데이터 구조(Data Structure) II

- 알고리즘에 빈번히 활용되는 순서가 없는(unordered) 데이터 구조
  - 세트(Set)
  - 딕셔너리(Dictionary)

세트(Set)

> 변경 가능하고(mutable), 순서가 없고(unordered), 순회 가능한(iterable)

데이터 구조로서의 세트(set)와 조작법(method)

- https://docs.python.org/ko/3/library/stdtypes.html#set-types-set-frozenset

추가 및 삭제

.add(elem)

elem을 세트에 추가합니다.

In [ ]:
```
# add를 사용해봅시다.
a = {'사과', '바나나', '수박'}
```

In [ ]:
```
# 코드 입력
a.add('파인애플')  # 순서가 상관이 없다
print(a)
```

.update(*others)

여러가지의 값을 추가합니다.

인자로는 반드시 iterable 데이터 구조를 전달해야합니다.

In [ ]:
```
# update를 사용해봅시다.
a = {'사과', '바나나', '수박'}
```

In [ ]:
```
# 코드 입력
a.update(('파인애플','애플'))
print(a)
```

.remove(elem)

elem을 세트에서 삭제하고, 없으면 KeyError가 발생합니다.

In [ ]:
```
# remove를 사용해봅시다.
```

```
a = {'사과', '바나나', '수박'}
```

In [ ]:
```
# 코드 입력
a.remove('사과')
print(a)
```

In [ ]:
```
a.remove('파인애플')
print(a)
```

.discard(elem)

elem을 세트에서 삭제하고 없어도 에러가 발생하지 않습니다.

In [ ]:
```
# discard를 사용해봅시다.
a = {'사과', '바나나', '수박'}
```

In [ ]:
```
# 코드 입력
a.discard('파인애플')
print(a)
```

.pop()

임의의 원소를 제거해 반환합니다.

In [ ]:
```
# pop을 사용해봅시다.
a = {'사과', '바나나', '수박', '아보카도'}
```

In [ ]:
```
# 코드 입력
a.pop()
print(a)
```

딕셔너리(Dictionary)

> 변경 가능하고(mutable), 순서가 없고(unordered), 순회 가능한(iterable)
>
> `Key: Value` 페어(pair)의 자료구조

데이터 구조로서의 딕셔너리(dictionary)와 조작법(method)

- https://docs.python.org/ko/3/library/stdtypes.html#mapping-types-dict

조회

.get(key[, default])

key를 통해 value를 가져옵니다.

절대로 KeyError가 발생하지 않습니다. default는 기본적으로 None입니다.

In [ ]:
```
# get을 사용해봅시다.
my_dict = {'apple': '사과', 'banana': '바나나', 'melon': '멜론'}
my_dict['pineapple']
```

In [ ]:
```
my_dict.get('pineapple')   # None return
```

In [ ]:
```
my_dict.get('pineapple',0)   # 없으면 0을 리턴해라
```

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 [ ]:
```
# pop을 사용해봅시다.
my_dict = {'apple': '사과', 'banana': '바나나'}
```

In [ ]:
```
my_dict.pop('apple')
```

In [ ]:
```
my_dict
```

In [ ]:
```
# 딕셔너리에 없으면 에러가 발생합니다.
```

In [ ]:
```
my_dict.pop('pine')
```

In [ ]:
```
# 두번째 인자로 default를 설정할 수 있습니다.
```

In [ ]:
```
my_dict.pop('pine',0)
```

.update()

값을 제공하는 key, value로 덮어씁니다.

In [ ]:
```
# update를 사용해봅시다.
my_dict = {'apple': '사과', 'banana': '바나나', 'melon': '멜론'}
```

In [ ]:
```
my_dict.update({'banana':'빠나나'})
```

In [ ]:
```
my_dict
```

In [ ]:
```
```

```
my_dict.update(melon = '메론')
```

In [ ]:
```
my_dict
```

딕셔너리 순회(반복문 활용) 잘 알아두기!!

딕셔너리에 `for` 문을 실행하면 기본적으로 다음과 같이 동작합니다.

In [ ]:
```
grades = {'john':  80, 'eric': 90, 'justin': 90}
for student in grades:
    print(student)
```

딕셔너리의 **key**를 접근할 수 있으면 **value**에도 접근할 수 있기 때문입니다.

따라서 딕셔너리의 value를 출력하기 위해서는 아래와 같이 작성합니다.

In [ ]:
```
# 학생 이름(key)을 활용하여 점수(value)를 출력해봅시다.
```

In [ ]:
```
print(grades.keys())
print(grades.values())
print(grades.items())  # 튜플로 묶여있음
```

- dictionary에서 `for` 를 활용하는 4가지 방법

```
# 0. dictionary 순회 (key 활용)
for key in dict:
    print(key)
    print(dict[key])


# 1. `.keys()` 활용
for key in dict.keys():
    print(key)
    print(dict[key])


# 2. `.values()` 활용
for val in dict.values():
    print(val)


# 3. `.items()` 활용
for key, val in dict.items():
    print(key, val)
```

[연습] 딕셔너리 순회

> 혈액형 검사한 결과가 담긴 `blood_types` 이 주어졌을때, 해당 딕셔너리를 순회하며, `key` 와 `value` 를 출력해보세요.

[출력 예시]

```
A형은 40명입니다.
B형은 11명입니다.
AB형은 4명입니다.
O형은 45명입니다.
```

In [ ]:
```
blood_types = {'A': 40, 'B': 11, 'AB': 4, 'O': 45}
```

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
# key로 반복하는 코드를 작성해봅시다.
```

In [ ]:
```
for blood_type in blood_types:
    print(f'{blood_type}형은 {blood_types[blood_type]}명입니다.')
```

In [ ]:
```
# .keys()를 활용하여 반복하는 코드를 작성해봅시다.
```

In [ ]:
```
for blood_type in blood_types.keys():
    print(f'{blood_type}형은 {blood_types[blood_type]}명입니다.')
```

In [ ]:
```
# .items()를 활용하여 반복하는 코드를 작성해봅시다.
```

In [ ]:
```
for blood_type, number in blood_types.items():
    print(f'{blood_type}형은 {people}명입니다.')
```

[실습] 딕셔너리 순회

> 혈액형 검사한 결과가 담긴 `blood_types` 이 주어졌을때, 해당 검사에 참가한 사람들의 총합을 구해보세요.

[출력 예시]

```
검사에 참가한 사람은 총 100명입니다.
```

In [ ]:
```
# key로 반복하는 코드를 작성해봅시다.
```

In [ ]:
```
total = 0
for blood_type in blood_types:
    total += blood_types[blood_type]
print(f'검사에 참가한 사람은 총 {total}명입니다.')
```

In [ ]:
```
# .values()를 활용하여 작성해봅시다.  # 오 이런 방법이 !!
```

In [ ]:
```
total = 0
for number in blood_types.values():
    total += number
print(f'검사에 참가한 사람은 총 {total}명입니다.')
```

In [ ]:
```
# 더 간단하게 구할 수도 있습니다.
```

In [ ]:
```
total = sum(blood_types.values())  # 이런 방법이 !!
print(f'검사에 참가한 사람은 총 {total}명입니다.')
```

[응용] 딕셔너리 구축하기(counter)

> 리스트가 주어질 때, 각각의 요소의 개수를 value 값으로 갖는 딕셔너리를 만드세요.

[출력 예시]

{'great': 2, 'expectations': 1, 'the': 2, 'adventures': 2, 'of': 2, 'sherlock': 1, 'holmes': 1, 'gasby': 1, 'hamlet': 1, 'huckleberry': 1, 'fin': 1}

In [ ]:
```
book_title =  ['great', 'expectations', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'th

# 아래에 코드를 작성하세요.
```

In [ ]:
```
count = {'great': 2}
count['great'] += 1
```

In [ ]:
```
count
```

In [ ]:
```
count = {}
# 단어들을 반복하면서
for word in book_title:
    # count 딕셔너리에 해당하는 key의 value를 증가시킨다.
    count[word] = count[word] + 1
print(count)
```

In [ ]:
```
count = {}
for word in book_title:
    count[word] = count.get(word,0) + 1

print(count)
```

In [ ]:
```
book_title =  ['great', 'expectations', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'th
# 문제 발생!! key 있을 때랑 없을때를 나누자!
count = {}
# 단어들을 반복하면서
for word in book_title:
    # 만약에 키가 있으면,
    if word in count.keys():
        # count 딕셔너리에 해당하는 key의 value를 증가시킨다.
        count[word] = count[word] + 1
    # 없으면,
    else:
        count[word] = 1
count
```

In [ ]:
```
book_title =  ['great', 'expectations', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'th
# 문제 발생!! key 있을 때랑 없을때를 나누자!
count = {}
# 단어들을 반복하면서
for word in book_title:
    # 가지고 오는데 없으면 0..
    count[word] = count.get(word, 0) + 1
count
```

> get(key[, default])
>
> - key 가 딕셔너리에 있는 경우 key 에 대응하는 값을 돌려주고, 그렇지 않으면 default 를 돌려준다.

Dictionary comprehension

dictionary도 comprehension을 활용하여 만들 수 있습니다.

활용법

`iterable` 에서 `dict` 를 생성할 수 있습니다.

```
{키: 값 for 요소 in iterable}
```

```
dict({키: 값 for 요소 in iterable})
```

In [ ]:
```
# iterable(range) -> dict
```

In [ ]:
```
a = [1,2,3]
{str(n): n for n in a}
```

In [ ]:
```
# iterable(dict) -> dict
```

In [ ]:
```
blood_types = {'A': 40, 'B': 11, 'AB': 4, 'O': 45}
negative_blood_types = {'-' + key: value for key, value in blood_types.items()}
# negative_blood_types = {'-' + key: blood_types[key] for key in blood_types}
print(negative_blood_types)
```

Dictionary comprehension + 조건

List comprehension과 유사하게, 조건문에 참인 식으로 딕셔너리를 생성합니다.

활용법

```
{키: 값 for 요소 in iterable if 조건식}
```

```
{키: 값 if 조건식 else 값 for 요소 in iterable}
```

Dictionary comprehension 사용해보기

In [ ]:
```
dusts = {'서울': 72, '대전': 82, '구미': 29, '광주': 45, '중국': 200}
```

In [ ]:
```
# 미세먼지 농도가 80 초과 지역만 뽑아주세요.
# 예) {'대전': 82, '중국': 200}
```

In [ ]:
```
result = {key: value for key, value in dusts.items() if value > 80}
```

In [ ]:
```
print(result)
```

In [ ]:
```
# 미세먼지 농도가 80초과는 나쁨 80이하는 보통으로 하는 value를 가지도록 바꾸세요.
```

In [ ]:
```
result = {key: '나쁨' if value > 80 else '보통' for key, value in dusts.items()}
```

In [ ]:
```
print(result)
```

In [ ]:
```
# elif 도 사용할 수 있습니다. (if else 열거)
```

In [ ]:
```
result = {key: '매우나쁨' if value > 150 else '나쁨' if value > 80 else '보통' if value > 30
```

In [ ]:
```
print(result)
```