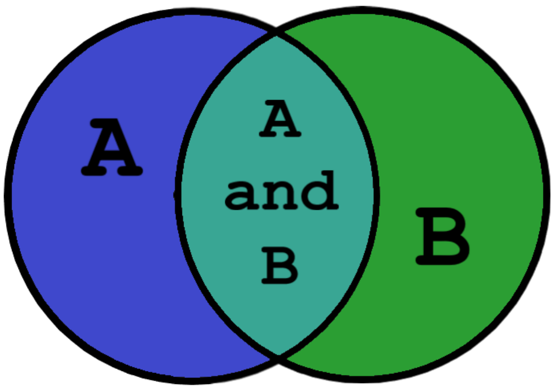
Python Set

https://www.techbeamers.com/python-set/

Python Sets

Sets in Python



定义集合

Python内置的set类型具有以下特点:

```
集是无序的。
集合元素是惟一的。不允许重复元素。
集合本身可以修改，但是集合中包含的元素必须是不可变类型。

x = {<obj>, <obj>, ..., <obj>}
```

```
A = set('qwerty')
print(A)
```

```
{'w', 'y', 'e', 'q', 't', 'r'}
```

```
A = {1, 2, 3}
# A = set('qwerty')
print(A)
```

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

```
A = {1, 2, 3}
B = {3, 2, 3, 1}
print(A == B)
```

```
True
```

```
x = {42, 'foo', 3.14159, None}
print(x)
```

```
{3.14159, None, 42, 'foo'}
```

不要忘记set元素必须是不可变的。例如，一个元组可以包含在一个集合中:

```
x = {42, 'foo', (1, 2, 3), 3.14159}
print(x)
```

```
{3.14159, 42, 'foo', (1, 2, 3)}
```

但是列表和字典是可变的，所以不能设置元素:

```
a = [1, 2, 3]
{a}
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-40-e87edcc78bd7> in <module>()
      1 a = [1, 2, 3]
----> 2 {a}
```

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

```python
d = {'a': 1, 'b': 2}
{d}
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-41-290bfaed1b7f> in <module>()
      1 d = {'a': 1, 'b': 2}
----> 2 {d}
```

```
TypeError: unhashable type: 'dict'
```

```python
# create a set of numbers
py_set_num = {3, 7, 11, 15}
print(py_set_num)

# create a set of mixed data types
py_set_mix = {11, 1.1, "11", (1, 2)}
print(py_set_mix)
```

```
{11, 3, 15, 7}
{'11', 1.1, 11, (1, 2)}
```

```python
# set can't store duplicate elements
py_set_num = {3, 7, 11, 15, 3, 7}
# it'll automatically filter the duplicates
print(py_set_num)

# create a set using the set() method
# creating set with a fixed set of elements
py_set_mix = set([11, 1.1, "11", (1, 2)])
print(py_set_mix)

# creating set with dynamic elements
py_list = [11, 1.1, "11", (1, 2)]
py_list.append(12)
```

```
print(py_list)
py_set_mix = set(py_list)
print(py_set_mix)
```

```
{11, 3, 15, 7}
{'11', 1.1, 11, (1, 2)}
[11, 1.1, '11', (1, 2), 12]
{'11', 1.1, (1, 2), 11, 12}
```

```
# Let's try to create an empty Python set
py_set_num = {}
print("The value of py_set_num:", py_set_num)
print("The type of py_set_num:", type(py_set_num))

py_set_num = set()
print("The value of py_set_num:", py_set_num)
print("The type of py_set_num:", type(py_set_num))
```

```
The value of py_set_num: {}
The type of py_set_num: <class 'dict'>
The value of py_set_num: set()
The type of py_set_num: <class 'set'>
```

```
# Let's try to change a Python set
py_set_num = {77, 88}

try:
    print(py_set_num[0])
except Exception as ex:
    print("Error in py_set_num[0]:", ex)

print("The value of py_set_num:", py_set_num)

# Let's add an element to the set
py_set_num.add(99)
print("The value of py_set_num:", py_set_num)

# Let's add multiple elements to the set
py_set_num.update([44, 55, 66])
print("The value of py_set_num:", py_set_num)

# Let's add a list and a set as elements
py_set_num.update([4.4, 5.5, 6.6], {2.2, 4.4, 6.6})
print("The value of py_set_num:", py_set_num)
```

```
Error in py_set_num[0]: 'set' object does not support indexing
The value of py_set_num: {88, 77}
The value of py_set_num: {88, 99, 77}
The value of py_set_num: {66, 99, 44, 77, 55, 88}
The value of py_set_num: {66, 99, 4.4, 5.5, 6.6, 2.2, 44, 77, 55, 88}
```

```python
# Let's try to use a Python set
py_set_num = {22, 33, 55, 77, 99}

# discard an element from the set
py_set_num.discard(99)
print("py_set_num.discard(99):", py_set_num)

# remove an element from the set
py_set_num.remove(77)
print("py_set_num.remove(77):", py_set_num)

# discard an element not present in the set
py_set_num.discard(44)
print("py_set_num.discard(44):", py_set_num)

# remove an element not present in the set
try:
    py_set_num.remove(44)
except Exception as ex:
    print("py_set_num.remove(44) => KeyError:", ex)
```

```
py_set_num.discard(99): {33, 77, 22, 55}
py_set_num.remove(77): {33, 22, 55}
py_set_num.discard(44): {33, 22, 55}
py_set_num.remove(44) => KeyError: 44
```

```python
# Let's use the following Python set
py_set_num = {22, 33, 55, 77, 99}
print("py_set_num:", py_set_num)

# pop an element from the set
py_set_num.pop()
print("py_set_num.pop():", py_set_num)

# pop one more element from the set
py_set_num.pop()
print("py_set_num.pop():", py_set_num)

# clear all elements from the set
py_set_num.clear()
print("py_set_num.clear():", py_set_num)
```

```
py_set_num: {33, 99, 77, 22, 55}
py_set_num.pop(): {99, 77, 22, 55}
py_set_num.pop(): {77, 22, 55}
py_set_num.clear(): set()
```

```
# We'll use the setA and setB for our illustration
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}
```

添加元素

Python Set add()

```
vowels = {'a', 'e', 'i', 'u'}

# adding 'o'
vowels.add('o')
print('Vowels are:', vowels)

# adding 'a' again
vowels.add('a')
print('Vowels are:', vowels)
```

```
Vowels are: {'e', 'u', 'a', 'i', 'o'}
Vowels are: {'e', 'u', 'a', 'i', 'o'}
```

```
# Python code to demonstrate addition of tuple to a set.
s = {'g', 'e', 'e', 'k', 's'}
t = ('f', 'o')

# adding tuple t to set s.
s.add(t)

print(s)
```

```
{'g', 'e', 'k', 's', ('f', 'o')}
```

```
r = {'f', 'o'}
print(s|r)
```

```
{'g', 'f', 'o', ('f', 'o'), 'e', 'k', 's'}
```

与字典括号区分

```
# initialize a with {}
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
# Output: <class 'set'>
print(type(a))
```

```
<class 'dict'>
<class 'set'>
```

集合运算

```
a = set('abracadabra')
b = set('alacazam')
print(a)
print(b)
```

```
{'d', 'a', 'r', 'b', 'c'}
{'a', 'm', 'l', 'z', 'c'}
```

```
print(a - b)  # 集合a中包含而集合b中不包含的元素
```

```
{'b', 'r', 'd'}
```

```
print(a | b)  # 集合a或b中包含的所有元素
```

```
{'d', 'a', 'm', 'l', 'r', 'z', 'b', 'c'}
```

```
print(a & b)  # 集合a和b中都包含了的元素
```

```
{'c', 'a'}
```

```
print(a ^ b)  # 不同时包含于a和b的元素
```

```
{'r', 'z', 'd', 'b', 'm', 'l'}
```

列表推导式

```
a = {x for x in 'abracadabra' if x not in 'abc'}
print(a)
```

```
{'r', 'd'}
```

移除元素

s.remove(x)

将元素 x 从集合 s 中移除，如果元素不存在，则会发生错误。

```
thisset = set(("Google", "Runoob", "Taobao"))
thisset.remove("Taobao")
print(thisset)
```

```
{'Google', 'Runoob'}
```

```
thisset.remove("Facebook")   # 不存在会发生错误
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-12-2974ebbe1ae6> in <module>()
----> 1 thisset.remove("Facebook")   # 不存在会发生错误
```

```
KeyError: 'Facebook'
```

s.discard(x)

如果元素不存在，不会发生错误

```
thisset = set(("Google", "Runoob", "Taobao"))
thisset.discard("Runoob")
thisset.discard("Facebook")  # 不存在不会发生错误
print(thisset)
```

```
{'Google', 'Taobao'}
```

s.pop()

设置随机删除集合中的一个元素

```
thisset = set(("Google", "Runoob", "Taobao", "Facebook"))
x = thisset.pop()

print(x)
```

```
Google
```

多次执行测试结果都不一样。

然而在交互模式，pop 是删除集合的第一个元素（排序后的集合的第一个元素）

计算集合元素个数

len(s)

```
thisset = set(("Google", "Runoob", "Taobao"))
print(len(thisset))
```

```
3
```

s.clear()

清空集合 s

```
thisset = set(("Google", "Runoob", "Taobao"))
thisset.clear()
print(thisset)
```

```
set()
```

x in s

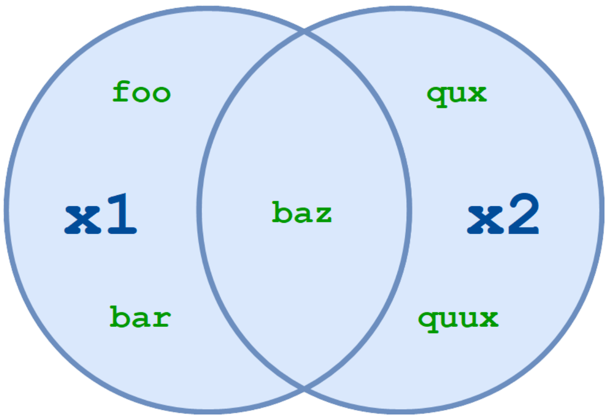
判断元素 x 是否在集合 s 中，存在返回 True，不存在返回 False。

```
thisset = set(("Google", "Runoob", "Taobao"))
print("Runoob" in thisset)

print("Facebook" in thisset)
```

```
True
False
```

并集

```
x1.union(x2[, x3 ...])

x1 | x2 [| x3 ...]
```



```
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}

print(x1.union(x2))
print(x1 | x2)
```

```
{'baz', 'qux', 'quux', 'bar', 'foo'}
{'baz', 'qux', 'quux', 'bar', 'foo'}
```

多个集合取并集

```
a = {1, 2, 3, 4}
b = {2, 3, 4, 5}
c = {3, 4, 5, 6}
d = {4, 5, 6, 7}

print(a.union(b, c, d))
print(a | b | c | d)
```

```
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

```
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
population = people.union(vampires)
print(population)
```

```
{'Jay', 'Arjun', 'Idrish', 'Karan', 'Archil'}
```

```
# We'll use the setA and setB for our illustration
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

print("Initial setA:", setA, "size:", len(setA))
print("Initial setB:", setB, "size:", len(setB))

print("(setA | setB):", setA | setB, "size:", len(setA | setB))
```

```
Initial setA: {'o', 'e', 'u', 'h', 'i', 'g', 'a'} size: 7
Initial setB: {'o', 'b', 'e', 'u', 't', 'a', 'z'} size: 7
(setA | setB): {'z', 'o', 'b', 'e', 'u', 'h', 't', 'i', 'g', 'a'} size: 10
```

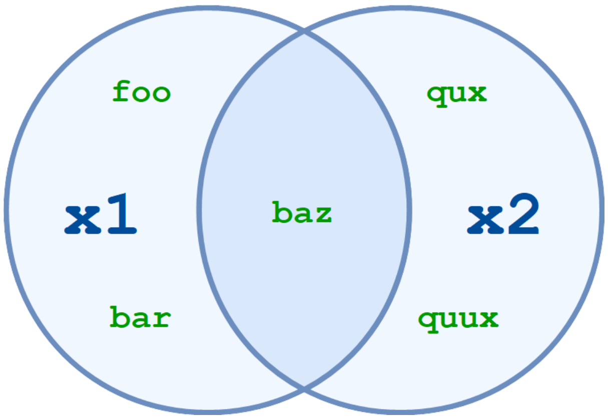
```

```

```
# Python set example using the union() method
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

print("setA.union(setB):", setA.union(setB), "size:", len(setA.union(setB)))
print("setB.union(setA):", setB.union(setA), "size:", len(setB.union(setA)))
```

```
setA.union(setB): {'z', 'o', 'b', 'e', 'u', 'h', 't', 'i', 'g', 'a'} size: 10
setB.union(setA): {'o', 'b', 'e', 'u', 'h', 't', 'i', 'a', 'g', 'z'} size: 10
```

交集 intersection

```
x1.intersection(x2[, x3 ...])

x1 & x2 [& x3 ...]
```



```
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}

print(x1.intersection(x2))
print(x1 & x2)
```

```
{'baz'}
{'baz'}
```

```
a = {1, 2, 3, 4}
b = {2, 3, 4, 5}
c = {3, 4, 5, 6}
d = {4, 5, 6, 7}

print(a.intersection(b, c, d))
print(a & b & c & d)
```

```
{4}
{4}
```

```
# Python intersection example using the & operator
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

print("Initial setA:", setA, "size:", len(setA))
print("Initial setB:", setB, "size:", len(setB))

print("(setA & setB):", setA & setB, "size:", len(setA & setB))
```

```
Initial setA: {'o', 'e', 'u', 'h', 'i', 'g', 'a'} size: 7
Initial setB: {'o', 'b', 'e', 'u', 't', 'a', 'z'} size: 7
(setA & setB): {'o', 'e', 'u', 'a'} size: 4
```

```
# Python set example using the intersection() method
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

intersectAB = setA.intersection(setB)
print("setA.intersection(setB):", intersectAB, "size:", len(intersectAB))
intersectBA = setB.intersection(setA)
print("setB.intersection(setA):", intersectBA, "size:", len(intersectBA))
```

```
setA.intersection(setB): {'o', 'e', 'u', 'a'} size: 4
setB.intersection(setA): {'o', 'e', 'u', 'a'} size: 4
```

差集 difference

```
x1.difference(x2[, x3 ...])

x1 - x2 [- x3 ...]
```

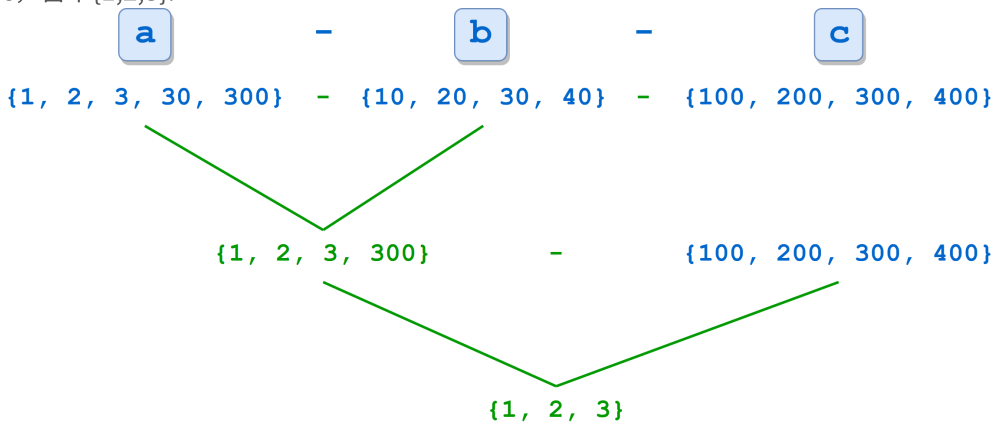
```
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}

print(x1.difference(x2))
print(x1 - x2)
```

```
{'bar', 'foo'}
{'bar', 'foo'}
```

```
a = {1, 2, 3, 30, 300}
b = {10, 20, 30, 40}
c = {100, 200, 300, 400}

print(a.difference(b, c))
print(a - b - c)
```

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

当指定多个集合时，操作从左到右执行。在上面的例子中，首先计算a - b，得到{1,2,3,300}。然后从该集合中减去c，留下{1,2,3}:



```
# Python set's difference operation
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

diffAB = setA - setB
print("diffAB:", diffAB, "size:", len(diffAB))
diffBA = setB - setA
print("diffBA:", diffBA, "size:", len(diffBA))
```

```
diffAB: {'h', 'g', 'i'} size: 3
diffBA: {'t', 'b', 'z'} size: 3
```

```
```

```
# Python set's difference operation using the difference() method
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

diffAB = setA.difference(setB)
print("diffAB:", diffAB, "size:", len(diffAB))
diffBA = setB.difference(setA)
print("diffBA:", diffBA, "size:", len(diffBA))
```

```
diffAB: {'h', 'g', 'i'} size: 3
diffBA: {'t', 'b', 'z'} size: 3
```

交集的补集

```
x1.symmetric_difference(x2)

x1 ^ x2 [^ x3 ...]
```

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}

print(x1.symmetric_difference(x2))
print(x1 ^ x2)
```

```
{'bar', 'qux', 'quux', 'foo'}
{'bar', 'qux', 'quux', 'foo'}
```

```
a = {1, 2, 3, 4, 5}
b = {10, 2, 3, 4, 50}
c = {1, 50, 100}

print(a ^ b ^ c)
```

```
{10, 100, 5}
```

虽然^运算符允许多个集合，但是.symmetric_difference()方法不允许:

```
a = {1, 2, 3, 4, 5}
b = {10, 2, 3, 4, 50}
c = {1, 50, 100}

a.symmetric_difference(b, c)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-53-4c7748c7540f> in <module>()
      3 c = {1, 50, 100}
      4
----> 5 a.symmetric_difference(b, c)
```

```
TypeError: symmetric_difference() takes exactly one argument (2 given)
```

```python
# Python set example using the caret ^ operator
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

symdiffAB = setA^setB
print("symdiffAB:", symdiffAB, "size:", len(symdiffAB))
symdiffBA = setB^setA
print("symdiffBA:", symdiffBA, "size:", len(symdiffBA))
```

```
symdiffAB: {'b', 'h', 't', 'i', 'g', 'z'} size: 6
symdiffBA: {'h', 'b', 't', 'i', 'g', 'z'} size: 6
```

```python
# Python set example using the symmetric_difference() method
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

symdiffAB = setA.symmetric_difference(setB)
print("symdiffAB:", symdiffAB, "size:", len(symdiffAB))
symdiffBA = setB.symmetric_difference(setA)
print("symdiffBA:", symdiffBA, "size:", len(symdiffBA))
```

```
symdiffAB: {'b', 'h', 't', 'i', 'g', 'z'} size: 6
symdiffBA: {'h', 'b', 't', 'i', 'g', 'z'} size: 6
```

```
```

```python
# Python set example to access elements from a set
basket = set(["apple", "mango", "banana", "grapes", "orange"])

for fruit in basket:
    print(fruit)
```

```
orange
grapes
banana
apple
mango
```

```python
# Python set example to test elements in a set
basket = set(["apple", "mango", "banana", "grapes", "orange"])

# confirm if 'apple' is in the basket
print("Is 'apple' in the basket?", 'apple' in basket)

# confirm if 'grapes' is in the basket
print("Is 'watermelon' in the basket?", 'watermelon' in basket)
```

```
Is 'apple' in the basket? True
Is 'watermelon' in the basket? False
```

```python
# Python Sample - Standard vs. Frozen Set

# A standard set
std_set = set(["apple", "mango","orange"])

# Adding an element to normal set is fine
std_set.add("banana")

print("Standard Set:", std_set)

# A frozen set
frozen_set = frozenset(["apple", "mango","orange"])

print("Frozen Set:", frozen_set)

# Below code will raise an error as we are modifying a frozen set
try:
    frozen_set.add("banana")
except Exception as ex:
    print("Error:", ex)
```

```
Standard Set: {'apple', 'orange', 'mango', 'banana'}
Frozen Set: frozenset({'apple', 'orange', 'mango'})
Error: 'frozenset' object has no attribute 'add'
```

x1.isdisjoint (x2)

确定两个集合是否具有任何公共元素。

如果x1和x2没有共同的元素，则isdisjoint(x2)返回True:

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}

print(x1.isdisjoint(x2))
print(x2 - {'baz'})
print(x1.isdisjoint(x2 - {'baz'}))
```

```
False
{'qux', 'quux'}
True
```

```
x1 = {1, 3, 5}
x2 = {2, 4, 6}

print(x1.isdisjoint(x2))

print(x1 & x2)
```

```
True
set()
```

注意:没有对应于.isdisjoint()方法的操作符。

子集判断

```
x1.issubset (x2)
x1 < = x2
```

确定一个集合是否是另一个集合的子集。

在集合理论中，一个集合x1被认为是另一个集合x2的子集，如果x1的每个元素都在x2中。

如果x1是x2的子集，则x1 <= x2返回True:

```
x1 = {'foo', 'bar', 'baz'}
print(x1.issubset({'foo', 'bar', 'baz', 'qux', 'quux'}))

x2 = {'baz', 'qux', 'quux'}
print(x1 <= x2)
```

```
True
False
```

集合被认为是它自身的子集:

```
x = {1, 2, 3, 4, 5}
print(x.issubset(x))

print(x <= x)
```

```
True
True
```

真子集判断

```
x1 < x2
```

确定一个集合是否是另一个集合的真子集。

一个真子集与一个子集是相同的，只是集合不可能是相同的。如果x1的每个元素都在x2中，且x1和x2不相等，则集合x1被认为是另一个集合x2的恰当子集。 如果x1是x2的一个子集，则x1 < x2返回True:

```
x1 = {'foo', 'bar'}
x2 = {'foo', 'bar', 'baz'}
print(x1 < x2)


x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'bar', 'baz'}
print(x1 < x2)
```

```
True
False
```

虽然一个集合被认为是它自身的一个子集，但它不是它自身的一个真子集:

```
x = {1, 2, 3, 4, 5}
print(x <= x)

print(x < x)
```

```
True
False
```

父集

```
x1.issuperset(x2)
x1 >= x2
```

确定一个集合是否是另一个集合的父集。

父集是子集的逆集。一个集合x1被认为是另一个集合x2的父集，如果x1包含x2的每个元素。

如果x1是x2的父集，则x1 >= x2返回True:

```
x1 = {'foo', 'bar', 'baz'}

print(x1.issuperset({'foo', 'bar'}))


x2 = {'baz', 'qux', 'quux'}
print(x1 >= x2)
```

```
True
False
```

集合也被认为是它自身的超集:

```
x = {1, 2, 3, 4, 5}
print(x.issuperset(x))

print(x >= x)
```

```
True
True
```

真父集

```
x1 > x2
```

确定一个集合是否是另一个集合的真父集。

一个真父集与一个父集是相同的，只是集合不可能是相同的。如果x1包含x2的每个元素，且x1和x2不相等，则一个集合x1被认为是另一个集合x2的适当父集。

如果x1是x2的一个适当的父集，则x1 > x2返回True:

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'bar'}
print(x1 > x2)


x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'bar', 'baz'}
print(x1 > x2)
```

```
True
False
```

```
x = {1, 2, 3, 4, 5}
print(x > x)
```

```
False
```

Sets in Python

```
# Python program to demonstrate differences
# between normal and frozen set

# Same as {"a", "b","c"}
normal_set = set(["a", "b","c"])
```

```
# Adding an element to normal set is fine
normal_set.add("d")

print("Normal Set")
print(normal_set)

# A frozen set
frozen_set = frozenset(["e", "f", "g"])

print("Frozen Set")
print(frozen_set)

# Uncommenting below line would cause error as
# we are trying to add element to a frozen set
# frozen_set.add("h")
```

```
Normal Set
{'b', 'c', 'a', 'd'}
Frozen Set
frozenset({'g', 'e', 'f'})
```

add(x) 添加元素

```
people = {"Jay", "Idrish", "Archil"}
people.add("Daxit")

print(people)
```

```
{'Jay', 'Daxit', 'Idrish', 'Archil'}
```

union() 并集

```
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
population = people.union(vampires)
print(population)
```

```
{'Jay', 'Arjun', 'Idrish', 'Karan', 'Archil'}
```

```
population = people|vampires
print(population)
```

```
{'Jay', 'Arjun', 'Idrish', 'Karan', 'Archil'}
```

intersect(s)

```
victims = people.intersection(vampires)

print(victims)
```

```
set()
```

difference(s) 差集

```
safe = people.difference(vampires)

print(safe)
```

```
{'Jay', 'Idrish', 'Archil'}
```

```
safe = people - vampires
print(safe)
```

```
{'Jay', 'Idrish', 'Archil'}
```

计算元音字符个数

```
# Python3 code to count vowel in
# a string using set


# Function to count vowel
def vowel_count(str):

    # Intializing count variable to 0
    count = 0

    # Creating a set of vowels
    vowel = set("aeiouAEIOU")

    # Loop to traverse the alphabet
    # in the given string
    for alphabet in str:

        # If alphabet is present
        # in set vowel
        if alphabet in vowel:
            count = count + 1

    print("No. of vowels :", count)


# Driver code
str = "GeeksforGeeks"

# Function Call
vowel_count(str)
```

```
No. of vowels : 5
```

https://www.geeksforgeeks.org/python-set-pairs-complete-strings-two-sets/

```
# Function to find pairs of complete strings
# in two sets of strings

def completePair(set1,set2):

    # consider all pairs of string from
    # set1 and set2
    count = 0
    for str1 in set1:
        for str2 in set2:
            result = str1 + str2

            # push all alphabets of concatenated
            # string into temporary set
            tmpSet = set([ch for ch in result if (ord(ch)>=ord('a') and ord(ch)<=ord('z'))])
            if len(tmpSet)==26:
                count = count + 1
    print (count)

# Driver program
if __name__ == "__main__":
    set1 = ['abcdefgh', 'geeksforgeeks','lmnopqrst', 'abc']
    set2 = ['ijklmnopqrstuvwxyz', 'abcdefghijklmnopqrstuvwxyz','defghijklmnopqrstuvwxyz']
    completePair(set1,set2)
```

```
7
```

https://www.geeksforgeeks.org/python-remove-discard-sets/

```
# Python program to remove random elements of choice
# Function to remove elements using discard()
def Remove(sets):
    sets.discard(20)
    print (sets)

# Driver Code
sets = set([10, 20, 26, 41, 54, 20])
Remove(sets)
```

```
{41, 10, 54, 26}
```

https://www.geeksforgeeks.org/output-python-programs-set-24-sets/

```
sets = {3, 4, 5}
sets.update([1, 2, 3])
print(sets)
```

```
{1, 2, 3, 4, 5}
```

```
set1 = {1, 2, 3}
set2 = set1.copy()
set2.add(4)
print(set1)
print(set2)
```

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

修改集合

虽然集合中包含的元素必须是不可变的类型，但是集合本身可以修改。与上面的操作类似，也有一些操作符和方法可以用来更改集合的内容。

增广赋值运算符和方法

上面列出的并集、交集、差分和对称差分运算符都有一个可用于修改集合的增广赋值形式。

```
x1.update(x2[, x3 ...])

x1 |= x2 [| x3 ...]
```

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'baz', 'qux'}

x1 |= x2
print(x1)


x1.update(['corge', 'garply'])
print(x1)
```

```
{'baz', 'qux', 'bar', 'foo'}
{'baz', 'qux', 'bar', 'foo', 'corge', 'garply'}
```

```
x1.intersection_update(x2[, x3 ...])

x1 &= x2 [& x3 ...]
```

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'baz', 'qux'}

x1 &= x2
print(x1)


x1.intersection_update(['baz', 'qux'])
print(x1)
```

```
{'baz', 'foo'}
{'baz'}
```

```
x1.difference_update(x2[, x3 ...])

x1 -= x2 [| x3 ...]
```

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'baz', 'qux'}

x1 -= x2
print(x1)


x1.difference_update(['foo', 'bar', 'qux'])
print(x1)
```

```
{'bar'}
set()
```

```
x1.symmetric_difference_update(x2)

x1 ^= x2
```

```
x1 = {'foo', 'bar', 'baz'}
x2 = {'foo', 'baz', 'qux'}

x1 ^= x2
print(x1)


x1.symmetric_difference_update(['qux', 'corge'])
print(x1)
```

```
{'bar', 'qux'}
{'bar', 'corge'}
```

```
x.add(<elem>)
```

```
x = {'foo', 'bar', 'baz'}

x.add('qux')
print(x)
```

```
{'bar', 'baz', 'foo', 'qux'}
```

```
x.remove(<elem>)
```

```python
x = {'foo', 'bar', 'baz'}

x.remove('baz')
print(x)


x.remove('qux')
print(x)
```

```
{'bar', 'foo'}
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-69-9c56678f409b> in <module>()
      5
      6
----> 7 x.remove('qux')
      8 print(x)
```

```
KeyError: 'qux'
```

```
x.discard(<elem>)
```

```python
x = {'foo', 'bar', 'baz'}

x.discard('baz')
print(x)


x.discard('qux')
print(x)
```

```
{'bar', 'foo'}
{'bar', 'foo'}
```

```
x.pop()
```

```
x = {'foo', 'bar', 'baz'}

x.pop()
print(x)

x.pop()
print(x)

x.pop()
print(x)

x.pop()
print(x)
```

```
{'baz', 'foo'}
{'foo'}
set()
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-71-911da97443f7> in <module>()
     10 print(x)
     11
---> 12 x.pop()
     13 print(x)
```

```
KeyError: 'pop from an empty set'
```

Frozen Sets

Python提供了另一种称为Frozen Sets的内置类型，除了Frozen Sets是不可变的之外，它在所有方面都与set完全一样。您可以执行非修改操作对一个Frozen Sets:

```
x = frozenset(['foo', 'bar', 'baz'])
print(x)

print(x)

print(x & {'baz', 'qux', 'quux'})
```

```
frozenset({'bar', 'baz', 'foo'})
frozenset({'bar', 'baz', 'foo'})
frozenset({'baz'})
```

试图修改frozenset会抛出异常

```
x = frozenset(['foo', 'bar', 'baz'])

x.add('qux')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-73-3e04e3ca6b5f> in <module>()
      1 x = frozenset(['foo', 'bar', 'baz'])
      2
----> 3 x.add('qux')
```

```
AttributeError: 'frozenset' object has no attribute 'add'
```

```
x.pop()
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-74-e5f026e5bee7> in <module>()
----> 1 x.pop()
```

```
AttributeError: 'frozenset' object has no attribute 'pop'
```

```
x.clear()
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-75-36236e138dcc> in <module>()
----> 1 x.clear()
```

```
AttributeError: 'frozenset' object has no attribute 'clear'
```

```
print(x)
```

```
frozenset({'bar', 'baz', 'foo'})
```

```
由于frozenset是不可变的，可能认为它不可能是增广赋值操作符的目标。
```

```
f = frozenset(['foo', 'bar', 'baz'])
s = {'baz', 'qux', 'quux'}

f &= s
print(f)
```

```
frozenset({'baz'})
```

Python不会对现有的冻结集执行增广赋值。语句x &= s实际上等价于x = x & s。它没有修改原始的x，而是将x重新分配给一个新对象，而最初引用的对象x已经没有了。 你可以用id()函数来验证:

```
f = frozenset(['foo', 'bar', 'baz'])
print(id(f))
```

```
140418632
```

```
s = {'baz', 'qux', 'quux'}

f &= s
print(f)
```

```
frozenset({'baz'})
```

```
print(id(f))
```

```
137035112
```

在需要使用集合，但需要一个不可变对象的情况下，Frozensets非常有用。例如，你不能定义一个集合，它的元素也是集合，因为集合元素必须是不可变的:

```
x1 = set(['foo'])
x2 = set(['bar'])
x3 = set(['baz'])
x = {x1, x2, x3}
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-82-c8af6ea6ce48> in <module>()
      2 x2 = set(['bar'])
      3 x3 = set(['baz'])
----> 4 x = {x1, x2, x3}
```

```
TypeError: unhashable type: 'set'
```

如果你真的觉得必须定义一组集合，你可以在元素是frozensets的情况下定义，因为它们是不可变的:

```
x1 = frozenset(['foo'])
x2 = frozenset(['bar'])
x3 = frozenset(['baz'])
x = {x1, x2, x3}
print(x)
```

```
{frozenset({'baz'}), frozenset({'foo'}), frozenset({'bar'})}
```

回忆一下前面关于字典的教程，字典键必须是不可变的。你不能使用内置的set类型作为字典键:

```
x = {1, 2, 3}
y = {'a', 'b', 'c'}

d = {x: 'foo', y: 'bar'}
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-84-f6436f7a86ac> in <module>()
      2 y = {'a', 'b', 'c'}
      3 
----> 4 d = {x: 'foo', y: 'bar'}
```

```
TypeError: unhashable type: 'set'
```

如果你发现自己需要使用集作为字典键，你可以使用frozensets:

```
x = frozenset({1, 2, 3})
y = frozenset({'a', 'b', 'c'})

d = {x: 'foo', y: 'bar'}
print(d)
```

```
{frozenset({1, 2, 3}): 'foo', frozenset({'b', 'c', 'a'}): 'bar'}
```