Методы множеств

set.add(a)

Добавляет в множество указанный элемент `a` . Примеры.

```
>>> s = set('abc')
>>> s
{'b', 'a', 'c'}
>>>
>>> s.add('d')
>>> s
{'b', 'a', 'd', 'c'}
```

Добавлять можно только неизменяемые (хешируемые объекты). Если попытаться добавить изменяемый объект, то будет вызвано исключение *TypeError*:

```
>>> # добавляем неизменяемый кортеж и все Ok:
>>> s.add((1, 0))
>>> s
{'b', 'a', (1, 0), 'd', 'c'}
>>>
>>>
>>> # добавляем изменяемый список и все не Ok:
>>> s.add([1, 0])
TypeError: unhashable type: 'list'
```

set.clear()

Очищает множество (удаляет все его элементы). Примеры.

```
>>> s = set('abc')
>>> s
{'b', 'a', 'c'}
>>>
>>> s.clear()
>>> s
set()
```

set.copy()

Возвращает поверхностную копию множества. Примеры.

```
>>> x = set('abc')
>>>
>>> y = x    # y ссылается на x
>>>
>>> z = x.copy()     # ссылается на поверхностную копию x
>>>
>>> x.add('d')     # изменения в x
>>> x
{'b', 'a', 'd', 'c'}
>>>
>>> y     # повлияют на y
{'b', 'a', 'd', 'c'}
>>>
>>> z     # но не повлияют на z
{'b', 'a', 'c'}
```

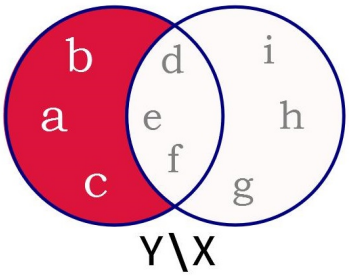
set.difference(x)

> Возвращает элементы множества которые отсутствуют в указанном множестве `x`.
> Примеры.

Данный метод соотвествует разности множеств:

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.difference(X)
{'b', 'a', 'c'}
```

В математике эквивалентная запись данной операции записывается как $Y \setminus X$ и соответствует следующей диаграмме Вена:



Методу `.difference()` соответствует оператор '`-`':

```
>>> Y - X
{'b', 'a', 'c'}
```

Данный метод так же применим и к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y.difference(X)
frozenset({'b', 'a', 'c'})
```

Если в объединении учавствуют два множества типа *set* и *frozenset*, то тип результата будет зависеть от того какое множество указано первым:

```
>>> Y - X
{'b', 'a', 'c'}
>>>
>>> X - Y
frozenset({'h', 'g', 'i'})
```

> `set.`**`difference_update`**`(x)`
>
> Удаляет из множества все элементы, присутствующие в указанном множестве `x`. Примеры.

Данный методо ничего не возвращает и меняет исходное множество:

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.difference_update(X)
>>> Y
{'b', 'a', 'c'}
```

Данному методу соответствует оператор `-=` :

```
>>> Y -= X
>>> Y
{'b', 'a', 'c'}
```

Оператор `-=` может быть применен к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>> Y -= X
>>> Y
frozenset({'b', 'a', 'c'})
```

set.discard(a)

Удаляет указанный элемент `a` , если он присутствует в множестве. Примеры.

Данный методо ничего не возвращает и меняет исходное множество:

```
>>> X = set('abcd')
>>> X
{'b', 'a', 'd', 'c'}
>>>
>>> X.discard('a')
>>> X
{'b', 'd', 'c'}
```

Если указанного элемента нет в множестве, то это не приводит к ошибке:

```
>>> X.discard('z')
>>> X
{'b', 'd', 'c'}
```

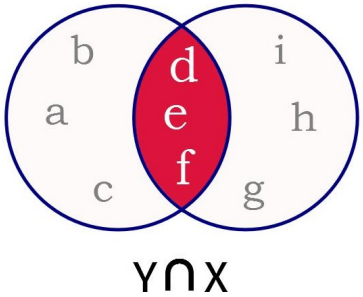
Данный метод схож с методом `.remove()` .

set.intersection(x)

Возвращает множество с элементам присутствующими в обоих множествах. Примеры.

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.intersection(X)
{'f', 'd', 'e'}
```

В математике эквивалентная запись данной операции записывается как $Y \cap X$ и соответствует следующей диаграмме Вена:



Методу `.intersection()` соответствует оператор `'&'`:

```
>>> Y & X
{'f', 'd', 'e'}
```

Данный метод так же применим и к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y.intersection(X)
frozenset({'f', 'd', 'e'})
```

Если в объединении учавствуют два множества типа *set* и *frozenset*, то тип результата будет зависеть от того какое множество указано первым:

```
>>> Y & X
{'f', 'd', 'e'}
>>>
>>> X & Y
frozenset({'f', 'd', 'e'})
```

set.intersection_update(x)

Оставляет в множестве только те элементы которые входят в его пересечение с указанным множеством `x`. Примеры.

Данный методо ничего не возвращает и меняет исходное множество:

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.intersection_update(X)
>>> Y
{'f', 'd', 'e'}
```

Данному методу соответствует оператор `&=`:

```
>>> Y &= X
>>> Y
{'f', 'd', 'e'}
```

Оператор `&=` может быть применен к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y &= X
>>> Y
frozenset({'f', 'd', 'e'})
```

set.isdisjoint(x)

Возвращает *True* если у множества нет общих элементов с указанным множеством `x` . Примеры.

```
>>> Y = set('abc')
>>> X = set('def')
>>>
>>> Y.isdisjoint(X)
True
>>>
>>> Y.add('d')
>>> Y.isdisjoint(X)
False
```

set.issubset(x)

Возвращает *True* если множество эквивалентно указанному множеству `x` или является его подмножеством. Примеры.

```
>>> Y = set('abc')
>>> X = set('abc')
>>>
>>> Y.issubset(X)
True
>>>
>>>
>>> Y = set('ab')
>>> Y.issubset(X)
True
>>>
>>> Y.add('z')
>>> Y.issubset(X)
False
```

Данный метод имет соответствующий оператор сравнения `<=` :

```
>>> Y = set('abc')
>>> X = set('abc')
>>>
>>> Y <= X
True
```

Что бы убедиться только в том что множество является подмножеством используется оператор `<` :

```
>>> Y = set('abc')
>>> X = set('abc')
>>>
>>> Y < X
False
>>>
>>> Z = set('abcd')
>>> Y < Z
True
```

set.issuperset(x)

Возвращает *True* если множество эквивалентно указанному множеству `x` или является его надмножеством. Примеры.

```
>>> Y = set('abc')
>>> X = set('abc')
>>>
>>> Y.issuperset(X)
True
>>>
>>> Y.add('z')
>>> Y.issuperset(X)
True
```

Данный метод имет соответствующий оператор сравнения `>=` :

```
>>> Y = set('abcd')
>>> X = set('adc')
>>>
>>> Y >= X
True
```

Что бы убедиться только в том что множество является подмножеством используется оператор `>` :

```
>>> Y = set('abc')
>>> X = set('adc')
>>>
>>> Y > X
False
>>>
>>> Y.add('d')
>>> Y > X
True
```

`set.`pop`()`

Удаляет и возвращает произвольный элемент множества. Если множество является пустым, то вызывается исключение *KeyError*. Примеры.

```
>>> X = set('abc')
>>>
>>> X.pop()
'b'
>>> X.pop()
'a'
>>> X.pop()
'c'
>>> X.pop()
KeyError: 'pop from an empty set'
```

`set.`remove`()`

Удаляет указанный элемент `x` из множества или вызывает исключение *KeyError* если такой элемент отсутсвует. Примеры.

Данный метод ничего не возвращает и меняет исходное множество:

```
>>> X = set('abc')
>>> X
{'b', 'a', 'c'}
>>>
>>> X.remove('a')
>>> X
{'b', 'c'}
>>>
>>> X.remove('z')
KeyError: 'z'
```

Данный метод схож с методом `.discard()`.

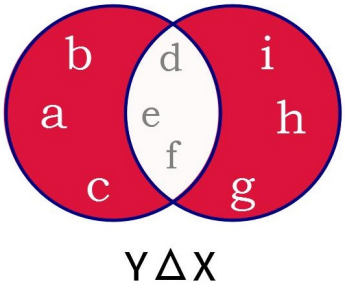
`set.`symmetric_difference`()`

> Возвращает элементы которые присутствуют в обоих множествах, кроме тех, которые являются общими. Примеры.

Данный метод соотвествует симметрической разности множеств (строгой дизъюнкции, исключающему *ИЛИ*):

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.symmetric_difference(X)
{'b', 'h', 'g', 'a', 'i', 'c'}
```

В математике эквивалентная запись данной операции записывается как $Y \triangle X$ и соответствует следующей диаграмме Вена:



Методу `.symmetric_difference()` соответствует оператор `'^'`:

```
>>> Y ^ X
{'b', 'h', 'g', 'a', 'i', 'c'}
```

Данный метод так же применим и к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y ^ X
frozenset({'b', 'h', 'g', 'a', 'i', 'c'})
```

Если в объединении учавствуют два множества типа *set* и *frozenset*, то тип результата будет зависеть от того какое множество указано первым:

```
>>> Y = set('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y ^ X
{'b', 'h', 'g', 'a', 'i', 'c'}
>>>
>>> X ^ Y
frozenset({'b', 'c', 'h', 'g', 'a', 'i'})
```

> `set.`**`symmetric_difference_update`**`(x)`
>
> Добавляет элементы из указанного множества `x` и удаляет элементы которые являются общими. Примеры.

Данный методо ничего не возвращает и меняет исходное множество:

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.symmetric_difference_update(X)
>>> Y
{'b', 'h', 'g', 'a', 'i', 'c'}
```

Данному методу соответствует оператор `^=` :

```
>>> Y = set('abcdef')
>>> X = set('defghi')
>>>
>>> Y ^= X
>>> Y
{'b', 'h', 'g', 'a', 'i', 'c'}
```

Оператор `^=` может быть применен к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

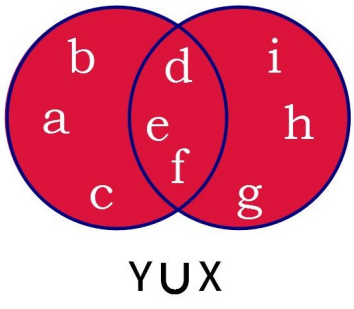
```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y ^= X
>>> Y
frozenset({'b', 'h', 'g', 'a', 'i', 'c'})
```

`set.`**`union`**`(x)`

Возвращает объединение множеств. Примеры.

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.union(X)
{'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'}
```

В математике эквивалентная запись данной операции записывается как $Y \cup X$ и соответствует следующей диаграмме Вена:



Методу `.union()` соответствует оператор `'|'`:

```
>>> Y | X
{'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'}
```

Данный метод так же применим и к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y | X
frozenset({'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'})
```

Если в объединении учавствуют два множества типа *set* и *frozenset*, то тип результата будет зависеть от того какое множество указано первым:

```
>>> Y = set('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y | X
{'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'}
>>>
>>> X | Y
frozenset({'b', 'h', 'g', 'a', 'i', 'd', 'e', 'f', 'c'})
```

`set.update(x)`

Добавляет в множество элементы указанного множества `x` . Примеры.

Данный методо ничего не возвращает и меняет исходное множество:

```
>>> Y = set('abcdef')
>>> Y
{'b', 'a', 'd', 'e', 'f', 'c'}
>>>
>>> X = set('defghi')
>>> X
{'h', 'g', 'd', 'e', 'i', 'f'}
>>>
>>> Y.update(X)
>>> Y
{'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'}
```

Данному методу соответствует оператор `|=` :

```
>>> Y = set('abcdef')
>>> X = set('defghi')
>>>
>>> Y |= X
>>> Y
{'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'}
```

Оператор `|=` может быть применен к фиксированным множествам, но возвращаемый результат будет иметь тип *frozenset*:

```
>>> Y = frozenset('abcdef')
>>> X = frozenset('defghi')
>>>
>>> Y |= X
>>> Y
frozenset({'b', 'h', 'g', 'a', 'd', 'e', 'i', 'f', 'c'})
```