집합 자료형

- **순서 없이**(unordered) 0개 이상의 객체를 참조하는 자료형을 말한다.
- 순회할 경우 집합형은 **임의의 순서**대로 객체를 꺼내온다.
  - 집합형은 매핑형처럼 키:매핑값의 자료구조가 아닌 단순히 키(key)만 있는 자료구조라고 생각하면 이해가 쉬울 것이다.
- 순서가 없이 참조하기 때문에 **중복을 허락하지 않는다**.
  - 집합형에 속한 자료형은 고유한 객체 값만 담는다.
- 따라서, 해시가능한 객체만 담을 수 있다.
- 해시가능한(hashable) 객체란?
  - 그 값이 불변성(immutable)인 자료형으로 **int**, **float**, **str**, **tuple**, **frosenset** 등이 있다.
- 해시가능하지 않는 자료형은 집합형에 담을 수 없는데 **list**, **dict**, **set**이 여기에 속한다.
- 집합형 종류
  - 세트(**set**) : 가변자료형(mutable)
  - 프로즌 세트(**frozenset**, 고정집합) : 불변자료형(immutable)

세트(set)

- **가변자료형**(mutable)이다.
  - 생성한 후 내용의 변경이 가능하다. 즉, 담고 있는 객체를 삭제, 변경, 삽입하는 것이 가능하다.
- **순서 없이**(unordered) 0개 이상의 해시가능한 객체를 참조하는 **집합형**이다.
- 각 객체는 쉼표(,)로 구분한다.
- 순서가 없이 참조하기 때문에 중복을 허락하지 않는다.
- 해시가능한 객체(불변자료형)만 담을 수 있다.
- 인덱스의 개념이 없기 때문에 분할(슬라이싱)하거나 구간 이동을 할 수 없다.
- 출력 형식
  - 세트는 항상 중괄호(**{ }**) 형태로 출력한다.

세트 생성

세트를 만드는 방법으로는 다음 세 가지가 있다.

- **{ }**(중괄호)
  - 단, 빈 세트는 중괄호(**{ }**)로 생성할 수 없다.
  - 빈 중괄호(**{ }**)는 빈 딕셔너리를 생성할 때 사용하기 때문이다.
- **set()** 생성자(클래스)
  - 빈 세트는 **set()** 생성자로만 만들 수 있다.
- 세트 축약(set comprehension)

중괄호 **{ }**를 사용한 세트 생성

- 각 객체는 쉼표(,)로 구분한다.

In [ ]:

```
set1 = {'a', 'b', 3}
print(set1)
```

순서가 없이 참조하기 때문에 중복을 허락하지 않는다.

In [ ]:

```
set2 = {'a', 'b', 3, 'a', 3}
print(set2)
```

In [ ]:

```
# 세트가 튜플을 담고 있다.
set3 = {True, 5, '드럼', ('x', 'y'), -3.14, None}
print(set3)
```

In [ ]:

```
len(set3)
```

복합자료형은 어떤 자료형도 담을 수 있지만,

세트는 가변자료형인 리스트와 딕셔너리는 담을 수 없다.

In [ ]:

```
# 세트가 리스트를 담고 있다.
set4 = {True, 5, '드럼', ['x', 'y'], -3.14, None}
```

In [ ]:

```
# 세트가 딕셔너리를 담고 있다.
set5 = {True, 5, {'drum': '드럼'}, -3.14, None}
```

In [ ]:

```
# 세트에 속한 튜플이 리스트를 담고 있다.
set6 = {True, 5, '드럼', ('x', 'y', [1, 2, 3]), -3.14, None}
```

이번에는 **set()** 생성자를 사용한 세트 생성.

- 빈 세트를 만드는 유일한 방법은 **set()** 생성자를 사용하는 것이다.
- **[주의]** 빈 중괄호(**{ }**)는 빈 딕 셔너리를 생성할 때 사용하기 때문에 빈 세트를 만들려면 **set()** 생성자를 사용해야만 한다.

In [ ]:

```
# 빈 세트를 초기화한다.
empty_set = set()
```

In [ ]:

```
# 자료형은 세트다.
type(empty_set)
```

In [ ]:

```
# 비워져 있기 때문에 set()로 출력된다.
# [참고] {}는 빈 딕셔너리를 뜻한다.
print(empty_set)
```

In [ ]:

```
set7 = set('abcde')
print(set7)
```

In [ ]:

```
set8 = set('바나나')
print(set8)
```

set comprehension 을 사용한 세트 생성.

In [1]:

```
list1 = [1, 2, 3, 4, 5]
set9 = {x**2 for x in list1}
print(set9)
```

```
{1, 4, 9, 16, 25}
```

형변환

- 세트는 리스트나 튜플로 형변환이 가능하지만 딕셔너리로 형변환 할 수는 없다.
- 리스트, 튜플, 딕셔너리는 세트로 형변환이 가능하다.
  - 단, 딕셔너리의 경우 키만 세트로 현변환한다.

세트는 리스트나 튜플로 형변환이 가능하다.

In [ ]:

```
s1 = {1, 2, 3}

# 세트를 튜플로 형변환한다.
t1 = tuple(s1)

# 세트를 리스트로 형변환한다.
L1 = list(s1)
```

In [ ]:

```
print(t1)  # 튜플이다.
print(L1)  # 리스트다.
```

리스트나 **튜플**도 세트로 형변환이 가능하다.

In [ ]:

```
# 튜플 (1, 2, 3)을 다시 세트로 형변환한다.
s2 = set(t1)

# 리스트 [1, 2, 3]을 다시 세트로 형변환한다.
s3 = set(L1)
```

In [ ]:

```
print(s2)  # 세트다.
print(s3)  # 세트다.
```

딕셔너리도 세트로 형변환이 가능하다.

이 경우 딕셔너리의 키만 세트의 객체로 형변환 된다.

In [2]:

```
d1 = {(10, 9): '한글날', 0: 55, '악기': ['드럼', '기타', '베이스']}

# 딕셔너리를 세트로 형변환한다.
s4 = set(d1)
```

In [3]:

```
print(s4)  # 세트다.
```

```
{0, '악기', (10, 9)}
```

리스트나 튜플이 **가변자료형**을 담고 있으면 세트로 형변환할 수 없다.

In [ ]:

```
# 튜플이 가변자료형인 리스트를 담고 있다.
t2 = 'a', (1, 2, [3, 4])
```

In [ ]:

```
# 세트로 형변환하면 오류가 난다.
s5 = set(t2)
```

In [ ]:

```
# 리스트가 가변자료형인 딕셔너리를 담고 있다.
L2 = [1, 2, 3, {'color': 'green'}]
```

In [ ]:

```
# 세트로 형변환하면 오류가 난다.
s6 = set(L2)
```

세트 관련 연산자

멤버십 연산자 : **in/not in**

진부분집합인지 확인하는 비교 연산자 : **<**, **>**

부분집합인지 확인하는 비교 연산자 : **<=**, **>=**

세트 결합(합집합) 연산자 : **|**

세트 교차(교집합) 연산자 : **&**

세트 빼기(차집합) 연산자 : **-**

세트 대칭차(대칭차집합) 연산자 : **^**

In [5]:

```
x = {1, 2, 3, 4, 5}
y = {3, 4, 5, 6, 7}
z = {4, 5}
```

멤버십 연산자, 비교 연산자

In [6]:

```
print(4 in x)
print(7 in x)
print(x < x)
print(x <= x)
print(z < x)
```

```
True
False
False
True
True
```

세트 연산 관련 연산자

세트 결합 연산자 **|** 는 수학의 합집합(union)과 같다. 따라서 두 세트의 객체를 모두 합한 세트를 반환한다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# 세트 결합 : 합집합(union)
x | y
```

세트 교차 연산자 **&** 는 수학의 교집합(intersection)과 같기 때문에 두 세트에 동시에 속하는 객체로 구성한 세트를 반환된다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# 세트 교차 : 교집합(intersection)
x & y
```

세트 빼기 연산자 **-** 는 수학의 차집합(difference)과 같다.

차집합 *X* - *Y*는

- *X*에는 속하지만,
- *Y*에는 속하지 않는 객체로 구성된 집합이며,
- 이를 *X*에 대한 *Y*의 차집합이라고 한다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# x에 대한 y의 차집합
x - y
```

*X*와 *Y*의 위치가 바뀌면 다른 결과를 가져올 수 있다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# y에 대한 x의 차집합
y - x
```

세트 대칭차 연산자 **^** 를 수학에서는 대칭차집합(symmetric difference)이라고 한다.

이는 두 집합의 상대 여집합의 합을 말한다.

다시 말해 *X*와 *Y* 두 집합이 있을 때

- 두 집합 모두의 차집합인
- *X* - *Y*와
- *Y* - *X*의
- 합집합을 반환한다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# 세트 대칭차 : 대칭차집합은 두 집합의 상대 여집합의 합(symmetric difference)이다.
x ^ y
```

*X*와 *Y*의 순서와 관계없이 결괏값은 같다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# --- y = {3, 4, 5, 6, 7}
# 세트 대칭차 : 대칭차집합은 두 집합의 상대 여집합의 합(symmetric difference)이다.
y ^ x
```

세트 관련 메소드

복사 메소드 : **copy()**

질의 메소드 : **issubset()**, **issuperset()**, **isdisjoint()**

세트 연산 메소드:

- **union()**,
- **intersection()**, **intersection_update()**,
- **difference()**, **difference_update()**,
- **symmetric_difference()**, **symmetric_difference_update()**

추가 메소드 : **add()**

삭제 메소드 : **pop()**, **remove()**, **discard()**, **clear()**

추가 메소드

In [ ]:

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

s.**add**(***x***)

- 객체 ***x***가 세트 *s*에 없다면 ***x***를 세트 *s*에 추가한다.

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# 같은 객체가 있으면 추가 하지 않는다.
x.add(5)
print(x)
```

In [ ]:

```
# --- x = {1, 2, 3, 4, 5}
# 'a'를 추가한다.
x.add('a')
print(x)
```

삭제 메소드

s.**pop()**

- 세트 *s*에서 임의(random) 객체를 반환하고 해당 객체를 세트 *s*에서 삭제한다.
- 세트 *s*가 비어 있으면 **KeyError**가 발생한다.

In [ ]:

```
# --- {1, 2, 3, 4, 5, 'a'}
# 임의(random) 객체를 반환하고 해당 객체를 세트에서 삭제한다.
x.pop()
print(x)
```

s.**remove**(***x***)

- 객체 ***x***를 세트 *s*에서 삭제한다.
- 객체 ***x***가 세트 *s* 안에 없으면 **KeyError**가 발생한다.
  - **discard()** 참고.

In [ ]:

```
# 4를 삭제한다.
x.remove(4)
```

In [ ]:

```
# 4가 삭제되었다.
print(x)
```

s.**discard**(***x***)

- 객체 ***x***를 세트 *s*에서 삭제한다.
- 객체 ***x***가 세트 *s* 안에 없어도 오류 메시지를 출력하지 않는다.
  - **remove()** 참고.

In [ ]:

```
# 'a'를 삭제한다.
x.discard('a')
```

In [ ]:

```
# 'a'가 삭제되었다.
print(x)
```

In [ ]:

```
# 방금 삭제한 'a'를 다시 삭제한다. 오류가 발생하지 않는다.
x.discard('a')
```

In [ ]:

```
# 방금 제거한 'a'를 다시 삭제한다. 오류가 발생한다.
x.remove('a')
```

s.**clear()**

- 세트 *s*의 모든 객체를 삭제한다.

In [ ]:

```
# 세트의 내용을 확인한다.
print(x)

# 세트의 모든 객체를 삭제한다.
x.clear()

# 세트의 내용을 확인한다.
print(x)
```