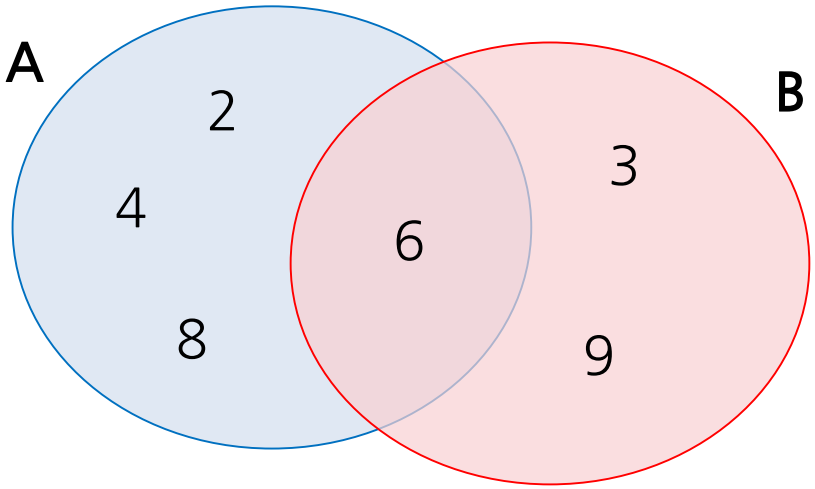
자료구조

집합

집합 (Set)

- 집합
  - 수학에서의 기본적인 집합의 개념과 동일하게, 순서가 없고 서로 다른 원소(항목)들이 모여 있는 자료 구조이다.
  - 집합의 원소들은 모두 서로 다른 값으로서 동일한 값이 여러 개 존재할 수 없으며, 그들 간의 순서에 따른 구분이 없다.
  - 집합 내에서 원소들 간의 산술 연산이 정의되지 않는다.



집합 (Set)

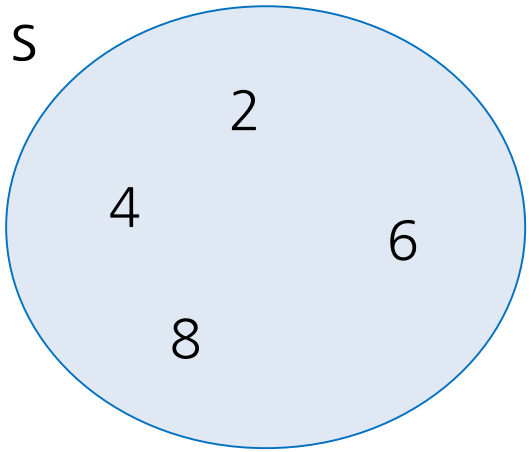
- 집합 만들기

  집합의 이름 = { **값1**, **값2**, ... }

  – 집합은 여러 원소를 쉼표 ‘,’ 기호로 구분하여 기재하고 그 원소들의 양 쪽 끝을 중괄호 ‘{ }’ 기호로 씌워서 만든다.

  – 괄호의 형태가 사전과 동일하지만, 안에 들어 가는 항목들은 키-값의 쌍이 아니라 하나의 값이다.

s = { 2, 4, 6, 8 }



집합 (Set)

- 집합 만들기

  집합의 이름 = { 값1, 값2, ... }

```
1  s1 = {1, 2, 3}
2
3  s2 = {2.5, 5, 7.5, 10}
4
5  s3 = {"a", "b", "c"}
6
7  s4 = set([1, 2, 3, 2, 3, -1, 2])
8
9  s5 = set("Hello")
10
11 s6 = set()
```

```
1  print(s1)
2  print(s2)
3  print(s3)
4  print(s4)
5  print(s5)
6  print(s6)
```

```
{1, 2, 3}
{2.5, 10, 5, 7.5}
{'b', 'a', 'c'}
{1, 2, 3, -1}
{'l', 'e', 'H', 'o'}
set()
```

※ 어떠한 자료 내에서 중복된 값을 제거하고자 할 때, 명령어 set을 이용하여 간단하게 처리할 수 있다.

집합 (Set)

- 집합 내의 1개의 원소 접근하기
  - 집합의 원소들 간에는 순서가 존재하지 않기 때문에 인덱싱을 할 수 없고, 사전처럼 키가 있는 것도 아니므로 키를 이용하여 값에 접근할 수도 없다.
  - 따라서 집합의 특정 원소에 접근하려면 그 집합을 리스트 또는 튜플로 변환한 뒤 인덱싱한다.

```
1  s1 = {1, 2, 3}
2
3  print(s1[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback
<ipython-input-13-15829da90971> in <module>
      1 s1 = {1, 2, 3}
      2
----> 3 print(s1[0])

TypeError: 'set' object does not support indexing
```

```
1  s1 = {1, 2, 3}
2
3  t1 = list(s1)
4
5  print(t1[0])
```

```
1
```

집합 (Set)

- 집합에 새로운 원소 1개 추가하기

  집합의 이름.add(새로운 원소)

  – 대상 집합에 1개의 원소를 추가한다.

```
1  s1 = {1, 2, 3}
2
3  s1.add(4)
4
5  print(s1)
```

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

```
1  a = 123
2
3  s2.add(a)
4
5  print(s2)
```

```
{2.5, 5, 7.5, 10, 'python', 123}
```

```
1  s2 = {2.5, 5, 7.5, 10}
2
3  s2.add("python")
4
5  print(s2)
```

```
{2.5, 5, 7.5, 10, 'python'}
```

```
1  b = True
2
3  s2.add(b)
4
5  print(s2)
```

```
{True, 2.5, 5, 7.5, 10, 'python', 123}
```

집합 (Set)

- 집합에 새로운 원소 여러 개 추가하기

  집합의 이름.update(1개의 리스트 또는 튜플 또는 문자열)

  – 대상 집합에 리스트 또는 튜플에 들어 있는 항목들을 추가하거나, 문자열의 각 문자들을 추가한다.

```
1  s1 = {1, 2, 3}
2
3  s1.update([2, 4, 6])
4
5  print(s1)
```

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

```
1  c = "how are you?"
2
3  s1.update(c)
4
5  print(s1)
```

```
{1, 2, 3, 4, 'e', 6, 'o', 'y', 'h', 'w', ' ', 'a', 'r', 'u', '?'}
```

집합 (Set)

- 집합에서 원소 삭제하기

  집합의 이름.remove(삭제하려는 원소)

  – 대상 집합에서 특정 원소를 제거한다.

```
1  print(s1)
2
3  s1.remove(4)
4
5  print(s1)
```

```
{1, 2, 3, 4, 'e', 6, 'o', 'y', 'h', 'w', ' ', 'a', 'r', 'u', '?'}
{1, 2, 3, 'e', 6, 'o', 'y', 'h', 'w', ' ', 'a', 'r', 'u', '?'}
```

```
1  s1.remove("?")
2
3  print(s1)
```

```
{1, 2, 3, 'e', 6, 'o', 'y', 'h', 'w', ' ', 'a', 'r', 'u'}
```

※ 존재하지 않는 원소를 삭제하려고 하면 오류가 발생한다.

집합 (Set)

- 부분 집합 (Subset) 확인하기

  집합의 이름.issubset(확인하려는 다른 집합)

  – 대상 집합이 다른 집합의 부분 집합인지 검사하여, 부분 집합이 맞으면 결과로 참(True)을 되돌려 준다.

```
1  a = {1, 2}
2  b = {1, 2, 3}
3  c = {2}
4
5  print(a.issubset(b))
6
7  print(a.issubset(c))
8
9  print(c.issubset(a))
```

```
True
False
True
```

집합 (Set)

- 상위 집합 (Superset) 확인하기

  집합의 이름.issuperset(확인하려는 다른 집합)

  – 대상 집합이 다른 집합의 상위 집합인지 검사하여, 상위 집합이 맞으면 결과로 참(True)을 되돌려 준다.

```
1  a = {1, 2}
2  b = {1, 2, 3}
3  c = {2}
4
5  print(b.issuperset(a))
6
7  print(a.issuperset(c))
8
9  print(c.issuperset(a))
```

```
True
True
False
```

집합 (Set)

- 집합에 특정 원소가 존재하는지 확인하기

  찾으려는 원소 in **집합의 이름**

  – 집합에 특정 원소가 있는지 검사하여, 원소가 그 집합에 들어 있다면 결과로 참(True)을 되돌려 준다.

```
1 in a
```

```
True
```

```
3 in b
```

```
True
```

```
"1" in a
```

```
False
```

```
3 in c
```

```
False
```

  찾으려는 원소 not in **집합의 이름**

  – 집합에 특정 원소가 있는지 검사하여, 원소가 그 사전에 들어 있다면 결과로 거짓(False)을 되돌려 준다.

집합 (Set)

- 교집합 (Intersection) 구하기

  ① **집합의 이름.intersection(다른 집합의 이름)**

  – 대상 집합과 다른 집합의 교집합을 구한다.

```
1  a = {1, 2, 3, 4, 5}
2  b = {2, 4, 6, 8}
3
4  a.intersection(b)
```

```
{2, 4}
```

```
1  b.intersection(a)
```

```
{2, 4}
```

```
1  a & b
```

```
{2, 4}
```

  ② **집합의 이름 & 다른 집합의 이름**

  – 또는, '&' 기호로 연결하여 교집합을 구할 수도 있다.

집합 (Set)

- 합집합 (Union) 구하기

① **집합의 이름.union(다른 집합의 이름)**

– 대상 집합과 다른 집합의 합집합을 구한다.

```
1  a = {1, 2, 3, 4, 5}
2  b = {2, 4, 6, 8}
3
4  a.union(b)
```

```
{1, 2, 3, 4, 5, 6, 8}
```

```
1  b.union(a)
```

```
{1, 2, 3, 4, 5, 6, 8}
```

② **집합의 이름 | 다른 집합의 이름**

– 또는, ‘|’ 기호로 연결하여 합집합을 구할 수도 있다.

```
1  a | b
```

```
{1, 2, 3, 4, 5, 6, 8}
```

집합 (Set)

- 차집합 (Difference) 구하기

① **집합의 이름.difference(다른 집합의 이름)**

– 대상 집합과 다른 집합의 차집합을 구한다.

```
1  a = {1, 2, 3, 4, 5}
2  b = {2, 4, 6, 8}
3
4  a.difference(b)
```

```
{1, 3, 5}
```

```
1  b.difference(a)
```

```
{6, 8}
```

```
1  a - b
```

```
{1, 3, 5}
```

```
1  b - a
```

```
{6, 8}
```

② **집합의 이름 - 다른 집합의 이름**

– 또는, '-' 기호로 연결하여 차집합을 구할 수도 있다.

집합 (Set)

- 집합의 크기 구하기

  len(집합의 이름)

  – 대상 집합에 원소가 모두 몇 개 존재하는지 확인하여 그 개수를 되돌려 준다.

```
1  s1 = {1, 2, 3}
2  print(len(s1))
3
4  s2 = {2.5, 5, 7.5, 10}
5  print(len(s2))
6
7  s5 = set("Hello")
8  print(len(s5))
9
10 s6 = set()
11 print(len(s6))
```

```
3
4
4
0
```

집합 (Set)

- 집합 자료형 확인하기

  type(자료형을 확인하고 싶은 1개의 자료)

  – 명령어 type을 이용하여 값 또는 변수가 어떤 자료형인지 확인할 수 있다.

```
1  s1 = {1, 2, 3}
2  print(type(s1))
3
4  s2 = {2.5, 5, 7.5, 10}
5  print(type(s2))
6
7  s5 = set("Hello")
8  print(type(s5))
9
10 s6 = set()
11 print(type(s6))
```

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