07. 집합

집합은 여러 값을 순서 없이 그리고 중복 없이 모아 놓은 자료형이다. 파이썬에서는 set과 frozenset 두 가지 집합 자료형을 제공한다. set은 변경 가능한 집합이고 frozenset은 변경 불가능한 집합이다.

7.1 set객체의 생성

set 객체를 생성하는 방법을 살펴보자.

```
>>> a = set()                    # 빈 set 객체를 생성한다.
>>> b = {1, 2, 3}                # 중괄호{}를 이용하여 객체를 생성한다.
>>> a                            # 공집합인 경우의 출력이다.
set()
>>> b                            # 공집합이 아닌 경우의 출력이다.
{1, 2, 3}
>>> type(a)                      # 자료형을 확인한다.
<class 'set'>
>>> type(a) == type(b)           # 동일 자료형인지 학인한다.
True
>>> b = a.copy()                 # copy() 함수를 사용한 객체 생성이다. a 복사
```

반복 가능한 객체로부터 집합을 만들수도 있다.

```
>>> set((1, 2, 3))# 튜플로부터 집합을 만든다.
{1, 2, 3}
>>> set('abcd')         # 문자열로부터 집합을 만든다.
{'d', 'b', 'a', 'c'}
>>> set([1, 2, 3])      # 리스트로부터 집합을 만든다.
{1, 2, 3}
>>> set((1,2,3,1,2,3,1,2,3))    # 중복된 원소는 한번만 표현한다.
{1, 2, 3}
>>> set({'one':1, 'two':2})            # 사전의 반복자는 키 값을 반환한다.
{'two', 'one'}
```

하지만, 모든 데이터가 집합의 원소로 사용할수 있는 것은 아니다. 해시가능이면서 변경 불가능한 자료형만이 집합의 원소로 사용할 수 있다.

```
>>> a = [1, 2, 3]
>>> b = [3, 4, 5]
>>> {a, b}              # 리스트는 집합의 원소로 사용할수 없다.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

7.2 set 객체의 연산

set 객체에 원소를 추가하는 메서드로는 add()와 update()가 있다. add() 메서드는 한 원소를 추가하고, update() 메서드는 주어진 객체에 대해 합집합 연산을 한다. copy() 메서드를 사용하면 set객체를 통째로 복사할수 있다.

```
>>> a = {1, 2, 3}
>>> len(a)              # 원소의 개수를 센다.
3
>>> a.add(4)            # 한 원소를 추가한다.
>>> a
{1, 2, 3, 4}
>>> a.update([4, 5, 6]) # a = a u {4, 5, 6}
>>> a
{1, 2, 3, 4, 5, 6}
>>> b = {6, 7, 8}       # a = a U b
>>> a
{1, 2, 3, 4, 5, 6}
>>> a = {1, 2, 3}
>>> a.update({4, 5, 6}, {7, 8, 9})      # 두 개 이상의 인수를 지정할수 있다.
>>> a.copy()                    # 집합 a를 복사한다.   한다.
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- 원소 제거

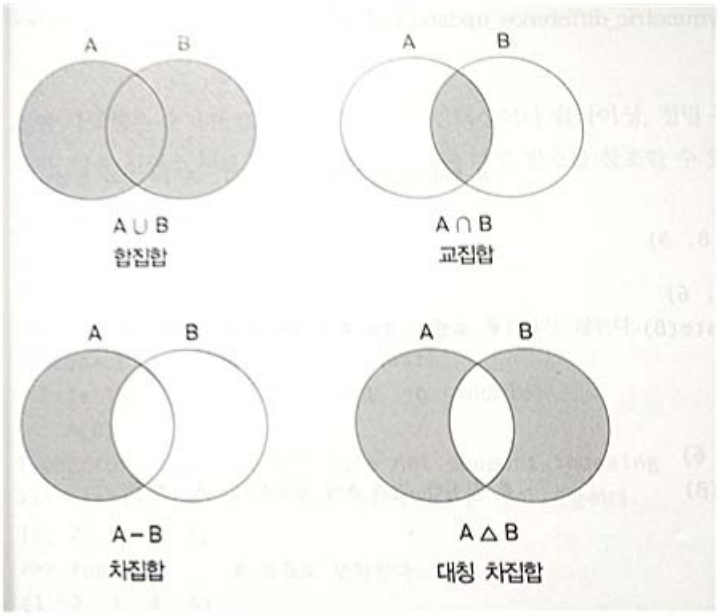
set객체에서 원소를 제거하는 메서드로는 clear()와 discard(), remove() 등이 있다.

```
>>> a = {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> a.clear()                   # 전체 원소를 제거한다.
>>> a
set()
>>> a = {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> a.discard(3)                # 원소 한 개를 제거한다.
>>> a
{1, 2, 4, 5, 6, 7, 8, 9}
>>> a.discard(3)                # 3이 없으면 그냥 통과한다
>>> a.remove(4)                 # 원소 한 개를 제거한다.
>>> a
{1, 2, 5, 6, 7, 8, 9}
>>> a.remove(4)         # 없으면 예외가 발생한다. discard() 메서드와의 차이점이다.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
>>> a.pop()             # 원소 한 개를 집합에서 제거하면서 반환한다.
```

```
1
>>> a.pop()
2
>>> a
{5, 6, 7, 8, 9}
```

- 집합 연산

일반적인 집합 연산으로는 union(합 집합), intersection(교집합), difference(차집합), symmetric_difference(대칭 차집합)이 있다.




```
>>> A = {1, 2, 3, 4, 5, 6}
>>> B = {4, 5, 6, 7, 8 , 9}
>>> C = {4, 10}
>>> A.union(B)  # 합집합 A | B와 동일하다.
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> A.intersection(B)          # 교집합 A & B
{4, 5, 6}
>>> A.intersection(B, C)              # 인수가 두 개 이상이어도 된다.
{4}
>>> A.difference(B)                   # 차집합 A - B
{1, 2, 3}
>>> A.symmetric_difference(B)         # 대칭 차집합 A ^ B
{1, 2, 3, 7, 8, 9}
```

연산 결과가 첫 인수 집합에 반영되기를 바라면 update() 와 intersection_update(), difference_update(), symmetric_difference_update() 메서드를 사용한다.

```
>>> A
{1, 2, 3, 4, 5, 6}
>>> A.update(B)                 # 합집합 결과 A에 저장한다. A |= B와 동일
>>> A
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>>
>>> A = {1, 2, 3, 4, 5, 6}
>>> A.intersection_update(B)    # 교집합 결과 A에 저장한다. A &= B
>>> A
{4, 5, 6}
>>>
>>> A = {1, 2, 3, 4, 5, 6}
>>> A.difference_update(B)      # 차집합 결과 A에 저장한다. A -= B
>>> A
{1, 2, 3}
>>>
>>> A = {1, 2, 3, 4, 5, 6}
>>> A.symmetric_difference_update(B)          # 대칭 차집합 결과 A에 저장한다. A ^= B
>>> A
{1, 2, 3, 7, 8, 9}
```

다음은 원소나 집합의 포함 관계를 시험하는 예이다.

```
>>> A
{1, 2, 3, 7, 8, 9}
>>> 2 in A                      # 2∈A 멤버 검사
True
>>> 2 not in A                  # 2∉A
False
>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> A.issuperset(B)             # A⊃B
True
>>> B.issubset(A)               # B⊂A
True
>>> A.isdisjoint(B)             # 교집합이 공집합인가?
False
```

집합 자료형은 순서가 없는 자료형이르모 인덱싱이나 슬라이싱, 정렬 등을 지원하지 않는다. 하지만, 다른 시퀀스 자료형으로 형변환을 하면 부분 원소를 참조할수 있다.

```
>>> A
```

```
{1, 2, 3, 4, 5}
>>> A[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> list(A)                    # 리스트로 변환하면 모든 것이 가능하다.
[1, 2, 3, 4, 5]
>>> tuple(A)                   # 튜플로 변환한다.
(1, 2, 3, 4, 5)
```

하지만, for 문에서는 직접 사용하는 것이 가능하다.

```
>>> A
{1, 2, 3, 4, 5}
>>> for ele in A:
...     print(ele, end =  '')
...
12345
```

7.3 frozenset 객체의 생성과 연산

frozenset 객체는 변경 가능하지 않은 집합 자료형이다. 값을 변경하지 않는 범위에서 set 객체와 동일하게 동작한다. frozenset 객체의 생성은 집합을 포함한 반복 가능한 자료형으로부터 가능하다.

```
>>> frozenset([1, 2, 3, 4, 5])          # 반복이 가능한 객체로부터 생성한다.
frozenset({1, 2, 3, 4, 5})
```

frozenset 객체는 값을 변경하지 않는 연산만 허용하며 set 객체와 동일하게 동작한다.

```
>>> A = frozenset((1, 2, 3, 4, 5, 6))
>>> B = frozenset((4, 5, 6, 7, 8, 9))
>>> A.union(B)                          # 합집합 A | B
frozenset({1, 2, 3, 4, 5, 6, 7, 8, 9})
>>> A.intersection(B)                   # 교집합 A & B
frozenset({4, 5, 6})
>>> A.difference(B)                     # 차집합 A - B      B
frozenset({1, 2, 3})
>>> A.symmetric_difference(B)           # 대칭 차집합 A - B
frozenset({1, 2, 3, 7, 8, 9})
>>> A.copy()                            # 복사
frozenset({1, 2, 3, 4, 5, 6})
>>>
>>> A = frozenset({1, 2, 3, 4, 5})
>>> B = frozenset({1, 2, 3})
>>> A.issuperset(B)                     # A⊃B
True
>>> B.issubset(A)                       # B⊂A
True
>>> A.isdisjoint(B)                     # 두집합의 교집합이 공집합인가?
False
```

7.4 집합 내장

중괄호 {}를 이용하면 리스트 내장과 같이 for 문을 통해서 집합을 직접 만들 수 있다.

```
>>> {v * v for v in [1, 2, 3, 4]}       # 연산 결과가 set객체로 모인다.합니다.
{16, 1, 4, 9}
>>> {v for v in 'python' if v not in 'aeiou'}   # 모음이 아닌 알파벳 집합니다.
{'n', 'h', 't', 'y', 'p'}
```