Sets

1. What is a set

Set in Python is a data structure equivalent to sets in mathematics. It may consist of various elements; the order of elements in a set is undefined. You can add and delete elements of a set, you can iterate the elements of the set, you can perform standard operations on sets (union, intersection, difference). Besides that, you can check if an element belongs to a set.

Unlike arrays, where the elements are stored as ordered list, the order of elements in a set is undefined (moreover, the set elements are usually not stored in order of appearance in the set; this allows checking if an element belongs to a set faster than just going through all the elements of the set).

Any immutable data type can be an element of a set: a number, a string, a tuple. Mutable (changeable) data types cannot be elements of the set. In particular, list cannot be an element of a set (but tuple can), and another set cannot be an element of a set. The requirement of immutability follows from the way how do computers represent sets in memory.

2. How to define a set

You can define a set as simple as by naming all of its elements in brackets. The only exception is *empty set*, which can be created using the function `set()`. If `set(..)` has a list, a string or a tuple as a parameter, it will return a set composed of its elements. For example,

```
1
2
3
```

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

will print `{'e', 'q', 'r', 't', 'w', 'y'}` as the output.

The order of elements is unimportant. For example, the program

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

will print `True`, because `A` and `B` are equal sets.

Each element may enter the set only once. `set('Hello')` returns the set of four elements: `{'H', 'e', 'l', 'o'}`.

3. Operations with elements

You can get the number of elements in the set using the function `len`.

You can also iterate over all the elements of the set (in an undefined order!) using the loop `for`:

```
primes = {2, 3, 5, 7, 11}
for num in primes:
    print(num)
```

You can check whether an element belongs to a set using the keyword `in`: expressions like `a in A` return a value of type `bool`. Similarly there's the opposite operation `not in`. To add an element to the set there is the method `add`:

```
A = {1, 2, 3}
print(1 in A, 4 not in A)
A.add(4)
```

There are two methods to remove an element from a set: `discard` and `remove`. Their behavior varies only in case if the deleted item was not in the set. In this case the method `discard` does nothing and the method `remove` throws exception `KeyError`.

.remove(x)

This operation removes element from the set.

If element does not exist, it raises a KeyError.

The *.remove(x)* operation returns None.

Example

>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> s.remove(5)

>>> print(s)

set([1, 2, 3, 4, 6, 7, 8, 9])

>>> print s.remove(4)

None

>>> print s

set([1, 2, 3, 6, 7, 8, 9])

>>> s.remove(0)

KeyError: 0

.discard(x)

This operation also removes element from the set.

If element does not exist, it **does not** raise a KeyError.

The *.discard(x)* operation returns None.

Example

```
>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s.discard(5)
>>> print s
set([1, 2, 3, 4, 6, 7, 8, 9])
>>> print s.discard(4)
None
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
>>> s.discard(0)
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
```

Finally, `pop` removes one random element from the set and returns its value. If the set is empty, `pop` generates the exception `KeyError`.

.pop()

This operation removes and return an arbitrary element from the set.
If there are no elements to remove, it raises a KeyError.

Example

```
>>> s = set([1])
>>> print s.pop()
1
>>> print s
set([])
>>> print s.pop()
KeyError: pop from an empty set
```

You can transform a set to list using the function `list`.

```
my_set = {'a', 'for', 'apple'}

s = list(my_set)
print(s)
['a', 'for', 'apple']
```

4. Operations on sets

This is how you perform the well-known operations on sets in Python:

A | B **A.union(B)**	Returns a set which is the union of sets `A` and `B`.
A |= B **A.update(B)**	Adds all elements of array `B` to the set `A`.
A & B **A.intersection(B)**	Returns a set which is the intersection of sets `A` and `B`.
A &= B **A.intersection_update(B)**	Leaves in the set `A` only items that belong to the set `B`.
A - B **A.difference(B)**	Returns the set difference of `A` and `B` (the elements included in `A`, but not included in `B`).
A -= B **A.difference_update(B)**	Removes all elements of `B` from the set `A`.
A ^ B **A.symmetric_difference(B)**	Returns the symmetric difference of sets `A` and `B` (the elements belonging to either `A` or `B`, but not to both sets simultaneously).
A ^= B **A.symmetric_difference_update(B)**	Writes in `A` the symmetric difference of sets `A` and `B`.
A <= B **A.issubset(B)**	Returns `true` if `A` is a subset of `B`.
A >= B **A.issuperset(B)**	Returns `true` if `B` is a subset of `A`.
A < B	Equivalent to `A <= B and A != B`
A > B	Equivalent to `A >= B and A != B`