UNIT 26

세트 사용하기

26 세트 사용하기

세트 사용하기

- 파이썬은 집합을 표현하는 세트(set)라는 자료형을 제공함
- 집합을 영어로 하면 세트인데 수학에서 배우는 그 집합이 맞음
- 세트는 합집합, 교집합, 차집합 등의 연산이 가능함

26.1 세트 만들기

세트 만들기

- 세트는 { }(중괄호) 안에 값을 저장하며 각 값은 ,(콤마)로 구분해줌

- 세트 = {값1, 값2, 값3}

```
>>> fruits = {'strawberry', 'grape', 'orange', 'pineapple', 'cherry'}
>>> fruits
{'pineapple', 'orange', 'grape', 'strawberry', 'cherry'}
```

- 세트를 출력해보면 매번 요소의 순서가 다르게 나옴
- 세트에 들어가는 요소는 중복될 수 없음

```
>>> fruits = {'orange', 'orange', 'cherry'}
>>> fruits
{'cherry', 'orange'}
```

26.1 세트 만들기

세트 만들기

- 세트는 리스트, 튜플, 딕셔너리와는 달리 [ ](대괄호)로 특정 요소만 출력할 수는 없음

```
>>> fruits = {'strawberry', 'grape', 'orange', 'pineapple', 'cherry'}
>>> print(fruits[0])
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    print(fruits[0])
TypeError: 'set' object does not support indexing
>>> fruits['strawberry']
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    fruits['strawberry']
TypeError: 'set' object is not subscriptable
```

26.1 세트 만들기

세트에 특정 값이 있는지 확인하기

- 세트에 특정 값이 있는지 확인하려면 리스트, 튜플, 딕셔너리에 사용했던 in 연산자를 사용하면 됨

• 값 in 세트

```
>>> fruits = {'strawberry', 'grape', 'orange', 'pineapple', 'cherry'}
>>> 'orange' in fruits
True
>>> 'peach' in fruits
False
```

- 세트에 특정 값이 있으면 True, 없으면 False가 나옴
- 반대로 in 앞에 not을 붙이면 특정 값이 없는지 확인함

• 값 not in 세트

```
>>> 'peach' not in fruits
True
>>> 'orange' not in fruits
False
```

26.1 세트 만들기

set를 사용하여 세트 만들기

- set(반복가능한객체)

```
>>> a = set('apple')    # 유일한 문자만 세트로 만듦
>>> a
{'e', 'l', 'a', 'p'}
```

```
>>> b = set(range(5))
>>> b
{0, 1, 2, 3, 4}
```

```
>>> c = set()
>>> c
set()
```

- 세트가 { }를 사용한다고 해서 c = {}와 같이 만들면 빈 딕셔너리가 만들어지므로 주의해야 함

26.1 세트 만들기

set를 사용하여 세트 만들기

- 다음과 같이 type을 사용하면 자료형의 종류를 알 수 있음

- type(객체)

```
>>> c = {}
>>> type(c)
<class 'dict'>
>>> c = set()
>>> type(c)
<class 'set'>
```

26.2 집합 연산 사용하기

집합 연산 사용하기

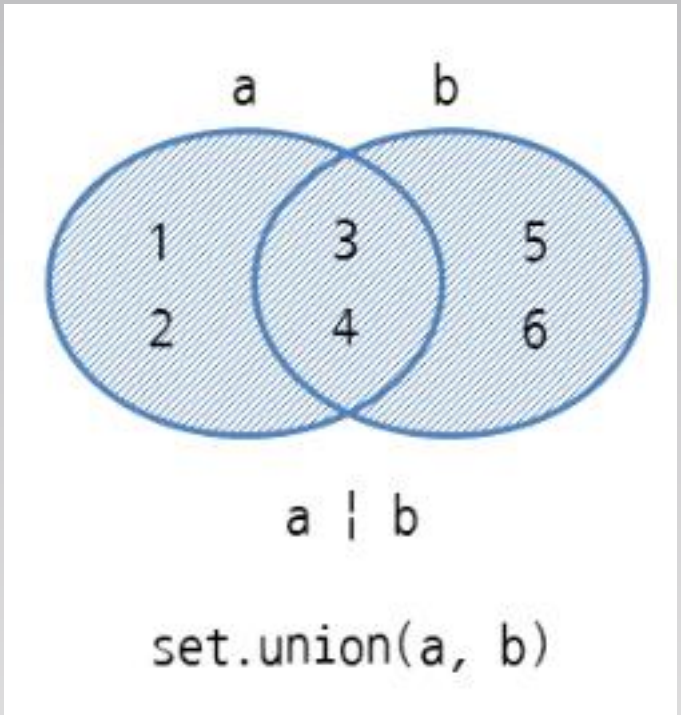
- 집합 연산은 파이썬의 산술 연산자와 논리 연산자를 활용함
- | 연산자는 합집합(union)을 구하며 OR 연산자 |를 사용함

- 세트1 | 세트2
- set.union(세트1, 세트2)

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

26.2 집합 연산 사용하기

▼ 그림 세트의 합집합 연산



26.2 집합 연산 사용하기

집합 연산 사용하기

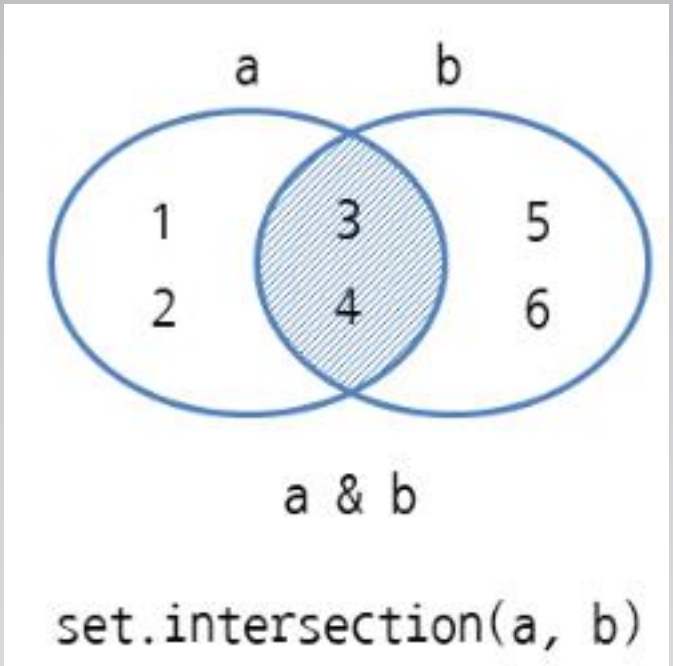
- & 연산자는 교집합(intersection)을 구하며 AND 연산자 &를 사용함
- set.defference 메서드와 동작이 같음

- 세트1 & 세트2
- set.intersection(세트1, 세트2)

```
>>> a & b
{3, 4}
>>> set.intersection(a, b)
{3, 4}
```

26.2 집합 연산 사용하기

▼ 그림 세트의 교집합 연산



26.2 집합 연산 사용하기

집합 연산 사용하기

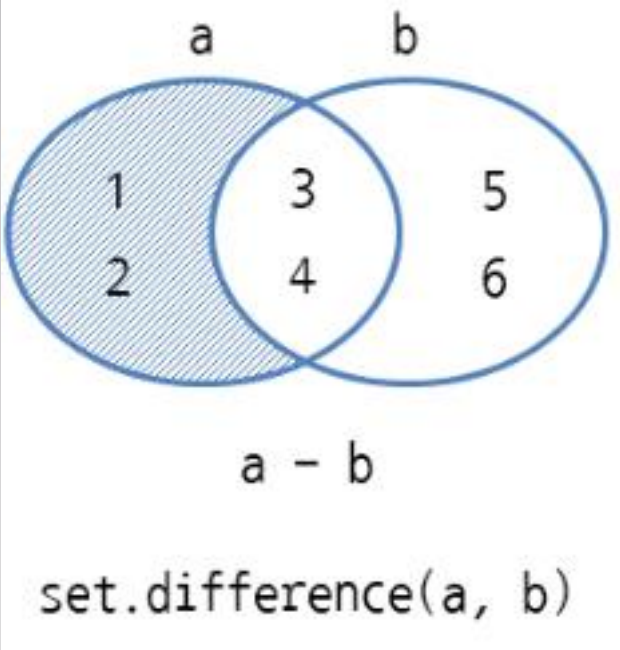
- - 연산자는 차집합(difference)을 구하며 뺄셈 연산자 -를 사용함
- set.defference 메서드와 동작이 같음

- 세트1 - 세트2
- set.difference(세트1, 세트2)

```
>>> a - b
{1, 2}
>>> set.difference(a, b)
{1, 2}
```

26.2 집합 연산 사용하기

▼ 그림 세트의 차집합 연산



26.2 집합 연산 사용하기

집합 연산 사용하기

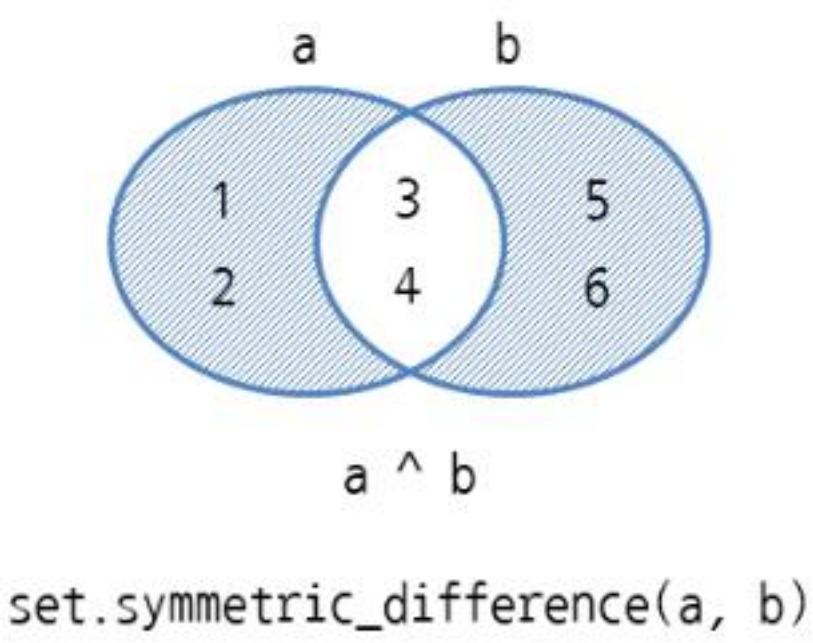
- ^ 연산자는 대칭차집합(symmetric difference)을 구하며 XOR 연산자 ^를 사용함
- set.symmetric_difference 메서드와 동작이 같음
- 대칭차집합은 XOR 연산자의 특성을 그대로 따르는데 XOR은 서로 다르면 참임
- 집합에서는 두 집합 중 겹치지 않는 요소만 포함함

- 세트1 ^ 세트2
- set.symmetric_difference(세트1, 세트2)

```
>>> a ^ b
{1, 2, 5, 6}
>>> set.symmetric_difference(a, b)
{1, 2, 5, 6}
```

26.2 집합 연산 사용하기

▼ 그림 세트의 대칭차집합 연산



26.2 집합 연산 사용하기

집합 연산 후 할당 연산자 사용하기

- 세트 자료형에 |, &, -, ^ 연산자와 할당 연산자 =을 함께 사용하면 집합 연산의 결과를 변수에 다시 저장(할당)함
- |=은 현재 세트에 다른 세트를 더하며 update 메서드와 같음

- 세트1 |= 세트2
- 세트1.update(세트2)

```
>>> a = {1, 2, 3, 4}
>>> a |= {5}
>>> a
{1, 2, 3, 4, 5}
>>> a = {1, 2, 3, 4}
>>> a.update({5})
>>> a
{1, 2, 3, 4, 5}
```

26.2 집합 연산 사용하기

집합 연산 후 할당 연산자 사용하기

- &=은 현재 세트와 다른 세트 중에서 겹치는 요소만 현재 세트에 저장하며 intersection_update 메서드와 같음

- 세트1 &= 세트2
- 세트1.intersection_update(세트2)

```
>>> a = {1, 2, 3, 4}
>>> a &= {0, 1, 2, 3, 4}
>>> a
{1, 2, 3, 4}
>>> a = {1, 2, 3, 4}
>>> a.intersection_update({0, 1, 2, 3, 4})
>>> a
{1, 2, 3, 4}
```

26.2 집합 연산 사용하기

집합 연산 후 할당 연산자 사용하기

- -=은 현재 세트에서 다른 세트를 빼며 difference_update 메서드와 같음

- 세트1 -= 세트2
- 세트1.difference_update(세트2)

```
>>> a = {1, 2, 3, 4}
>>> a -= {3}
>>> a
{1, 2, 4}
>>> a = {1, 2, 3, 4}
>>> a.difference_update({3})
>>> a
{1, 2, 4}
```

26.2 집합 연산 사용하기

집합 연산 후 할당 연산자 사용하기

- ^=은 현재 세트와 다른 세트 중에서 겹치지 않는 요소만 현재 세트에 저장하며 symmetric_difference_update 메서드와 같음

- 세트1 ^= 세트2
- 세트1.symmetric_difference_update(세트2)

```
>>> a = {1, 2, 3, 4}
>>> a ^= {3, 4, 5, 6}
>>> a
{1, 2, 5, 6}
>>> a = {1, 2, 3, 4}
>>> a.symmetric_difference_update({3, 4, 5, 6})
>>> a
{1, 2, 5, 6}
```

26.2 집합 연산 사용하기

부분 집합과 상위집합 확인하기

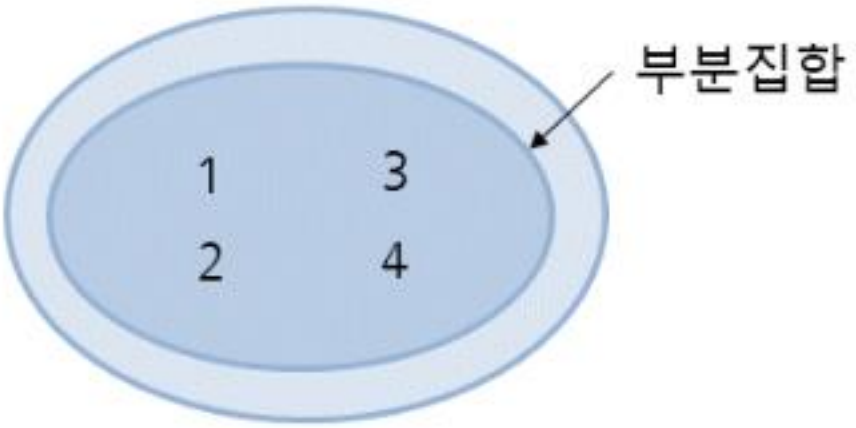
- 세트는 부분집합, 진부분집합, 상위집합, 진상위집합과 같이 속하는 관계를 표현할 수도 있음
- 현재 세트가 다른 세트의 (진)부분집합 또는 (진)상위집합인지 확인할 때는 세트 자료형에 부등호와 등호 사용함
- <=은 현재 세트가 다른 세트의 부분집합(subset)인지 확인하며 issubset 메서드와 같음

- 현재세트 <= 다른세트
- 현재세트.issubset(다른세트)

```
>>> a = {1, 2, 3, 4}
>>> a <= {1, 2, 3, 4}
True
>>> a.issubset({1, 2, 3, 4, 5})
True
```

26.2 집합 연산 사용하기

▼ 그림 세트가 부분집합인지 확인




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

{1, 2, 3, 4}.issubset({1, 2, 3, 4, 5})
```

26.2 집합 연산 사용하기

부분 집합과 상위집합 확인하기

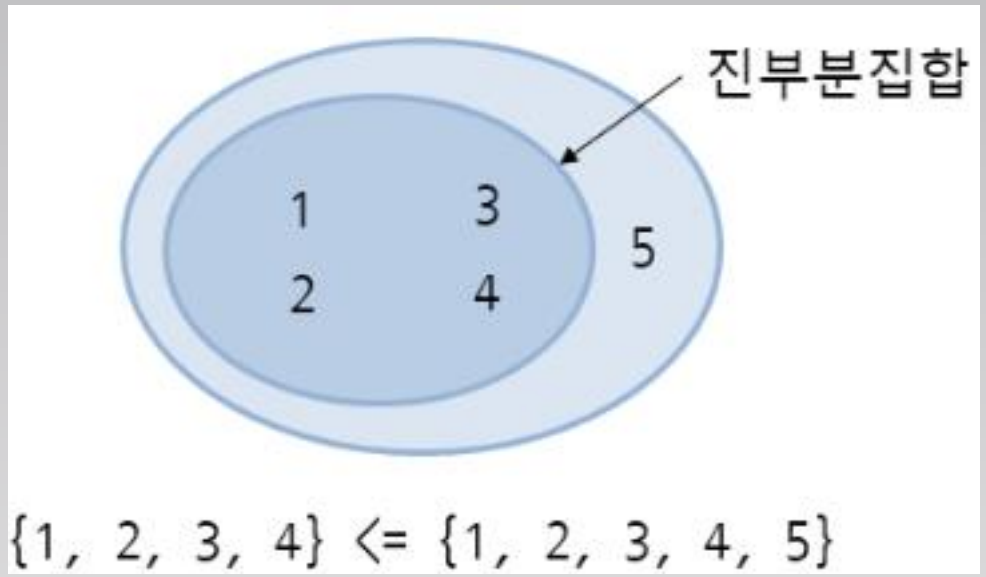
- <은 현재 세트가 다른 세트의 진부분집합(proper subset)인지 확인하며 메서드는 없음
- 다음은 세트 {1, 2, 3, 4}가 {1, 2, 3, 4, 5}의 진부분집합이므로 참임
- 부분집합이지만 같지는 않을 때 참임

• 현재세트 < 다른세트

```
>>> a = {1, 2, 3, 4}
>>> a < {1, 2, 3, 4, 5}
True
```

26.2 집합 연산 사용하기

▼ 그림 세트가 진부분집합인지 확인



26.2 집합 연산 사용하기

부분 집합과 상위집합 확인하기

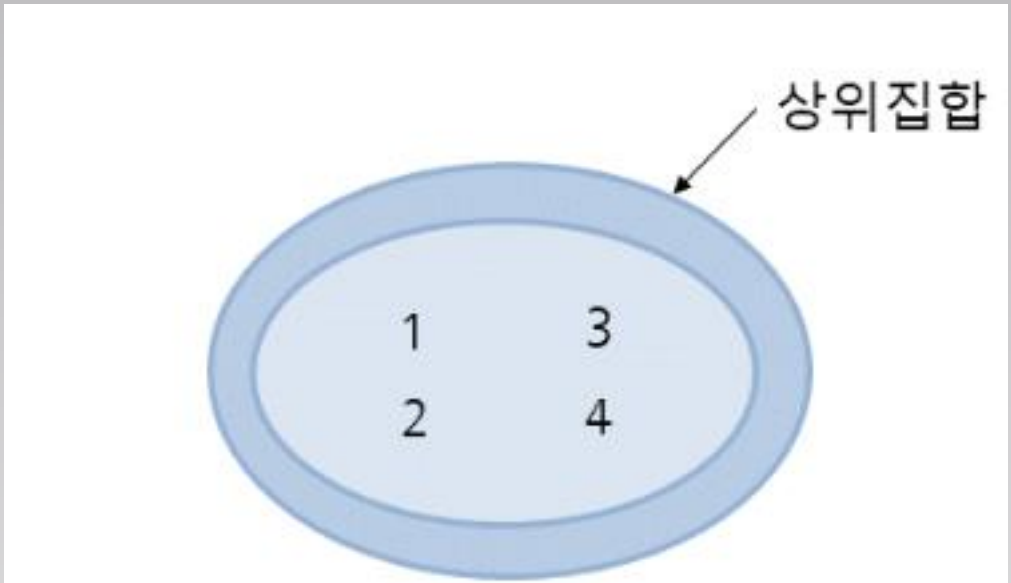
- 〉=은 현재 세트가 다른 세트의 상위집합(superset)인지 확인하며 issuperset 메서드와 같음

- 현재세트 >= 다른세트
- 현재세트.issuperset(다른세트)

```
>>> a = {1, 2, 3, 4}
>>> a >= {1, 2, 3, 4}
True
>>> a.issuperset({1, 2, 3, 4})
True
```

26.2 집합 연산 사용하기

▼ 그림 세트가 상위집합인지 확인




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

{1, 2, 3, 4}.issuperset({1, 2, 3, 4})
```

26.2 집합 연산 사용하기

부분 집합과 상위집합 확인하기

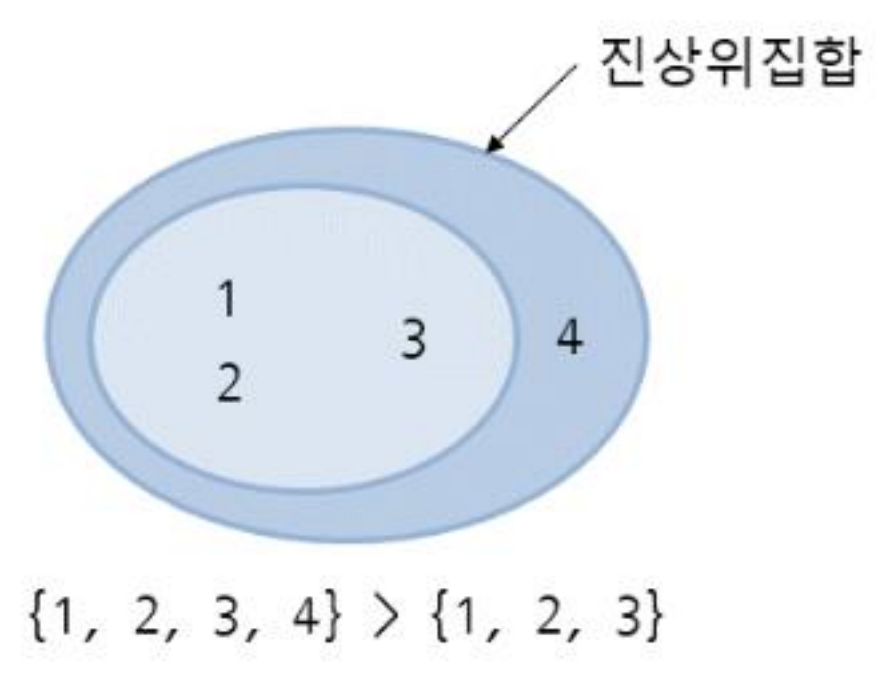
- 〉은 현재 세트가 다른 세트의 진상위집합(proper superset)인지 확인하며 메서드는 없음
- 다음은 세트 {1, 2, 3, 4}가 {1, 2, 3}의 진상위집합이므로 참임
- 상위집합이지만 같지는 않을 때 참임

• 현재세트 〉 다른세트

```
>>> a = {1, 2, 3, 4}
>>> a > {1, 2, 3}
True
```

26.2 집합 연산 사용하기

▼ 그림 세트가 진상위집합인지 확인



26.2 집합 연산 사용하기

세트가 같은지 다른지 확인하기

- 세트는 == 연산자를 사용하여 서로 같은지 확인할 수 있음

```
>>> a = {1, 2, 3, 4}
>>> a == {1, 2, 3, 4}
True
>>> a == {4, 2, 1, 3}
True
```

- 세트는 요소의 순서가 정해져 있지 않으므로 ==로 비교했을 때 각 요소만 같으면 참임
- != 연산자는 세트가 다른지 확인함

```
>>> a = {1, 2, 3, 4}
>>> a != {1, 2, 3}
True
```

26.2 집합 연산 사용하기

세트가 겹치지 않는지 확인하기

- disjoint는 현재 세트가 다른 세트와 겹치지 않는지 확인

• 현재세트.isdisjoint(다른세트)

```
>>> a = {1, 2, 3, 4}
>>> a.isdisjoint({5, 6, 7, 8})       # 겹치는 요소가 없음
True
>>> a.isdisjoint({3, 4, 5, 6})    # a와 3, 4가 겹침
False
```

26.3 세트 조작하기

세트에 요소 추가하기

- add(요소)는 세트에 요소를 추가함

```
>>> a = {1, 2, 3, 4}
>>> a.add(5)
>>> a
{1, 2, 3, 4, 5}
```

26.3 세트 조작하기

세트에서 특정 요소를 삭제하기

- remove(요소)는 세트에서 특정 요소를 삭제하고 요소가 없으면 에러를 발생시킴

```
>>> a.remove(3)
>>> a
{1, 2, 4, 5}
```

- discard(요소)는 세트에서 특정 요소를 삭제하고 요소가 없으면 그냥 넘어감
- 다음은 세트 a에 2가 있으므로 2를 삭제하고, 3은 없으므로 그냥 넘어감

```
>>> a.discard(2)
>>> a
{1, 4, 5}
>>> a.discard(3)
>>> a
{1, 4, 5}
```

26.3 세트 조작하기

세트에서 임의의 요소 삭제하기

- pop()은 세트에서 임의의 요소를 삭제하고 해당 요소를 반환함
- 만약 요소가 없으면 에러를 발생시킴

```
>>> a = {1, 2, 3, 4}
>>> a.pop()
1
>>> a
{2, 3, 4}
```

26.3 세트 조작하기

세트의 모든 요소를 삭제하기

- clear()는 세트에서 모든 요소를 삭제

```
>>> a.clear()
>>> a
set()
```

26.3 세트 조작하기

세트의 요소 개수 구하기

- len(세트)는 세트의 요소 개수(길이)를 구함

```
>>> a = {1, 2, 3, 4}
>>> len(a)
4
```

26.4 세트의 할당과 복사

세트의 할당과 복사

- 세트를 만든 뒤 다른 변수에 할당함

```
>>> a = {1, 2, 3, 4}
>>> b = a
```

- b = a와 같이 세트를 다른 변수에 할당하면 세트는 두 개가 될 것 같지만 실제로는 세트가 한 개임
- 변수 이름만 다를 뿐 세트 a와 b는 같은 객체임

```
>>> a is b
True
```

- a와 b는 같으므로 b에 요소를 추가하면 세트 a와 b에 모두 반영됨

```
>>> b.add(5)
>>> a
{1, 2, 3, 4, 5}
>>> b
{1, 2, 3, 4, 5}
```

26.4 세트의 할당과 복사

세트의 할당과 복사

- 세트 a와 b를 완전히 두 개로 만들려면 copy 메서드로 모든 요소를 복사해야 함

```
>>> a = {1, 2, 3, 4}
>>> b = a.copy()
```

- a와 b를 is 연산자로 비교해보면 False가 나옴
- 두 세트는 다른 객체임
- 복사한 요소는 같으므로 ==로 비교하면 True가 나옴

```
>>> a is b
False
>>> a == b
True
```

26.4 세트의 할당과 복사

세트의 할당과 복사

- 세트 a와 b는 별개이므로 한쪽의 값을 변경해도 다른 세트에 영향을 미치지 않음

```
>>> a = {1, 2, 3, 4}
>>> b = a.copy()
>>> b.add(5)
>>> a
{1, 2, 3, 4}
>>> b
{1, 2, 3, 4, 5}
```

26.5 반복문으로 세트의 요소를 모두 출력하기

반복문으로 세트의 요소를 모두 출력하기

- 간단하게 for in 뒤에 세트만 지정하면 됨

```
for 변수 in 세트:
    반복할 코드
```

```
>>> a = {1, 2, 3, 4}
>>> for i in a:
...     print(i)
...
1
2
3
4
```

- print로 i를 출력하면 요소를 모두 출력할 수 있음
- 세트의 요소는 순서가 없으므로 출력할 때마다 순서가 달라짐(숫자로만 이루어진 세트는 순서대로 출력됨)
- in 다음에 세트를 직접 지정해도 상관 없음

26.6 세트 표현식 사용하기

세트 표현식 사용하기

- 세트는 for 반복문과 if 조건문을 사용하여 세트를 생성할 수 있음

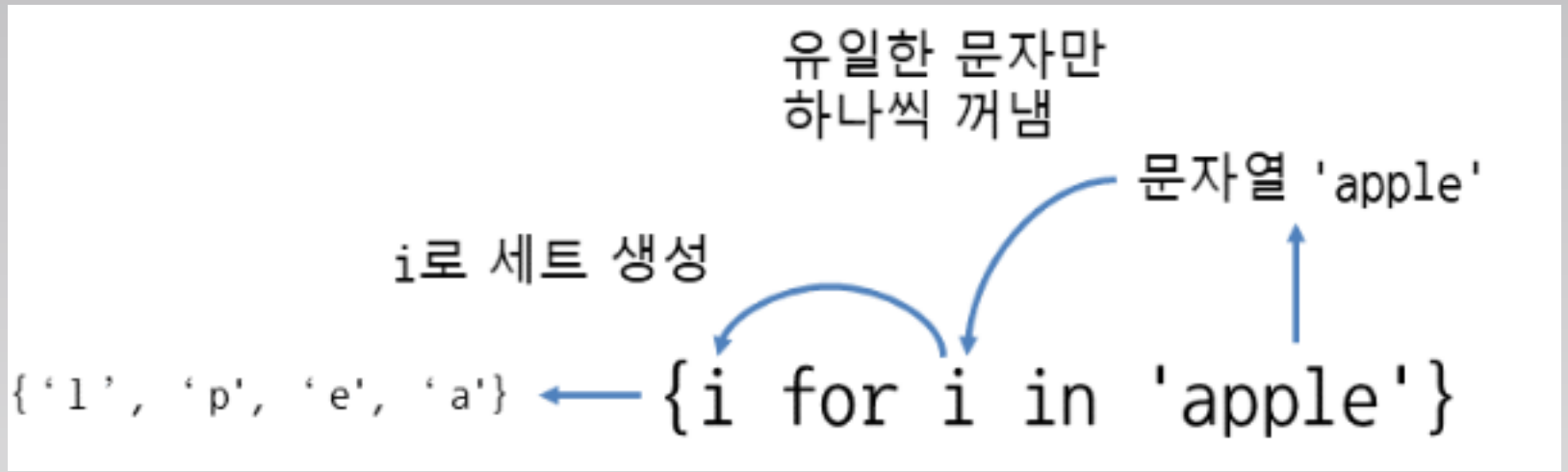
```
• {식 for 변수 in 반복가능한객체}
• set(식 for 변수 in 반복가능한객체)
```

```
>>> a = {i for i in 'apple'}
>>> a
{'l', 'p', 'e', 'a'}
```

- { } 또는 set() 안에 식, for, 변수, in, 반복 가능한 객체를 지정하여 세트를 생성함
- 반복 가능한 객체로 문자열 'apple'을 지정함
- 문자열에서 중복된 문자는 세트에 포함되지 않음

26.6 세트 표현식 사용하기

▼ 그림 세트 표현식의 동작 순서



26.6 세트 표현식 사용하기

세트 표현식에 if 조건문 사용하기

- 다음과 같이 if 조건문은 for 반복문 뒤에 지정함

- {식 for 변수 in 세트 if 조건식}
- set(식 for 변수 in 세트 if 조건식)

```
>>> a = {i for i in 'pineapple' if i not in 'apl'}
>>> a
{'e', 'i', 'n'}
```

26.6 세트 표현식 사용하기

▼ 그림 세트 표현식에서 if 조건문 사용하기