Python 程序设计基础

Python Programming

集合的概念

- 集合类型与数学中集合的概念一致。
- Python 中的集合分为可变集合（ set ）和不可变集合（ frozenset ）。
- 集合中的值称为元素，既可以包含同类型元素也可以包含不同类型元素。集合中的元素不可重复，元素之间也没有顺序关系。
- 集合中的元素可以是任何可哈希（ hash ）对象。整数、浮点数、布尔值、字符串、元组以及不可变集合等都是可哈希对象，列表、字典以及可变集合等都是非可哈希对象。
- 集合中的元素是依据哈希码来放置的。哈希码是根据可哈希对象的值计算出来的一个惟一值。不能像序列类型那样通过下标来访问集合的元素。
- 若不关心元素的顺序，使用集合来存储数据比使用列表效率更高。

集合的概念

- **集合中的元素用逗号分隔并且由一对花括号（ {} ）括住。**

```
>>> set1 = {1, 2, 3}
>>> set1
{1, 2, 3}
>>> set2 = {"red", "green", "blue"}
>>> set2
{'blue', 'red', 'green'}
>>> set3 = {2, "three", 4.5, True}
>>> set3
{True, 2, 4.5, 'three'}
>>> set4 = {"one", 2.0, 5, (100, 200)}
>>> set4
{2.0, 'one', 5, (100, 200)}
```

集合的概念

- 还可以使用 set 内置函数来创建集合。

```
>>> set1 = set()
>>> set1
set()
>>> set2 = set({})
>>> set2
set()
>>> set3 = set(range(1, 10))
>>> set3
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> set4 = set([x for x in range(1, 10)])
>>> set4
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> set5 = set("abcda")
>>> set5
{'d', 'a', 'b', 'c'}
```

  - 一个不包含任何元素的集合被称为空集合；可以用 set() 或 set({}) 创建一个空集合。
  - 注意： {} 不是创建空集合，而是创建空字典。

集合的概念

- **若希望集合中的元素具有稳定性，像元组一样不能随意增加或删除集合中的元素，可以用 frozenset 函数创建不可变集合。**

```
>>> set1 = frozenset()
>>> set1
frozenset()
>>> set2 = frozenset([1, 2, 3])
>>> set2
frozenset({1, 2, 3})
```

集合的基本操作

- **使用内置函数。**
  - **len 函数返回一个集合中的元素个数。**
  - **max 函数和 min 函数分别返回一个集合（元素必须是相同类型）中的最大值元素和最小值元素。**
  - **sum 函数返回一个集合（元素必须是数值）中所有元素的和。**

```
>>> set1 = {55, -22, 3, 45, 11, 62, 38, 985, 211}
>>> len(set1)
9
>>> max(set1)
985
>>> min(set1)
-22
>>> sum(set1)
1388
```

集合的基本操作

- **运算符。**
  - **使用 in 或 not in 运算符来判断元素是否在集合中。**

```
>>> set1 = {"C++", "Java", "Python"}
>>> "Python" in set1
True
>>> "Java" not in set1
False
```

  - **使用 is 或 is not 来判断两个集合是否是同一个对象。**

```
>>> set1 = {"C++", "Java", "Python"}
>>> set2 = {"C++", "Java", "Python"}
>>> id(set1)
2220667595688
>>> id(set2)
2220668086088
>>> set1 is set2
False
```

集合的基本操作

- 使用关系运算符 == 和 != 判断两个集合是否包含相同的元素。下面例子中，尽管 set1 和 set2 的元素顺序不同，但是这两个集合包含相同的元素。

```
>>> set1 = {1, 2, 3, 7, 9, 0, 5}
>>> set2 = {1, 3, 2, 7, 5, 0, 9}
>>> set1 == set2
True
```

- 使用关系运算符 < 可以判断一个集合是否是另一个集合的真子集，使用关系运算符 <= 可以判断一个集合是否是另一个集合的子集。注意：一个集合是它本身的子集。下面例子中， set1 是 set2 的真子集， set1 是其自身的子集。

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 < set2
True
>>> set1 <= set1
True
```

集合的基本操作

- 使用关系运算符 > 可以判断一个集合是否是另一个集合的真超集，使用关系运算符 >= 可以判断一个集合是否是另一个集合的超集。注意：一个集合是它本身的超集。下面例子中， set2 是 set1 的真超集， set2 是其自身的超集。

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set2 > set1
True
>>> set2 >= set2
True
```

- 两个集合的并集是一个包含这两个集合所有元素的新集合。可以使用 | 运算符来实现这个操作。

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 | set2
{1, 2, 7, 8, 9}
```

集合的基本操作

- **两个集合的交集是一个包含这两个集合共有元素的新集合。可以使用 & 运算符来实现这个操作。**

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 & set2
{8, 9, 7}
```

- **两个集合的差集是一个包含出现在第一个集合而不出现在第二个集合的元素的新集合。可以使用 - 运算符来实现这个操作。**

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 - set2
set()
```

集合的基本操作

- **两个集合的对称差集是一个包含这两个集合共有元素之外所有元素的新集合。可以使用 ^ 运算符来实现这个操作。**

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 ^ set2
{1, 2}
```

- **复合赋值运算： set1 |= set2 ，即 set1 = set1 | set2 ； set1 &= set2 ，即 set1 = set1 & set2 ； set1 -= set2 ，即 set1 = set1 - set2 ； set1 ^= set2 ，即 set1 = set1 ^ set2**

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1 |= set2
>>> set1
{1, 2, 7, 8, 9}
```

集合的基本操作

- **集合运算方法。**
  - **除了使用运算符，还可以使用方法来完成集合运算。**
  - **使用 issubset 方法判断子集。使用 issuperset 方法判断超集。**
  - **使用 union 方法来实现集合并集操作。使用 intersection 方法来实现集合交集操作。使用 difference 方法来实现集合差集操作。使用 symmetric_difference 方法来实现集合对称差集操作。**

```
>>> set1 = {7, 8, 9}
>>> set2 = {7, 1, 9, 2, 8}
>>> set1.issubset(set2)
True
>>> set2.issuperset(set1)
True
```

```
>>> set1.union(set2)
{1, 2, 7, 8, 9}
>>> set1.intersection(set2)
{8, 9, 7}
>>> set1.difference(set2)
set()
>>> set1.symmetric_difference(set2)
{1, 2}
```

集合的基本操作

- **遍历集合。**
  - **注意：集合不是序列类型，不能通过下标或切片操作来访问集合中的元素。**
  - **使用 for 语句。**

```
>>> set1 = {2, 3, 5, 2, 33, 21}
>>> for value in set1:
        print(value, end=' ')

33 2 3 5 21
```

集合的基本操作

- 集合解析。
  - 集合解析提供了一种创建集合的简洁方式。
  - 一个集合解析由花括号组成。花括号内包含后跟一个 for 子句的表达式，之后是 0 或多个 for 子句或 if 子句。集合解析产生一个由表达式求值结果组成的集合。

    {expr for iter_var in iterable}

    首先循环 iterable 里所有内容，每一次循环，都把 iterable 里相应内容放到 iter_var 中，再在 expr 中应用该 iter_var 的内容，最后用 expr 的计算值生成一个集合。

    {expr for iter_var in iterable if cond_expr}

    加入了判断语句，只有满足条件的才把 iterable 里相应内容放到 iter_var 中，再在 expr 中应用该 iter_var 的内容，最后用 expr 的计算值生成一个集合。

集合的基本操作

```
>>> set1 = {value for value in range(0, 11, 2)}
>>> print(set1)
{0, 2, 4, 6, 8, 10}
>>> set2 = {0.5 * value  for value in set1}
>>> print(set2)
{0.0, 1.0, 2.0, 3.0, 4.0, 5.0}
>>> set3 = {value for value in set2 if value > 2.0}
>>> print(set3)
{3.0, 4.0, 5.0}
```

集合的基本操作

- **向可变集合添加元素。**
  - **add(x) 方法。将元素 x 添加到可变集合中。**

```
>>> set1 = {"C++", "Java", "Python"}
>>> set1
{'C++', 'Python', 'Java'}
>>> set1.add("C#")
>>> set1
{'C++', 'C#', 'Python', 'Java'}
```

集合的基本操作

- update 方法。用两个集合的并集更新第一个集合。
- intersection_update 方法。用两个集合的交集更新第一个集合。

```
>>> set1 = {"C++", "Java", "Python"}
>>> set1
{'C++', 'Python', 'Java'}
>>> set2 = {"C#", "JavaScript"}
>>> set2
{'C#', 'JavaScript'}
>>> set1.update(set2)
>>> set1
{'Java', 'JavaScript', 'C++', 'C#', 'Python'}
>>> set2
{'C#', 'JavaScript'}
```

```
>>> set1 = {"C++", "Java", "Python"}
>>> set1
{'C++', 'Python', 'Java'}
>>> set2 = {"C#", "JavaScript", "Python"}
>>> set2
{'C#', 'Python', 'JavaScript'}
>>> set1.intersection_update(set2)
>>> set1
{'Python'}
>>> set2
{'C#', 'Python', 'JavaScript'}
```

集合的基本操作

- **difference_update 方法。用两个集合的差集更新第一个集合。**
- **symmetric_difference_update 方法。用两个集合的对称差集更新第一个集合。**

```
>>> set1 = {"C++", "Java", "Python"}
>>> set1
{'C++', 'Python', 'Java'}
>>> set2 = {"C#", "JavaScript", "Python"}
>>> set2
{'C#', 'Python', 'JavaScript'}
>>> set1.difference_update(set2)
>>> set1
{'C++', 'Java'}
>>> set2
{'C#', 'Python', 'JavaScript'}
```

```
>>> set1 = {"C++", "Java", "Python"}
>>> set1
{'C++', 'Python', 'Java'}
>>> set2 = {"C#", "JavaScript", "Python"}
>>> set2
{'C#', 'Python', 'JavaScript'}
>>> set1.symmetric_difference_update(set2)
>>> set1
{'C++', 'C#', 'JavaScript', 'Java'}
>>> set2
{'C#', 'Python', 'JavaScript'}
```

集合的基本操作

- 从可变集合删除元素。
  - pop 方法。从可变集合中删除并返回一个元素。若可变集合为空集合，则抛出“ KeyError” 异常。

```
>>> set1 = set("cba")
>>> set1
{'b', 'a', 'c'}
>>> set1.pop()
'b'
>>> set1.pop()
'a'
>>> set1.pop()
'c'
>>> set1.pop()
Traceback (most recent call last):
  File "<pyshell#151>", line 1, in <module>
    set1.pop()
KeyError: 'pop from an empty set'
```

集合的基本操作

- remove(x) 方法。从可变集合中删除元素 x 。若可变集合中元素 x 不存在，则抛出“ KeyError” 异常。
- discard(x) 方法。从可变集合中删除元素 x 。若可变集合中元素 x 不存在，则不做任何事情。

```
>>> set1 = {55, -22, 3, 45, 11, 62, 38, 985}
>>> set1
{3, 38, -22, 11, 45, 55, 985, 62}
>>> set1.remove(11)
>>> set1
{3, 38, -22, 45, 55, 985, 62}
>>> set1.remove(88)
Traceback (most recent call last):
  File "<pyshell#167>", line 1, in <module>
    set1.remove(88)
KeyError: 88
```

```
>>> set1 = {55, -22, 3, 45, 11, 62, 38, 985}
>>> set1
{3, 38, -22, 11, 45, 55, 985, 62}
>>> set1.discard(11)
>>> set1
{3, 38, -22, 45, 55, 985, 62}
>>> set1.discard(88)
>>> set1
{3, 38, -22, 45, 55, 985, 62}
```

集合的基本操作

- clear() 方法。删除集合中的所有元素。
- del 语句删除整个集合。

```
>>> set1 = {55, -22, 3, 45, 11, 62, 38, 985}
>>> set1
{3, 38, -22, 11, 45, 55, 985, 62}
>>> set1.clear()
>>> set1
set()
```

```
>>> set1 = {55, -22, 3, 45, 11, 62, 38, 985}
>>> set1
{3, 38, -22, 11, 45, 55, 985, 62}
>>> del set1
>>> set1
Traceback (most recent call last):
  File "<pyshell#161>", line 1, in <module>
    set1
NameError: name 'set1' is not defined
```

例子

- **编写程序，有一系列整数，整数之间以空格间隔，其中含有重复的整数，需要去掉重复后，升序排序输出。**
  - **集合中的元素不可重复，因此可以通过集合对数据进行去重。**

```
line = input().split()
numbers = [eval(x) for x in line]
items = sorted(list(set(numbers)))
for item in items:
    print(item, end=' ')
```

通过列表解析，将数据存放在列表中。

将列表转换为集合，去除重复数据，再转换为列表，升序排序。

```
11 22 35 68 97 63 22 68 11
11 22 35 63 68 97
```