2 & numbers_3
{3}
```

2. 求并集

```
交集 = 集合1.union(集合2, 集合3, 集合N)
```

```
>>> numbers_1 = {1, 2, 3}
>>> numbers_2 = {2, 3, 4}
>>> numbers_3 = {3, 4, 5}
>>> numbers_1.union(numbers_2, numbers_3)
{1, 2, 3, 4, 5}
```

也可以直接使用**或运算符** | 来代替，完全等效：

```
交集 = 集合1 | 集合2 | 集合N
```

```
>>> numbers_1 | numbers_2 | numbers_3
{1, 2, 3, 4, 5}
```

3. 求差集

```
交集 = 集合1.difference(集合2, 集合3, 集合N)
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> numbers_1 = {1, 2, 3}
>>> numbers_2 = {2, 3, 4}
>>> numbers_3 = {3, 4, 5}
>>> numbers_1.difference(numbers_2, numbers_3)
{1}
```

也可以直接使用运算符 - 来代替，完全等效：

```
交集 = 集合1 - 集合2 - 集合N
```

```
>>> numbers_1 - numbers_2 - numbers_3
{1}
```

4. 判断是否为子集

```
布尔值 = 集合1.issubset(集合2)
```

判断 集合1 是否为 集合2 的子集。

```
>>> numbers_1 = {2, 3}
>>> numbers_2 = {1, 2, 3}
>>> numbers_1.issubset(numbers_2)
True
>>> numbers_3 = {3, 4}
>>> numbers_1.issubset(numbers_3)
False
```

5. 判断是否为超集

```
布尔值 = 集合1.issuperset(集合2)
```

判断 集合1 是否为 集合2 的子集。

```
>>> numbers_1 = {1, 2, 3}
>>> numbers_2 = {2, 3}
>>> numbers_1.issuperset(numbers_2)
True
>>> numbers_3 = {3, 4}
>>> numbers_1.issuperset(numbers_3)
False
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

集合运算在编程的有什么用呢？以差集为例举个例子。

假如大学里有一个班，班里同学的花名册是「赵大，钱二，孙三，李四，周五，吴六，郑七，王八」。有一天上课，老师要求同学们在一张白纸上签到，大家陆续写上了自己的名字，上面有「周五，李四，王八，赵大，钱二，冯九，陈十」。哪些人缺席了呢？

要判断哪些人缺席了，通常的做法时，逐一从签到表上取出名字，然后去花名册上寻找并做标记，最终花名册上没被标记的名字便是缺席的。有些麻烦，这可苦了助教了。

我们可以用 Python 集合来快速解决这个问题。将花名册上的名字保存在集合中，将签到表上的名字保存在另一个集合中，然后求一下差集。如下：

```
>>> 花名册 = {‘赵大’，‘钱二’，‘孙三’，‘李四’，‘周五’，‘吴六’，‘郑七’，‘王八’}
>>> 签到表 = {‘周五’，‘李四’，‘王八’，‘赵大’，‘钱二’，‘冯九’，‘陈十’}
>>> 花名册 - 签到表
{‘吴六’，‘孙三’，‘郑七’}
```

吴六，孙三，郑七没来！

我们用反过来用 签到表 - 花名册 看看：

```
>>> 签到表 - 花名册
{‘陈十’，‘冯九’}
```

还有两个旁听生！

什么时候用集合

集合非常重要的一个特性是元素无重复，每个元素都是唯一的，重复的元素将被自动剔除（去重）。

所以如果我们需要存放一系列的数据，并且不希望其中出现重复，那么就可以使用集合。

另外如果想计算两个数据集的交集、并集、差集，使用集合来承载数据再合适不过了，集合自带的集合运算能轻松解决这些问题。

获取字典和集合中的元素数量

我们可以通过 len() 函数来获取字典中的键值对数量和集合中的元素数量。

```
>>> codes = {‘beijing’：‘010’，‘shanghai’：‘021’}
>>> len(codes)
2
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合 最近阅读

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> len(numbers)
4
```

总结

字典是一种用来存放若干键值对的数据类型，可通过键来快速查找值。

字典的键需要是不可变的类型，如数字，字符串和元组。字典的值可以是任意类型。字典本身是可变的，所以可向其中添加、修改、删除键值对。

集合是一个用于存放批量元素的序列。它不是有序的，且元素**不会有重复**。集合也是可变的，我们可以向其中添加、删除、修改元素。

← 13 这么多的数据结构（一）：列表、元祖、字符串

15 Python大法初体验：内置函数 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论

千学不如一看，千看不如一练