

파이썬 프로그래밍

튜플과 집합



한국기술교육대학교
온라인평생교육원

■ 집합 자료형

1. 집합 자료형 생성

- set 내장 함수를 사용한 집합 자료 생성
 - 변경 가능(Mutable)한 객체이다.
 - 각 원소간에 순서는 없다.
 - 각 원소는 중복될 수 없다.
 - [note] 시퀀스 자료형이 아니다.

- set 내장함수를 활용하여 생성하는 자료형
- 순서가 없음 = 시퀀스 자료형이 아님 → 인덱싱, 슬라이싱 X
- 1, 2, 2, 3 = 1, 2, 3

■ 집합 자료형

1. 집합 자료형 생성

```
a = set([1, 2, 3])
print type(a)
print a
print

b = set((1, 2, 3))
print type(b)
print b
print

c = set({'a':1, 'b':2, 'c':3})
print type(c)
print c
print

d = set({'a':1, 'b':2, 'c':3}.values())
print type(d)
print d
```

```
<type 'set'>
set([1, 2, 3])

<type 'set'>
set([1, 2, 3])

<type 'set'>
set(['a', 'c', 'b'])

<type 'set'>
set([1, 2, 3])
```

■ 집합 자료형

1. 집합 자료형 생성

- set 내장함수 인자 → 리스트, 튜플, 사전 가능
- set 안에 튜플이 들어가도 출력은 리스트로 됨
- 인자에 사전이 들어가면 value는 무시되고 key 값만 가져옴
- 사전에는 key 값이 value보다 중요한 역할
- Set 안에 사전이 들어가도 출력은 리스트
- , value() 사용 시 → 사전의 key값 무시되고 value 값 출력

■ 집합 자료형

1. 집합 자료형 생성

- set의 원소로는 변경 불가능(Immutable)한 것만 할당 가능하다.

```
print set() # 빈 set 객체 생성
print set([1,2,3,4,5]) # 초기 값은 일반적으로 시퀀스 자료형인 리스트를 넣어준다.
print set([1,2,3,2,3,4]) # 중복된 원소는 한 번만 표현
print set('abc') # 문자열은 각 문자를 집합 원소로 지닌다.
print set([(1,2,3),(4,5,6)]) # 각 튜플은 원소로 가질 수 있음
print set([[1,2,3],[4,5,6]]) # 변경 가능 자료인 리스트는 집합의 원소가 될 수 없다.
```

```
[set()]
set([1, 2, 3, 4, 5])
set([1, 2, 3, 4])
set(['a', 'c', 'b'])
set([(4, 5, 6), (1, 2, 3)])
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-1-7f19fe538c1f> in <module>()
      4 print set('abc')                # 문자열은 각 문자를 집합 원소로 지닌다.
      5 print set([(1,2,3),(4,5,6)])    # 각 튜플은 원소로 가질 수 있음
----> 6 print set([[1,2,3],[4,5,6]])  # 변경 가능 자료인 리스트는 집합의 원소가 될 수 없다.

TypeError: unhashable type: 'list'
```

- Set의 원소로 변경 불가능한 것만 가능
- 중복된 원소는 하나로 인식
- set 내장함수 인자로 문자열도 가능 → 문자 하나하나가 나누어짐
- 리스트는 변경 가능한 자료이므로 원소 X

■ 집합 자료형

1. 집합 자료형 생성

- set의 기본 연산

set 연산	동일 연산자	내용
len(s)		원소의 개수
x in s		x가 집합 s의 원소인가?
x not in s		x가 집합 s의 원소가 아닌가?

```
A = set([1,2,3,4,5,6,7,8,9])
```

```
print len(A)    # 집합의 원소의 수
```

```
print 5 in A    # 멤버십 테스트
```

```
print 10 not in A # 멤버십 테스트
```

```
9
True
True
```

- 기본 연산 = 슬라이싱, 인덱싱..... → set 적용 불가능
- len(A) → A 집합의 원소 개수
- in, not in → 원소가 존재 하는지 하지 않는지 확인

■ 집합 자료형

2. 집합 자료형 메소드

- set의 주요 메소드
 - 다음 연산은 원래 집합은 변경하지 않고 새로운 집합을 반환한다.

set 연산	동일 연산자	내용
S issubset(t)	s <= t	s가 t의 부분집합인가?
S issuperset(t)	s >= t	s가 t의 슈퍼집합인가?
s union(t)	s t	새로운 s와 t의 합집합
s intersection(t)	s & t	새로운 s와 t의 교집합
s difference(t)	s-t	새로운 s와 t의 차집합
s symmetric_difference(t)	s * t	새로운 s와 t의 배타집합
s copy()		집합 s의 shallow 복사

```
B = set([4,5,6,10,20,30])
```

```
C = set([10,20,30])
```

```
print C.issubset(B) # C가 B의 부분집합?
```

```
print C <= B
```

```
print B.issuperset(C) # B가 C를 포함하는 집합?
```

```
print B >= C
```

```
print
```

```
True
```

```
True
```

```
True
```

```
True
```

■ 집합 자료형

2. 집합 자료형 메소드

- C의 원소들이 B의 원소에 포함됨
- `C.issubset(B)` → C가 B의 부분집합인지를 확인
- `.issubset = '<='`
- `C.issuperset(B)` → B가 C를 포함되는지 확인
- `.issuperset = '>='`

```
A = set([1,2,3,4,5,6,7,8,9])
```

```
B = set([4,5,6,10,20,30])
```

```
print A.union(B) # A와 B의 합집합
```

```
print A
```

```
print
```

```
print A.intersection(B) # A와 B의 교집합
```

```
print A
```

```
print
```

```
print A.difference(B) # A - B (차집합)
```

```
print A
```

```
print
```

```
print A.symmetric_difference(B) # 베타집합. A와 B의 합집합에서 교집합의 원소를  
제외한 집합
```

```
print A
```


■ 집합 자료형

2. 집합 자료형 메소드

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30])  
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([4, 5, 6])  
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([1, 2, 3, 7, 8, 9])  
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([1, 2, 3, 7, 8, 9, 10, 20, 30])  
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- `A.union(B)` → A와 B의 합집합
- `union`을 쓰면 합쳐서 반환되는 것이지 A는 변경 X
- `intersection` → 교집합
- `A.difference(B)` → 차집합(A-B)
- `A.symmetric_difference(B)` → 베타집합
- 베타집합 = A와 B의 합집합에서 교집합 원소를 제외한 집합

■ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4,5,6,7,8,9])
```

```
B = set([4,5,6,10,20,30])
```

```
print A | B # A와 B의 합집합
```

```
print A
```

```
print
```

```
print A & B # A와 B의 교집합
```

```
print A
```

```
print
```

```
print A - B #A - B (차집합)
```

```
print A
```

```
print
```

```
print A.symmetric_difference(B) # 베타집합. A와 B의 합집합에서 교집합의 원소를  
                                제외한 집합
```

```
print A ^ B
```

```
print A
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30])
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([4, 5, 6])
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([1, 2, 3, 7, 8, 9])
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
set([1, 2, 3, 7, 8, 9, 10, 20, 30])
```

```
set([1, 2, 3, 7, 8, 9, 10, 20, 30])
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

▣ 집합 자료형

2. 집합 자료형 메소드

- union = ' | ' (오와 연산)
- intersection = ' & '
- difference = ' - '
- symmetric difference → ' ^ '

■ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4,5,6,7,8,9])
D = A.copy()
print D
print

print A == D #자료값 비교
print A is D #객체 동등성 비교
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

True
False
```

- `D = A.copy()` A 내용은 그대로 가져오나 D라는 새로운 집합 생성
- `A == D` → A와 D의 자료값 비교
- `is` → 객체 동등성 비교
- `copy`는 객체가 새로 생성되는 것이므로 A와 B는 다른 객체

■ 집합 자료형

2. 집합 자료형 메소드

- set은 시퀀스 자료형이 아니므로 인덱싱, 슬라이싱, 정렬 등을 지원하지 않는다.

```
A = set([1,2,3,4,5,6,7,8,9])  
print A[0]
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-9-a0fca37959a3> in <module>()  
      1 A = set([1,2,3,4,5,6,7,8,9])  
----> 2 print A[0]  
  
TypeError: 'set' object does not support indexing
```

- A[0] → 인덱싱 연산 → A에서 0번째 인덱스 반환
- set은 시퀀스 자료형이 아니라 순서가 없음

■ 집합 자료형

2. 집합 자료형 메소드

```
print A[1:4]
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-13-795e14598526> in <module>()  
----> 1 print A[1:4]  
  
TypeError: 'set' object has no attribute '__getitem__'
```

- 슬라이싱 연산도 지원되지 않음

```
print A.sort()
```

```
-----  
AttributeError                            Traceback (most recent call last)  
<ipython-input-11-9fbe6777337f> in <module>()  
----> 1 print A.sort()  
  
AttributeError: 'set' object has no attribute 'sort'
```

- sort() 도 마찬가지로 지원되지 않음

■ 집합 자료형

2. 집합 자료형 메소드

- 집합을 리스트나 튜플로 변경가능
 - 집합에 인덱싱, 슬라이싱, 정렬 등을 적용하기 위해서는 리스트나 튜플로 변경한다.

```
print list(A)
print tuple(A)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

- list, tuple 함수 지원

- 하지만 집합에 for ~ in 연산은 적용 가능하다.

```
A = set([1,2,3,4,5,6,7,8,9])
for ele in A:
    print ele,
```

```
1 2 3 4 5 6 7 8 9
```

- for문은 지원

■ 집합 자료형

2. 집합 자료형 메소드

- set은 변경 가능(Mutable)한 자료 구조 객체
- 다음 메소드들은 set을 변경하는 집합 자료 구조 메소드들임

set 연산	동일 연산자	내용
s.update(t)	s = t	s와 t의 합집합을 s에 저장
s.intersection_update(t)	s &= t	s와 t의 교집합을 s에 저장
s.difference_update(t)	s -= t	s와 t의 차집합을 s에 저장
s.symmetric_difference_update(t)	s ^= t	s와 t의 배타집합을 s에 저장
s.add(x)		원소 x를 집합 s에 추가
s.remove(x)		원소 x를 집합 s에서 제거, 원소 x가 집합 s에 없으면 예외 발생
s.discard(x)		원소 x를 집합 s에서 제거
s.pop()		임의의 원소를 집합 s에서 제거, 집합 s가 공집합이면 예외 발생
s.clear()		집합 s의 모든 원소 제거

- update → . 앞에 있는 집합 객체의 내용 변경

■ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4])
B = set([3,4,5,6])

A.update(B) # A에 B 집합의 원소를 추가 시킴
print A

A.intersection_update([4,5,6,7,8]) # &=
print A

A.difference_update([6,7,8]) # -=
print A

A.symmetric_difference_update([5,6,7]) # ^=
print A

A.add(8) # 원소 추가
print A

A.remove(8) # 원소 제거
print A
```

```
set([1, 2, 3, 4, 5, 6])
set([4, 5, 6])
set([4, 5])
set([4, 6, 7])
set([8, 4, 6, 7])
set([4, 6, 7])
```

■ 집합 자료형

2. 집합 자료형 메소드

- `update` → A에 B의 내용 추가
- A와 B에 중복으로 있는 내용은 제외하고 추가
- `update = union` (합집합)
- `update`는 `union`과 다르게 A가 변경된다는 것이 중요
- `intersection_update` → 교집합의 결과가 반영됨
- `_update` 없으면 A는 변경 X
- `_update`가 있어야 A 변경 가능
- `symetric_difference_update` → 합집합에서 교집합 내용 빼기
- `add()` → 단순히 원소를 더하기
- `remove()` → 단순히 원소 없애기
- ' `s.update(t)` ' = ' `s |= t` '
- 'A. `intersection_update(t)`' = ' `A &= t` '
- `difference_update` = ' `-=` '
- `symmetric_difference_update` = ' `^=` '

■ 집합 자료형

2. 집합 자료형 메소드

A.remove(10) # 없는 원소를 제거하면 KeyError 발생

```
-----  
KeyError                                Traceback (most recent call last)  
<ipython-input-21-1bcb28846ad3> in <module>()  
----> 1 A.remove(10) #없는 원소를 제거하면 KeyError 발생  
  
KeyError: 10
```

- remove는 원소가 없으면 error, discard는 error X

A.discard(10) # remove와 같으나 예외가 발생하지 않음

A.discard(6) # 원소 6제거

print A

A.pop() # 임의의 원소 하나 꺼내기

print A

A = set([1,2,3,4])

A.clear() # 원소들 없애기

print A

```
set([4, 7])  
set([7])  
set([])
```

- pop() → 임의의 원소 하나 꺼내기 (랜덤으로 꺼내기)
- clear() → 원소 모두 없애기