기초 PYTHON 프로그래밍

10. 집합(set), 사전(dict)

1. 집합 (set)
2. 집합 사용하기
3. 사전 (dict)
4. 사전 사용하기
5. 사전과 for 반복문

1. 집합 (set)

- 집합은 **중복된 데이터를 가질 수 없고 순서가 없는** 데이터 구조이다.
- 인덱스 기호 ([ ]), +, * 를 사용할 수 없다.
- in, not in, len()은 사용할 수 있다.
- mutable 자료형이다.
- 집합 생성하기

```
>>> s = {1,2,5}      # set 생성
>>> print(s)
{1, 2, 5}
>>> type(s)
<class 'set'>
```

```
>>> s = {1,4,3,5,2,4,3}
>>> len(s)
5
>>> print(s)
{1, 2, 3, 4, 5}
```

1. 집합 (set)

- 빈 집합 생성하기 - 반드시 set() 함수를 이용해야 한다.

```
>>> S = set()   # 빈 집합 생성
>>> type(S)
<class 'set'>
```

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

- in / not in

```
>>> s = {2,3,5,7,11}
>>> 5 in s
True
>>> 7 not in s
False
>>> 10 not in s
True
```

집합에는 immutable 자료형 객체만 저장할 수 있다. (리스트, 집합, 사전은 넣을 수 없다)

2. 집합 사용하기

메소드	설명
add (x)	집합에 원소 x를 추가한다.
clear()	공집합으로 만든다.
copy ()	집합을 복사한다.
discard (x)	집합에서 원소 x를 삭제한다. 없는 원소를 삭제하려고 할 때에도 에러를 발생하지 않는다. (remove와 비교)
pop()	집합에서 임의의 원소를 하나 가져온다. 어떤 원소를 가져올지 알 수 없다. 집합에서 그 원소는 삭제된다. (KeyError)
remove (x)	집합에서 원소 x를 삭제한다. 없는 원소를 삭제하려고 하면 에러가 발생한다. (KeyError)

```
〉〉〉 dir(set)
['__and__', '__class__', …… '__xor__', 'add', 'clear', 'copy', 'difference',
'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint',
'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference',
'symmetric_difference_update', 'union', 'update']
```

2. 집합 사용하기

- 집합에 원소 추가하기 - add() 메소드

```
>>> A = {4,2,6,8,3}
>>> print(A)
{8, 2, 3, 4, 6}
>>> A.add(5)        # 집합 A에 데이터 5를 추가한다.
>>> print(A)
{2, 3, 4, 5, 6, 8}
>>> A.add(3)        # 이미 있는 데이터를 추가하면 변동이 없다.
>>> print(A)
{2, 3, 4, 5, 6, 8}
```

2. 집합 사용하기

- 집합에서 원소 삭제하기 - discard() 메소드

```
>>> A = {4,6,2,5,3}
>>> A.discard(5)
>>> print(A)
{2, 3, 4, 6}
>>> A.discard(7)     # 집합 A에 없는 원소 삭제해도 에러 없다.
>>> print(A)
{2, 3, 4, 6}
```

2. 집합 사용하기

- 집합에서 원소 삭제하기 - remove() 메소드

```
>>> B = {4, 2, 9, 7, 3}
>>> B.remove(2)
>>> B
{3, 4, 7, 9}
>>> B.remove(5)   # 없는 원소를 remove()하면 KeyError 발생함.
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    B.remove(5)
KeyError: 5
```

2. 집합 사용하기

◆ 집합에서 원소 삭제하기 / 공집합 만들기

```
>>> A = {1,3,5,6,4}
>>> A.pop()  # pop 메소드는 집합에서 임의의 원소를 반환하고 삭제한다.
1
>>> print(A)
{3, 4, 5, 6}
>>> A.clear()
>>> print(A)
set()
```

```
>>> B = {5, 7}
>>> B.pop()    # B에서 임의의 원소 반환하고 B에서 삭제함.
5
>>> B.pop()
7
>>> B
Set()
>>> B.pop()    # 공집합에 pop() 메소드 적용하면 KeyError
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in <module>
    B.pop()
KeyError: 'pop from an empty set'
```

2. 집합 사용하기

- 집합과 for 반복문

```
primes = {2,3,5,7,11,13}
for n in primes:
    print('prime :', n)
```

```
prime : 2
prime : 3
prime : 5
prime : 7
prime : 11
prime : 13
```

3. 사전 (dict)

- 사전은 집합의 일종이다 (순서 개념이 없다).
- +, * 를 사용할 수 없다. 인덱스 기호([])는 사용한다.
- in, not in, len()은 사용할 수 있다.
- 사전에는 (키:값)의 쌍으로 하나의 데이터가 저장된다.
- 사전에 키는 유일해야 한다 (중복된 키가 없어야 한다).
- 사전 예
  - 1반 25명, 2반 30명, 3반 27명이라는 정보를 저장하고자 한다.

```
〉〉〉 number = {1:25, 2:30, 3:27}
〉〉〉 type(number)
〈class 'dict'〉
```

3. 사전 (dict)

- 사전의 키
  - mutable 자료형은 '키'가 될 수 없다. (리스트, 집합, 사전)
  - 정수, 실수, bool, 복소수, 문자열, 튜플은 '키'가 될 수 있다.
- 사전의 값
  - 모든 자료형이 사전의 '값'이 될 수 있다.
- 빈 사전 생성하기

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

3. 사전 (dict)

◆ 사전에 아이템 추가 / 수정하기

```
>>> colorpen = {'red':2, 'blue':3, 'yellow':1}
>>> colorpen['green'] = 3    # 추가하기
>>> print(colorpen)
{'green': 3, 'red': 2, 'yellow': 1, 'blue': 3}
>>> colorpen['red'] = 4        # 수정하기
>>> print(colorpen)
{'green': 3, 'red': 4, 'yellow': 1, 'blue': 3}
```

3. 사전 (dict)

- 사전에 아이템 삭제하기 / in, not in, len()

```
>>> del colorpen['red']      # 삭제하기
>>> colorpen
{'yellow': 1, 'green': 4, 'blue': 3}
>>> len(colorpen)
3
>>> 'blue' in colorpen
True
>>> 'red' not in colorpen
True
```

4. 사전 사용하기

```
〉〉〉 dir(dict)
['__class__', '__contains__', …… '__subclasshook__', 'clear',
'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem',
'setdefault', 'update', 'values']
```

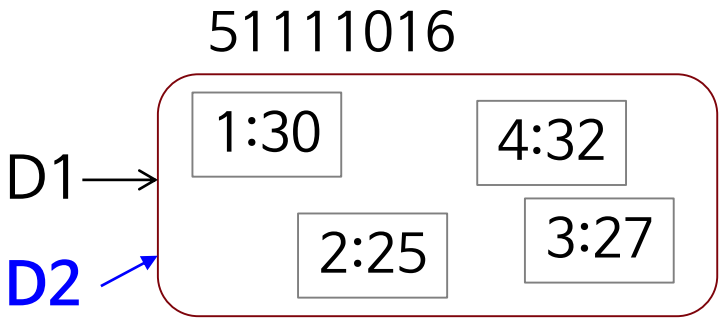
- clear() 메소드

```
〉〉〉 D = {1:30, 2:25, 3:27}
〉〉〉 print(D)
{1: 30, 2: 25, 3: 27}
〉〉〉 D.clear()
〉〉〉 print(D)
{}
```

4. 사전 사용하기

- copy() 메소드

```
>>> D1 = {1:30, 2:25, 3:27}
>>> id(D1)
51111016
>>> D2 = D1
>>> id(D2)
51111016
>>> D2[4] = 32
>>> print(D1)
{1: 30, 2: 25, 3: 27, 4: 32}
>>> print(D2)
{1: 30, 2: 25, 3: 27, 4: 32}
```




```
>>> D1 = {1:30, 2:25, 3:27}
>>> D2 = D1.copy()
>>> D2[4] = 32
>>> print(D1)
{1: 30, 2: 25, 3: 27}
>>> print(D2)
{1: 30, 2: 25, 3: 27, 4: 32}
```

4. 사전 사용하기

- fromkeys() 메소드
  - '키'들을 리스트로 모아 두었다가 '사전'을 쉽게 만들 수 있다.

```
>>> keys = ['파이썬', '자바', '아두이노']
>>> books = dict.fromkeys(keys)
>>> print(books)
{'파이썬': None, '자바': None, '아두이노': None}
>>> books2 = dict.fromkeys(keys, 0)
>>> print(books2)
{'파이썬': 0, '자바': 0, '아두이노': 0}
>>> books3 = dict.fromkeys(keys, 5)
>>> print(books3)
{'파이썬': 5, '자바': 5, '아두이노': 5}
```

4. 사전 사용하기

- get(x) 메소드 - x에는 '키'가 입력되고 x의 값을 반환한다.

```
>>> pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
>>> v = pencil.get('blue')
>>> print(v)
5
```

```
>>> pencil['blue']
5
```

[참고] 사전.get(x)와 사전[x]의 차이점 - 없는 '키' 사용할 때

```
>>> v = pencil.get('white')
>>> print(v)
None
>>> pencil['white']
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    pencil['white']
KeyError: 'white'
```

4. 사전 사용하기

- items() 메소드
  - 사전에 있는 모든 키와 값을 튜플로 묶어서 dict_items 객체로 반환한다.

```
>>> tests = {'toeic':40, 'toefl':27, 'gre':12}
>>> x = tests.items()
>>> print(x)
dict_items([('toeic', 40), ('toefl', 27), ('gre', 12)])
>>> type(x)
<class 'dict_items'>
>>> y = list(tests.items())
>>> print(y)
[('toeic', 40), ('toefl', 27), ('gre', 12)]
```

4. 사전 사용하기

- keys() 메소드
  - 사전에 있는 키들을 리스트로 묶어서 dict_keys 객체로 반환한다.

```
>>> tests = {'toeic':40, 'toefl':27, 'gre':12}
>>> k = tests.keys()
>>> print(k)
dict_keys(['toeic', 'toefl', 'gre'])
>>> type(k)
<class 'dict_keys'>
>>> m = list(tests.keys())
>>> print(m)
['toeic', 'toefl', 'gre']
```

4. 사전 사용하기

- pop(k,v) 메소드
  - 사전에서 k를 키로 갖는 원소를 삭제하고 키에 해당하는 값을 반환한다.
  - 사전에 없는 키를 k에 입력하면 에러가 발생한다.

```
>>> math_test = {'Alice':90, 'Paul':88, 'David':75, 'Cindy':93}
>>> x = math_test.pop('David')
>>> print(x)
75
>>> y = math_test.pop('Tom')
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    y = math_test.pop('Tom')
KeyError: 'Tom'
>>> y = math_test.pop('Tom', 70)
>>> print(y)
70
```

4. 사전 사용하기

- popitem() 메소드
  - 컴퓨터가 사전에서 임의의 키를 선택하여 해당하는 값을 반환하고 사전에서 삭제한다. 빈 사전에 popitem() 메소드를 적용하면 에러가 발생한다.

```
>>> data = {'name':'Alice', 'age':20, 'school':'sogang'}
>>> x = data.popitem()
>>> print(x)
('school', 'sogang')
>>> print(data)
{'name': 'Alice', 'age': 20}
>>> data = {}
>>> x = data.popitem()
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    x = data.popitem()
KeyError: 'popitem(): dictionary is empty'
```

4. 사전 사용하기

- setdefault() 메소드
  - 사전에 원소를 추가할 수 있도록 하는 메소드이다. 만약에 k가 사전에 있는 '키'이면 해당하는 '값'을 반환하고 사전의 내용은 변하지 않는다.
  - 만약에 k가 사전에 없는 '키'이면 사전에 k를 추가한다. 해당하는 '값'은 None으로 추가된다.
  - 만약에 k가 사전에 없는 '키'인 경우 두 번째 인자로 defaul로 사용할 '값'을 넣어주면 해당하는 값으로 초기화된 원소가 추가된다.

4. 사전 사용하기

- setdefault() 메소드

```
>>> pencil = {'white':5, 'black':3, 'red':7}
>>> pencil.setdefault('red')
7

>>> pencil.setdefault('blue')
>>> print(pencil)
{'white': 5, 'black': 3, 'red': 7, 'blue': None}

>>> pencil.setdefault('brown', 2)
2
>>> print(pencil)
{'white': 5, 'black': 3, 'red': 7, 'blue': None, 'brown': 2}

>>> pencil.setdefault('black', 6)
3
>>> print(pencil)
{'white': 5, 'black': 3, 'red': 7, 'blue': None, 'brown': 2}
```

4. 사전 사용하기

- update(D) 메소드
  - 인자 D는 사전이어야 한다. update를 호출하는 사전과 D를 합한다.

```
>>> Voca = {1:'one', 2:'two', 3:'three'}
>>> Voca2 = {100:'hundred', 1000:'thousand'}
>>> Voca.update(Voca2)
>>> print(Voca)
{1: 'one', 2: 'two', 3: 'three', 1000: 'thousand', 100: 'hundred'}
>>> print(Voca2)
{1000: 'thousand', 100: 'hundred'}
```

4. 사전 사용하기

- values()
  - 사전에서 값만을 dict_values 객체로 반환한다.

```
>>> tests = {'toeic':30, 'toefl':20, 'opic':25}
>>> x = tests.values()
>>> print(x)
dict_values([30, 20, 25])
>>> type(x)
<class 'dict_values'>
>>> y = list(tests.values())
>>> print(y)
[30, 20, 25]
```

5. 사전과 for 반복문

- keys() 메소드와 for 반복문

  사전 score에는 '학번:성적' 으로 구성된다.

```
score = {20181234:90, 20181111:80, 20180015:85, 20181007:93}


for k in score.keys():
    print(k)
```

```
20181234
20181111
20180015
20181007
```

5. 사전과 for 반복문

- values() 메소드와 for 반복문

  사전 score에는 '학번:성적' 으로 구성된다.

```
score = {20181234:90, 20181111:80, 20180015:85, 20181007:93}


for v in score.values():
    print(v)
```

```
90
80
85
93
```

5. 사전과 for 반복문

- items() 메소드와 for 반복문

  사전 score에는 '학번:성적' 으로 구성된다.

```
score = {20181234:90, 20181111:80, 20180015:85, 20181007:93}


for k,v in score.items():
    print('학번 : {}, 성적 : {}'.format(k,v))
```

```
학번 : 20181234, 성적 : 90
학번 : 20181111, 성적 : 80
학번 : 20180015, 성적 : 85
학번 : 20181007, 성적 : 93
```