딕셔너리 메소드 활용

추가 및 삭제

`.pop(key[, default])`

key가 딕셔너리에 있으면 제거하고 그 값을 돌려줍니다. 그렇지 않으면 default를 반환합니다.

default가 없는 상태에서 딕셔너리에 없으면 KeyError가 발생합니다.

In [1]:

```
my_dict = {'apple' : '사과', 'banana' : '바나나'}
```

In [2]:

```
my_dict.pop('apple')
```

Out[2]:

```
'사과'
```

In [4]:

```
my_dict.pop('melon')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-4-2a8b4f967f4a> in <module>
----> 1 my_dict.pop('melon')

KeyError: 'melon'
```

In [5]:

```
print(my_dict)
```

```
{'banana': '바나나'}
```

In [3]:

```
my_dict.pop('melon', 0)
```

Out[3]:

```
0
```

`.update()`

- key, value 페어를 추가합니다.
- 만약 key 가 존재한다면, value 를 덮어씁니다.

In [ ]:

```
my_dict = {'apple' : '사과', 'banana' : '바나나', 'melon' : '멜론'}
my_dict.update({'pear' : '배'})
print(my_dict)
```

In [6]:

```
my_dict = {'apple' : '사과', 'banana' : '바나나', 'melon' : '멜론'}
my_dict.update({'apple' : '사과아아아'})
print(my_dict)
```

```
{'apple': '사과아아아', 'banana': '바나나', 'melon': '멜론'}
```

`.get(key[, default])`

key를 통해 value를 가져옵니다.

절대로 KeyError가 발생하지 않습니다. default는 기본적으로 None입니다.

In [7]:

```
my_dict = {'apple' : '사과', 'banana' : '바나나', 'melon' : '멜론'}
my_dict['pineapple']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-bf4f1e674618> in <module>
      1 my_dict = {'apple' : '사과', 'banana' : '바나나', 'melon' : '멜론'}
----> 2 my_dict['pineapple']

KeyError: 'pineapple'
```

In [8]:

```
my_dict.get('pineapple')
```

In [9]:

```
my_dict.get('apple')
```

Out[9]:

'사과'

In [10]:

```
my_dict.get('pineapple', 1)
```

Out[10]:

1

dictionary comprehension

dictionary도 comprehension을 활용하여 만들 수 있습니다.

In [11]:

```
cubic = {x: x**3 for x in range(1, 8)}
print(cubic)
```

```
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343}
```

In [16]:

```
# 다음의 딕셔너리에서 미세먼지 농도가 80 초과 지역만 뽑아 봅시다.
# 예) {'경기': 82, '부산': 90}
dust = {'서울' : 72, '경기': 82, '대전': 29, '중국': 200}
dust_air = {key: value for key, value in dust.items() if value > 80 }
print(dust_air)
```

```
{'경기': 82, '중국': 200}
```

In [17]:

```
# 다음의 딕셔너리에서 미세먼지 농도가 80초과는 나쁨 80이하는 보통으로 하는 value를 가지도록 바꿔봅시다.
# 예) {'서울': '나쁨', '경기': '보통', '대전': '나쁨', '부산': '보통'}
dust = {'서울' : 72, '경기': 82, '대전': 29, '중국': 200}
dust_air = {key: '나쁨' if value > 80 else '보통' for key, value in dust.items()}
print(dust_air)
```

```
{'서울': '보통', '경기': '나쁨', '대전': '보통', '중국': '나쁨'}
```

In [ ]:

```
# 만약 elif 말해주면 이렇게 말해주자^_^ 강사용
{key: '매우나쁨' if value > 150 else '나쁨'
                 if value > 80 else '보통'
                 if value > 30 else '좋음' for key, value in dust.items()}
```

정리! `map()`, `zip()`, `filter()`

`map(function, iterable)`

ex) `list(map(함수, 리스트))`

- `map` 은 Iterable 의 요소를 지정된 함수로 처리해주는 함수
- 대표적으로 iterable한 타입 - list, dict, set, str, bytes, tuple, range
- return은 `map_object` 형태로 됩니다.
- map 은 원본을 변경하지 않고 새 값을 생성합니다.

for 문으로 반복하면서 요소를 변환하기 어려울 때, map 을 사용하면 편리합니다.

In [20]:

```
# a 리스트를 문자열 '123'으로 만들어봅시다.
a = [1, 2, 3]
b = ''
for i in range(len(a)):
    b += str(a[i])
b
```

Out[20]:

```
'123'
```

In [22]:

```
# map 으로 문자열 '123'으로 만들어봅시다.
a = [1, 2, 3]
''.join(map(str, a))
```

Out[22]:

```
'123'
```

In [23]:

```
# comprehension
''.join([str(x) for x in a])
```

Out[23]:

```
'123'
```

In [27]:

```
a = ['1', '2', '3']
# 위의 코드를 map 으로 [1, 2, 3]으로 만들어봅시다.
list(map(int, a))
```

Out[27]:

```
[1, 2, 3]
```

In [29]:

```
# comprehension
[int(x) for x in a]
```

Out[29]:

```
[1, 2, 3]
```

In [31]:

```
# input 으로 받은 문자열 리스트의 요소들을 int 로 반환.
a, b = map(int, input().split())
```

```
10 30
```

- **function은 사용자 정의 함수도 가능하다!**

In [32]:

```
# 세제곱의 결과를 나타내는 함수를 만들어봅시다.
def cube(n):
    return n**3
```

In [34]:

```
a = [1, 2, 3]
list(map(cube, a))
```

Out[34]:

```
[1, 8, 27]
```

`zip(*iterables)`

- 복수 iterable한 것들을 모아준다.
- 결과는 튜플의 모음으로 구성된 zip object를 반환한다.

In [35]:

```
# 예시를 봅시다.
girls = ['jane', 'iu', 'mary']
boys = ['justin', 'david', 'kim']
list(zip(girls, boys))
```

Out[35]:

```
[('jane', 'justin'), ('iu', 'david'), ('mary', 'kim')]
```

In [39]:

```
# dictionary comprehension으로 한 명씩 순서대로 매칭시켜봅시다.
# 예) {'jane': 'justin', 'iu': 'david', 'mary': 'kim'}
girls = ['jane', 'iu', 'mary']
boys = ['justin', 'david', 'kim']
{x: y for x in girls for y in boys}
# 딕셔너리에서 key는 유일한 값 tutor로 확인해보기
```

Out[39]:

```
{'jane': 'kim', 'iu': 'kim', 'mary': 'kim'}
```

In [40]:

```
{x: y for x, y in zip(girls, boys)}
```

Out[40]:

```
{'jane': 'justin', 'iu': 'david', 'mary': 'kim'}
```

- 그리고 아래와 같이 사용가능하다.

In [43]:

```
a = '123'
b = '567'
for digit_a, digit_b in zip(a, b):
    print(digit_a, digit_b)
```

```
1 5
2 6
3 7
```

- zip은 반드시 길이가 같을 때 사용해야한다. 가장 짧은 것을 기준으로 구성한다.

In [44]:

```
num1 = [1, 2, 3]
num2 = ['1', '2']
list(zip(num1, num2))
```

Out[44]:

```
[(1, '1'), (2, '2')]
```

- 물론 길이가 긴 것을 맞춰서 할 수도 있지만, 기억 저 멀리 넣어놓자.

In [48]:

```
from itertools import zip_longest
list(zip_longest(num1, num2, fillvalue=0))
```

Out[48]:

```
[(1, '1'), (2, '2'), (3, 0)]
```

- unpack

In [56]:

```
letters = ['a', 'b', 'c']
nums = [1, 2, 3]

zip_list = list(zip(letters, nums))
print(zip_list)
```

```
[('a', 1), ('b', 2), ('c', 3)]
```

In [58]:

```
new_letters, new_nums = zip(*zip_list)
```

In [59]:

```
print(new_letters)
print(new_nums)
```

```
('a', 'b', 'c')
(1, 2, 3)
```

`filter(function, iterable)`

- iterable에서 function의 반환된 결과가 참인 것들만 구성하여 반환한다.

In [63]:

```
# 짝수인지 판단하는 함수를 작성해봅시다.
def even(n):
    return not n % 2
```

In [65]:

```
a = [1, 2, 3, 4]
list(filter(even, a))
```

Out[65]:

```
[2, 4]
```

In [67]:

```
# 다음의 list comprehension과 동일하다.
[x for x in [1, 2, 3, 4] if even(x)]
```

Out[67]:

```
[2, 4]
```

In [68]:

```
# 다음의 list comprehension과 동일하다.
[x for x in [1, 2, 3, 4] if not x % 2]
```

Out[68]:

```
[2, 4]
```

세트 메소드 활용

추가 및 삭제

`.add(elem)`

elem을 세트에 추가합니다.

In [70]:

```
a = {1, 2, 3, 4}
a.add(5)
a.add(5)    #중복 x
print(a)
```

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

update(*others)

여러가지의 값을 순차적으로 추가합니다.

여기서 반드시 iterable한 값을 넣어야합니다.

In [71]:

```
# * : iterable한 객체를 의미
a = {1, 2, 3}
a.update((5, 5, 5, 2), (7, 9))
print(a)
```

```
{1, 2, 3, 5, 7, 9}
```

.remove(elem)

elem을 세트에서 삭제하고, 없으면 KeyError가 발생합니다.

In [72]:

```
# 에러를 확인해봅시다.
a = {1, 2, 3}
a.remove(2)
print(a)
```

```
{1, 3}
```

In [73]:

```
a.remove(11)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-73-2f6a2b387944> in <module>
----> 1 a.remove(11)

KeyError: 11
```

discard(elem)

elem 를 세트에서 삭제하고 없어도 에러가 발생하지 않습니다.

In [76]:

```
a = {1, 2, 3}
a.discard(11)
print(a)
```

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

pop()

임의의 원소를 제거해 반환합니다.

In [87]:

```
a = {3, 4, 5, 6}
```

In [88]:

```
a.pop()
```

Out[88]:

```
3
```

In [89]:

```
print(a)
```

```
{4, 5, 6}
```