11. 집합 자료형

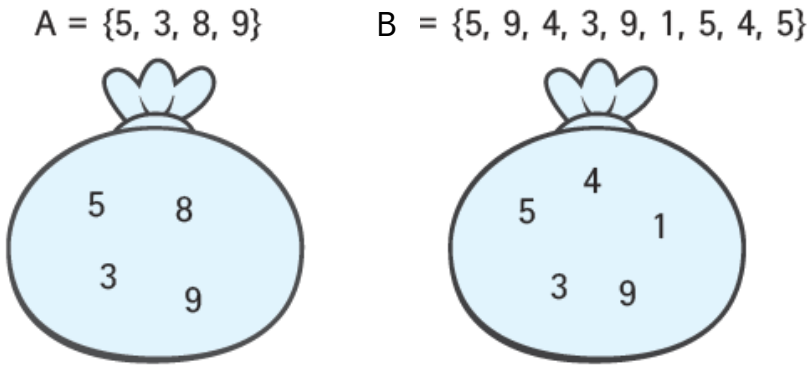
1. 집합 만들기
2. 집합은 인덱싱(indexing), 슬라이싱(slicing)할 수 없습니다
3. 집합은 mutable 객체입니다
4. for 반복문과 집합
5. 집합에 함수 적용하기
6. 집합에 in, not in, del 연산자 사용하기
7. 집합의 원소 추가/삭제 메소드
8. 집합의 연산 메소드
9. 기호로 합집합(|), 교집합(&), 차집합(-) 구하기
10. 집합 안에 for 반복문 사용하기 (Set Comprehension)
11. frozenset 자료형
12. 정리

1. 집합 만들기

- 집합의 특징
  - 중복된 원소가 없음
  - 집합 내의 원소 간에는 순서 개념이 없음
  - 파이썬에서 집합은 mutable한 객체임
- 집합의 연산
  - 집합의 원소 여부 ($\in$,$\notin$)
  - 부분 집합 ($\subset$,$\not\subset$)
  - 집합의 연산 - 합집합, 교집합, 차집합

1. 집합 만들기

- 파이썬에서도 집합 기호는 {} 임
- 집합은 중복된 원소를 한 개의 원소로 간주함. (집합 A의 원소의 개수는 4개, B의 원소의 개수는 5개임)



- 집합은 순서 개념이 없기 때문에 인덱스가 없음 (시퀀스 자료형이 아님)
- 집합은 mutable 객체라서 집합에 원소를 추가하거나 삭제할 수 있음
- 집합에는 immutable 객체들만 저장되어야 함 (리스트, 집합, 사전은 넣을 수 없음)

1. 집합 만들기

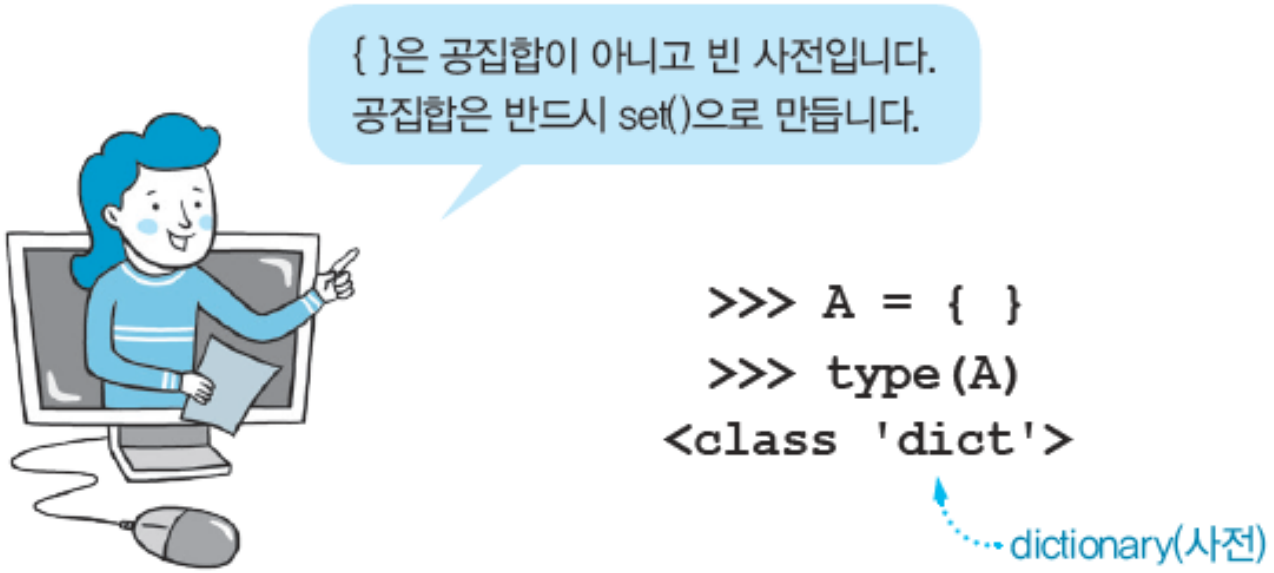
◆ 집합 만들기 예제

```
>>> colors = {'white', 'black', 'gray', 'red', 'blue'}
>>> type(colors)                    # 집합의 type은 set입니다.
<class 'set'>
>>> print(colors)                   # 집합에는 순서 개념이 없습니다.
{'white', 'gray', 'blue', 'black', 'red'}
>>> A = {3, 5, 9, 1, 5, 7, 3, 5}    # 집합에서 중복된 데이터는 하나로 취급합니다.
>>> print(A)
{1, 3, 5, 7, 9}
>>> len(A)
5
```

◆ 빈 집합 만들기

```
>>> S = set( )      # 공집합을 만들려면 반드시 set( )이라고 해야 합니다.
>>> type(S)
<class 'set'>
```

1. 집합 만들기




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

dictionary(사전)

1. 집합 만들기

- 집합에는 immutable 자료형만 저장 가능함
  - mutable 자료형을 넣으면 TypeError 발생함.

```
>>> S = {1, 3.5, True, 2+5j}
>>> S
{(2+5j), 1, 3.5}

>>> T = {[1,2], 3, 4, 5}                    # 집합에 리스트를 넣으면 TypeError가 발생합니다.
.....
TypeError: unhashable type: 'list'

>>> W = {1, 2, {3, 4, 5}, 10, 20}           # 집합에 집합을 넣으면 TypeError가 발생합니다.
.....
TypeError: unhashable type: 'set'

>>> V = {10, {1:'one', 2:'two'}, 70}        # 집합에 사전을 넣으면 TypeError가 발생합니다.
.....
TypeError: unhashable type: 'dict'
```

1. 집합 만들기

- 다른 자료형의 데이터를 집합으로 변환하기
  - set() 함수를 이용하여 iterable 객체를 집합 객체로 변환할 수 있음

집합 ← 문자열	>>> book = 'python programming'
	>>> S1 = set(book) # 중복을 제거하고 저장합니다.
	>>> print(S1)
	{'i', 'm', ' ', 'o', 'r', 't', 'y', 'h', 'a', 'p', 'g', 'n'}

집합 ← 리스트	>>> score = [88, 94, 70, 85]
	>>> S2 = set(score)
	>>> print(S2)
	{88, 70, 85, 94}

집합 ← 튜플	>>> data = (1, 3, 5, 10, 2, 3)
	>>> S3 = set(data)
	>>> print(S3)
	{1, 2, 3, 5, 10}

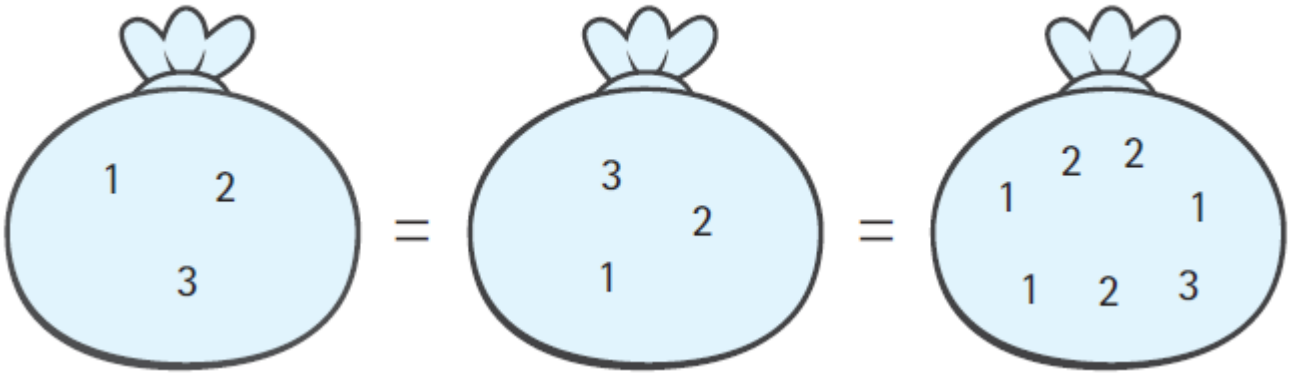
집합 ← 사전	>>> sports = {'baseball':9, 'basketball':6, 'soccer':11}
	>>> S4 = set(sports) # 키만 가져옵니다.
	>>> print(S4)
	{'baseball', 'basketball', 'soccer'}

1. 집합 만들기

- 두 집합이 같은지 비교하기
  - 두 집합의 원소의 구성이 같아야 함

```
〉〉〉 A = {1, 2, 3}
〉〉〉 B = {2, 3, 1}
〉〉〉 C = {1, 2, 2, 2, 3, 1, 2}      # 중복된 원소는 하나로 취급합니다.
〉〉〉 A == B, A == C, B == C
(True, True, True)
```

아래 세 집합은 모두 같습니다.



2. 집합은 인덱싱, 슬라이싱할 수 없습니다

- 집합은 순서 개념이 없기 때문에 인덱싱/슬라이싱할 수 없음
- 데이터를 순서 개념을 갖고 유지하려면 집합을 사용하면 안 됨

3. 집합은 mutable 객체입니다

- 집합 객체를 생성한 후에 새로운 원소를 추가하거나 이미 존재하는 원소를 삭제할 수 있음
- 집합은 인덱스 개념이 없기 때문에 반드시 메소드를 이용해서 원소를 추가/삭제해야 함

4. for 반복문과 집합

- 집합은 iterable 객체이므로 for 반복문에 바로 이용 가능함
- 이 때, 어떤 순서로 루프를 수행할 지 알 수 없음

s = {'red', 'yellow', 'blue'} for x in s: print(x)	[결과] blue red yellow
s = {3.5, 8, 2.1, 5.0} for x in s: print(x)	[결과] 8 3.5 5.0 2.1

5. 집합에 함수 적용하기

len()	집합 안의 원소 개수를 반환합니다.
max()	집합 안의 원소 중에서 가장 큰 수를 반환합니다.
min()	집합 안의 원소 중에서 가장 작을 수를 반환합니다.
sum()	집합 안의 원소의 합을 구하여 반환합니다. 원소들이 수치자료형이어야 합니다.
sorted()	집합을 오름차순으로 정렬시켜 리스트로 반환합니다.
reversed()	집합 자료형에는 reversed() 함수를 적용할 수 없습니다.

```
>>> S = {1, 2, 3, 4, 3}
>>> len(S)
4
>>> max(S)
4
>>> min(S)
1
>>> sum(S)     # 중복된 데이터는 한번만 더함
10
```

```
>>> S = {5, 7, 9, 2, 3, 6}
>>> T = sorted(S)        # 리스트로 반환함
>>> T
[2, 3, 5, 6, 7, 9]
>>> R = reversed(S)     # 집합에 적용 못 함
.....
R = reversed(S)
TypeError: 'set' object is not reversible
```

5. 집합에 함수 적용하기

CODE 60 리스트 score에서 동점인 학생 수를 출력하는 코드

```
score = [80, 90, 77, 80, 87, 95, 87, 80, 85, 90, 90, 80]
s_set = set(score)     # 리스트에서 중복을 없앰
for s in s_set:
    print('{} : {}'.format(s, score.count(s)))
```

[결과]

```
77 : 1
80 : 4
85 : 1
87 : 2
90 : 3
95 : 1
```

5. 집합에 함수 적용하기

CODE 61 하나의 문자열을 input() 으로 입력받아서 집합을 이용하여 입력받은 문자열에 알파벳이 각각 몇 번씩 나오는지를 출력하는 코드. 예를 들어서, 문자열 'python'에는 'p', 'y', 't', 'h', 'o', 'n' 이 각각 1개씩 있음.

```
string = input('Enter string : ')
string_set = set(string)          # 입력받은 문자열을 집합으로 변환합니다.
for s in string_set:              # 각 문자가 string에 몇 번 나오는지 찾습니다.
    print('{} : {}'.format(s, string.count(s)))
```

[결과 1]

```
Enter string : mississippi
s : 4
m : 1
p : 2
i : 4
```

[결과 2]

```
Enter string : good
g : 1
d : 1
o : 2
```

[결과 3]

```
Enter string : desk
e : 1
s : 1
d : 1
k : 1
```

6. 집합에 in, not in, del 연산자 사용하기

- in, not in 집합에 원소인지를 판단함 (∈,∉ 기호)
- del 연산자를 이용하면 집합을 통째로 삭제 가능함
- 집합에는 +, * 연산은 적용할 수 없음

```
〉〉〉 S = {5, 9, 2, 8, 1}
〉〉〉 9 in S          # 9 ∈ S
True
〉〉〉 7 not in S      # 7∉ S
True
```

```
〉〉〉 T = {2, 4, 6}
〉〉〉 del T          # 집합을 통째로 삭제
〉〉〉 T
.....
NameError: name 'T' is not defined
```

7. 집합의 원소 추가/삭제 메소드

◆ 집합에 적용할 수 있는 메소드 목록

```
〉〉〉 dir(set)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__',
'__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__',
'__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__ repr__',
'__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update',
'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset',
'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union',
'update']
```

7. 집합의 원소 추가/삭제 메소드

◆ 집합에 원소를 추가하거나 삭제하는 메소드

메소드	설명	반환값	발생 가능 에러
add(x)	집합에 데이터 x를 추가합니다. 만약에 x가 이미 집합에 있으면 추가하지 않습니다.	None	없음
clear()	집합을 비웁니다(공집합을 만듬).	None	없음
copy()	집합을 복사하여 새로운 집합을 반환합니다.	있음	없음
discard(x)	집합에 데이터 x를 삭제합니다. x가 없으면 아무 일도 생기지 않습니다.	None	없음
pop()	집합에서 임의로 하나의 원소를 삭제하고 반환합니다.	있음	KeyError
remove(x)	집합에서 데이터 x를 삭제합니다.	None	KeyError

7. 집합의 원소 추가/삭제 메소드

- 집합에 데이터 추가하기 – add(x)
  - 이미 존재하는 원소를 추가하려면 추가되지 않음
  - x에는 mutable 객체는 넣을 수 없음

```
>>> S = {4, 5, 6, 7}
>>> a = S.add(5)        # S에 원소 5를 추가하는데, 중복 데이터라서 추가되지 않습니다.
>>> print(a)            # add( ) 메소드는 반환값이 없습니다. None이 출력됩니다.
None
>>> S.add(10)           # 10은 없는 데이터이기 때문에 S에 추가됩니다.
>>> print(S)
{4, 5, 6, 7, 10}
```

7. 집합의 원소 추가/삭제 메소드

- 공집합 만들기 – clear()

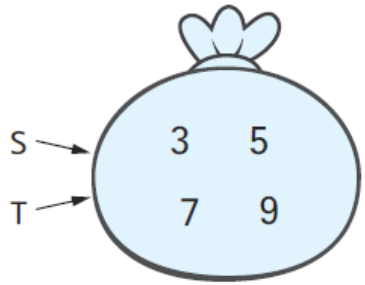
```
>>> print(S)
{4, 5, 6, 7, 10}
>>> S.clear( )          # 집합을 공집합으로 만듭니다.
>>> len(S)
0
>>> bool(S)             # 공집합은 False로 간주합니다.
False
```

7. 집합의 원소 추가/삭제 메소드

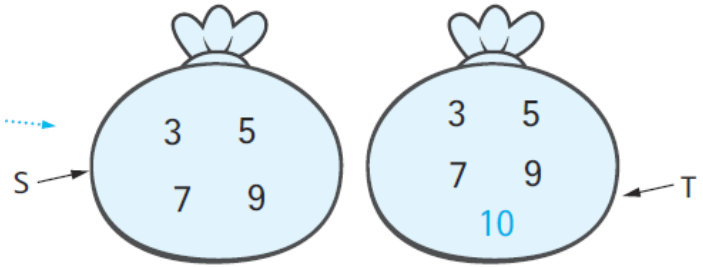
◆ 집합 복사하기 – copy()

- 리스트처럼 '=' 연산자는 복사본을 만들지 못 함.
- 복사본을 만들려면 copy() 메소드를 사용해야 함.

```
〉〉〉 S = {3, 5, 7, 9}
〉〉〉 T = S
〉〉〉 id(S), id(T)
(35343936, 35343936)
〉〉〉 T.add(10)
〉〉〉 S, T
({3, 5, 7, 9, 10}, {3, 5, 7, 9, 10})
```




```
〉〉〉 S = {3, 5, 7, 9}
〉〉〉 T = S.copy( )
〉〉〉 id(S), id(T)
(35343696, 35343816)
〉〉〉 T.add(10)
〉〉〉 S, T
({9, 3, 5, 7}, {3, 5, 7, 9, 10})
```



copy() 메소드를 이용하면 복사본이 생깁니다.

7. 집합의 원소 추가/삭제 메소드

- 집합에 있는 데이터 삭제하기 – discard(), pop(), remove()
  - discard(x) – 집합에서 x를 삭제함. 반환값 없음. x가 없는 원소라도 에러 발생하지 않음.
  - pop() – 임의의 원소를 선택하여 삭제하고 그 원소를 반환함. 공집합에 pop() 메소드를 적용하면 KeyError 발생함.
  - remove(x) – discard(x)처럼 x를 삭제하는 메소드이고 반환값 없음. discard(x)와 다른 점은 x가 없는 원소인 경우 KeyError 발생함.

```
>>> S = {3, 4, 5}
>>> result = S.discard(4)     # discard( ) 메소드는 원소를 삭제하고 반환값은 없습니다.
>>> print(result)
None
>>> print(S)
{3, 5}
>>> S.clear( )
>>> S.discard(7)          # 공집합에 discard( ) 메소드를 적용해도 에러가 없습니다.
```

7. 집합의 원소 추가/삭제 메소드

```
>>> S = {5, 9, 10}
>>> result = S.pop( )              # pop( ) 메소드는 임의의 수를 선택해서 삭제하고 반환합니다.
>>> print(result)
9
>>> print(S)
{10, 5}
>>> S.clear( )                     # S를 공집합으로 만듭니다.
>>> S.pop( )                       # 공집합에 pop( ) 메소드를 적용하면 KeyError가 발생합니다.
.....
KeyError: 'pop from an empty set'
```

```
>>> S = {4, 7, 9}
>>> result = S.remove(7)           # remove( ) 메소드는 원소를 삭제하고 반환값은 없습니다.
>>> print(result)
None
>>> print(S)
{9, 4}
>>> S.remove(7)                    # 없는 원소를 remove( )하면 KeyError가 발생합니다.
.....
KeyError: 7
```

8. 집합의 연산 메소드

연산	메소드	반환값	발생 가능 에러
합집합	union()	합집합 결과	없음
	update()	None	없음
교집합	intersection()	교집합 결과	없음
	intersection_update()	None	없음
차집합	difference()	차집합 결과	없음
	difference_update()	None	없음
	symmetric_difference()	차집합 결과	없음
	symmetric_difference_update()	None	없음
부분 집합	issubset()	True/False	없음
	issuperset()	True/False	없음
서로소	isdisjoint()	True/False	없음

8. 집합의 연산 메소드

◆ 합집합 - union(), update()

- union(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합과 x의 합집합을 구하여 반환함. (두 집합은 변하지 않음)
- update(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합에 x를 합함.

```
>>> A = {1,2,6,7,9}
>>> B = {1,3,5,7}
>>> C = A.union(B)        # C = A ∪ B
>>> A                     # 집합 A는 그대로입니다.
{1, 2, 6, 7, 9}
>>> B                     # 집합 B는 그대로입니다.
{1, 3, 5, 7}
>>> C
{1, 2, 3, 5, 6, 7, 9}
```

```
>>> A = {1,2,6,7,9}
>>> B = {1,3,5,7}
>>> C = A.update(B)       # A = A ∪ B
>>> print(C)              # 반환값이 없습니다.
None
>>> A                     # 집합 A가 합집합의 결과입니다.
{1, 2, 3, 5, 6, 7, 9}
>>> B
{1, 3, 5, 7}
```

8. 집합의 연산 메소드

◆ 교집합 - intersection(), intersection_update()

- intersection(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합과 x의 교집합을 구하여 반환함. (두 집합은 변하지 않음)
- intersection_update(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합이 교집합의 결과가 됨.

```
〉〉〉 A = {1, 2, 3, 4, 5, 6, 7}
〉〉〉 B = {2, 4, 6, 8}
〉〉〉 C = A.intersection(B)   # C=A∩B
〉〉〉 C
{2, 4, 6}
〉〉〉 A
{1, 2, 3, 4, 5, 6, 7}
〉〉〉 B
{8, 2, 4, 6}
```

```
〉〉〉 A = {1, 2, 3, 4, 5, 6, 7}
〉〉〉 B = {2, 4, 6, 8}
〉〉〉 C = A.intersection_update(B)    # A=A∩B
〉〉〉 print(C)      # 반환값이 없음.
None
〉〉〉 A              # 집합 A가 교집합의 결과임.
{2, 4, 6}
〉〉〉 B
{8, 2, 4, 6}
```

8. 집합의 연산 메소드

◆ 차집합 - difference(), difference_update()

- difference(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합과 x의 차집합을 구하여 반환함. (두 집합은 변하지 않음)
- difference_update(x) - 인수 x에는 집합을 넣음. 이 메소드를 호출하는 집합과 x의 차집합을 구함.

```
>>> A = {1,2,3,4,5,6,7,8}
>>> B = {1,3,4,6}
>>> C = A.difference(B)    # C=A-B
>>> C
{8, 2, 5, 7}
>>> A
{1, 2, 3, 4, 5, 6, 7, 8}
>>> B
{1, 3, 4, 6}
```

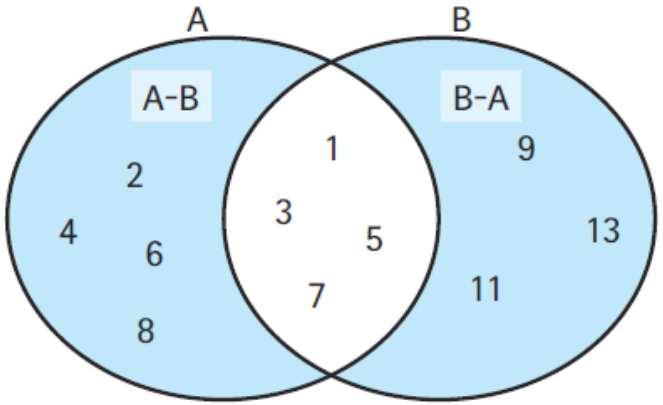
```
>>> A = {1,2,3,4,5,6,7,8}
>>> B = {1,3,4,6}
>>> C = A.difference_update(B)    # A=A-B
>>> print(C)
None
>>> A
{2, 5, 7, 8}
>>> B
{1, 3, 4, 6}
```

8. 집합의 연산 메소드

- symmetric_difference(), symmetric_difference_update()

```
>>> A = {1,2,3,4,5,6,7,8}
>>> B = {1,3,5,7,9,11,13}
>>> C = A.symmetric_difference(B)
>>> A
{1, 2, 3, 4, 5, 6, 7, 8}
>>> B
{1, 3, 5, 7, 9, 11, 13}
>>> C
{2, 4, 6, 8, 9, 11, 13}
```

```
>>> A = {1,2,3,4,5,6,7,8}
>>> B = {1,3,5,7,9,11,13}
>>> C = A.symmetric_difference_ update(B)
>>> print(C)
None
>>> A
{2, 4, 6, 8, 9, 11, 13}
>>> B
{1, 3, 5, 7, 9, 11, 13}
```



symmetric_difference = (A-B) ∪ (B-A)

8. 집합의 연산 메소드

◆ 부분 집합 확인하기 - issubset(), issuperset()

- issubset(x) - 메소드를 호출하는 집합이 x의 부분 집합인지 판단함.
- issuperset(x) - x가 메소드를 호출하는 집합의 부분 집합인지 판단함.

```
>>> A = {1,2,3,4,5}
>>> B = {2,3}
>>> B.issubset(A)
True
>>> A.issubset(B)
False
>>> A.issuperset(B)
True
```

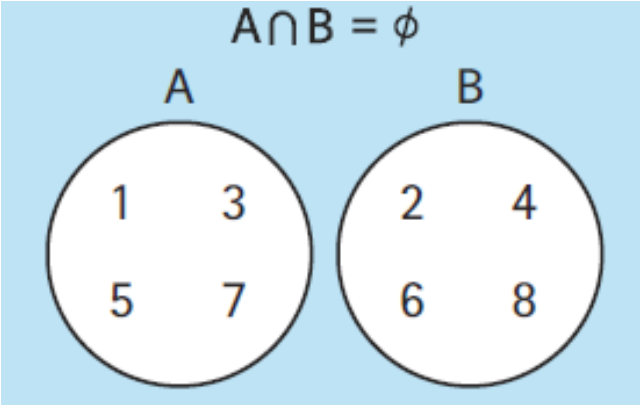
```
>>> C = {1,2,3}
>>> D = {1,2,3}
>>> C.issubset(D)
True
>>> C.issuperset(B)
True
```

8. 집합의 연산 메소드

◆ 서로소 확인하기 - isdisjoint()

- 두 집합 간에 공통된 원소가 없는 것을 판단함.

```
〉〉〉 A = {1,3,5,7}
〉〉〉 B = {2,4,6,8}
〉〉〉 A.isdisjoint(B)
True
〉〉〉 B.isdisjoint(A)
True
```



9. 기호로 합집합, 교집합, 차집합 구하기

- union() 메소드 대신 '|' 기호 사용 가능함
- intersection() 메소드 대신 '&' 기호 사용 가능함
- difference() 메소드 대신 '-' 기호 사용 가능함

```
〉〉〉 A = {1,2,3,4,5}
〉〉〉 B = {2,4,6,8}
〉〉〉 C = A | B       # C = A ∪ B
〉〉〉 D = A & B       # C = A ∩ B
〉〉〉 E = A - B       # C = A - B
```

```
〉〉〉 print(A, B)       # A, B는 그대로예요.
{1, 2, 3, 4, 5} {8, 2, 4, 6}
〉〉〉 print(C)
{1, 2, 3, 4, 5, 6, 8}
〉〉〉 print(D)
{2, 4}
〉〉〉 print(E)
{1, 3, 5}
```

10. 집합 안에 for 반복문 사용하기 (Set Comprehension)

- 기본적으로 list comprehension과 같음.
- 결과가 중복된 데이터를 없고, 순서 개념이 없는 집합임.

CODE 62 리스트 L에는 단어들이 여러 개 있는데, 이 단어들을 모두 대문자로 바꾸어서 집합에 넣으려고 함. (집합에서는 저장 순서가 중요하지 않음)

```
L = ['grape', 'banana', 'apple', 'orange']
S = { x.upper( ) for x in L }      # 리스트에 있는 단어들을 모두 대문자로 바꿉니다.
print(S)
```

[결과]

{'BANANA', 'ORANGE', 'GRAPE', 'APPLE'}

10. 집합 안에 for 반복문 사용하기 (Set Comprehension)

CODE 63 이차 함수 y = x2 + 3의 y 값을 구하는 set comprehension을 이용한 코드. 정의역 x의 범위는 −5 ≦ x ≦ 5 임.

```
y = { x**2+3 for x in range(-5, 6) }
print(y)
```

[결과]

{3, 4, 7, 12, 19, 28}

11. frozenset 자료형

- frozenset 자료형은 고정된(수정할 수 없는) 집합
  - frozenset은 immutable set임.
  - frozenset 객체에 저장할 수 있는 메소드 목록을 보면 update가 붙는 메소드가 없음.

```
〉〉〉 dir(frozenset)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_
subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__',
'__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__xor__', 'copy', 'difference', 'intersection', 'isdisjoint', 'issubset', 'issuperset',
'symmetric_difference', 'union']
```

11. frozenset 자료형

```
>>> S = {1, 3, 5, 7, 9}
>>> T = frozenset(S)            # T는 immutable 집합이 됩니다.
>>> T.update({2,4,6})           # fronzenset에는 update( ) 메소드가 없습니다.
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    T.update({2,4,6})
AttributeError: 'frozenset' object has no attribute 'update'
>>> W = T.union({2, 4, 6})      # W = T ∪ {2, 4, 6}
>>> print(T)                    # fronzenset은 출력하면 다음과 같이 나옵니다.
frozenset({1, 3, 5, 7, 9})
>>> print(W)
frozenset({1, 2, 3, 4, 5, 6, 7, 9})
```

12. 정리

- 집합은 mutable 자료형으로 중복이 없음.
- 집합은 순서 개념이 없음.
- 집합 자료형은 교집합, 합집합, 차집합 등의 집합 연산을 쉽게 할 수 있는 메소드들을 제공함.
- Set Comprehension으로 집합을 구성할 수 있음.