Python_06

데이터 구조(Data Structure)

자료구조 == 데이터 구조 + 연산

순서가 없는 데이터 구조

셋(Set) == 집합

- Set이란 `중복되는 요소 없이`, `순서에 상관없는 데이터` 들의 묶음
  - 데이터의 중복을 허용하지 않기 때문에 중복되는 원소가 있다면 하나만 저장
  - 순서가 없기 때문에 인덱스를 이용한 접근 불가능
- 수학에서의 집합을 표현한 컨테이너
  - 집합 연산이 가능(여집합을 표현하는 연산자는 별도로 존재X)
  - 중복된 값이 존재하지 않음
- 담고 있는 요소를 삽입, 변경, 삭제 가능 → 가변 자료형(mutable)

문법	설명
s.copy()	셋은 얕은 복사본을 반환
s.add(x)	항복 x가 셋 s에 없다면 추가
s.pop()	셋 s에서 랜덤하게 항목을 반환하고, 해당 항목을 제거 set이 비어 있을 경우, KeyError
s.remove(s)	항목 x를 셋 s에서 삭제 항목이 존재하지 않을 경우, KeyError
s.discard(t)	항복 x가 셋 s에 있는 경우, 항목 x를 셋 s에서 삭제 셋에서 삭제하고 없어도 에러가 발생하지 않음
s.update(t) s.update(*others)	셋 t에 있는 모든 항목 중 셋 s에 없는 항목을 추가
s.clear()	모든 항목을 제거
s.isdisjoint(t)	셋 s가 셋 t의 서로 같은 항목을 하나라도 갖고 있지 않은 경우, True반환(서로소)
s.issubset(t)	셋 s가 셋 t의 하위 셋인경우, True반환
s.issuperset(t)	셋 s가 셋 t의 상위 셋인경우, True반환

> **TIP**
>
> Q. set.pop() 이라는 메서드가 랜덤한 항목(임의의 원소) 를 제거한다고 설명이 되어있는데 아래 코드로 항상 사과가 반환되는 이유가 뭘까요?
>
> ```
> a = {'사과', '바나나', '수박'}
> print(a.pop()) # '사과'
> ```
>
> - 세트도 동일하게 내부 구조로 해시 테이블을 사용합니다.
> - 세트의 pop 메서드를 사용하면 랜덤한 항목이 제거되는 것이 맞습니다.

- 이러한 이유는 hash 함수를 사용하여 저장하는 구조에 있습니다.
- 동일한 값을 동일한 순서로 저장하는 경우에도 pop을 수행할 시에 동일한 값이 반환되는 것을 알 수 있습니다.

TIP

set은 immutable 데이터만 추가 가능
리스트는 hash를 생성할 수 없는 객체 =⇒ 리스트는 변경을 하면 hash 값이 바뀌므로

```
a = {'사과', '바나나', '수박'}
a.add('포도')

TypeError: unhashable type: 'list'
```

tuple은 가능!

```
a.add(('포도',))
# {'사과', ('포도',), '바나나', '수박'}
```

딕셔너리(Dict)

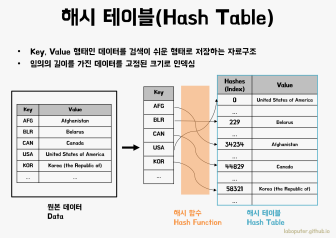
- 키-값(key-value)쌍으로 이뤄진 자료형
- Dictionary의 키(key)
  - key는 변경 불가능한 데이터(immutable)만 활용 가능
    - String, integer, float, boolean, tuple, range
- 각 키의 값(values)
  - 어떠한 형태는 관계없음

문법	설명
d.clear()	모든 항목을 제거
d.copy()	딕셔너리 d의 얕은 복사본을 반환
d.keys()	딕셔너리 d의 모든 키를 담은 뷰를 반환
d.values()	딕셔너리 d의 모든 값을 담은 뷰를 반환
d.items()	딕셔너리 d의 모든 키-값의 쌍을 담은 뷰를 반환
d.get(k)	키 k의 값을 반환하는데, 키 k가 딕셔너리 d에 없을 경우 None을 반환 KeyError가 발생하지 않으며 default값을 설정할 수 있음 `.get(key[,default])`
d.get(k, v)	키 k의 값을 반환하는데, 키 k가 딕셔너리 d에 없을 경우 v을 반환
d.pop(k)	키 k의 값을 반환하고 키 k인 항목을 딕셔너리 d에서 삭제하는데, 키 k가 딕셔너리 d에 없을 경우 KeyError를 발생 KeyError를 발생시키지 않으려면 `.pop(key[,default])`
d.pop(k, v)	키 k의 값을 반환하고 키 k인 항목을 딕셔너리 d에서 삭제하는데, 키 k가 딕셔너리 d에 없을 경우 V를 반환
d.update([other])	딕셔너리 d의 값을 매핑하여 업데이트

TIP

Q. 딕셔너리에서 키를 입력하면 입력한 키를 어떻게 동작해서 찾는 건가요?

- 딕셔너리는 내부 구조로 해시 테이블이라는 것을 사용합니다.
- 해시 테이블은 사용하는 값을 `hash()` 라는 함수를 사용해서 위치 정보 지정합니다. 위치 : `hash(객체) % 저장하는 테이블 크기`
- 이 위치 정보를 통해 탐색을 수행하고, 없다면 다음 요소를 순차적으로 탐색하는 방식입니다. (이 방식을 `체이닝 해시 테이블`이라고 함)

> 해시테이블
>
> 
>
>
> - key 값을 해시 함수를 통해 주소값으로 변환
> - 배열 [해시 값 % size] = '값'
> - keyError : 키값이 존재 하지 않는 곳에 접근

추가 예제

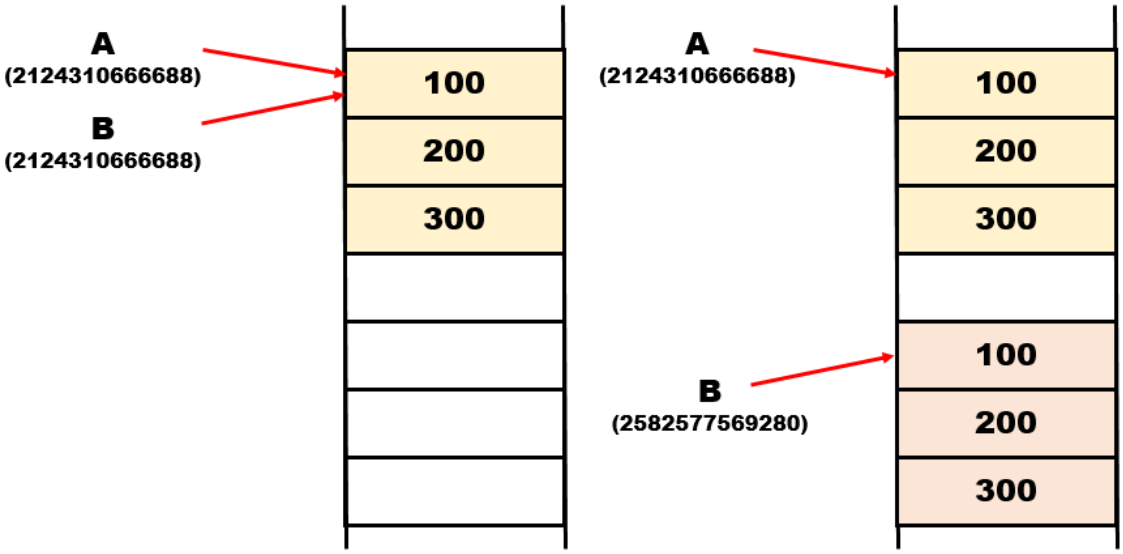
```
sample_list = [11, 22, 33, 55, 66]

# 주어진 리스트의 3번 index에 있는 항목을 제거하고 변수에 할당해주세요

print("original list", sample_list) # original list [11, 22, 33, 55, 66]

elem = sample_list.pop(3)

print('list after: ', sample_list) # list after:  [11, 22, 33, 66]
print('elem: ', elem) # elem:  55

# sample_list의 가장 뒤에 77을 추가해보세요
sample_list.append(77)

# 할당해놓은 변수의 값을 sample_list의 2번 index에 추가해보세요.
print(sample_list) # [11, 22, 33, 66, 77]
sample_list.insert(2, elem)
print(sample_list) # [11, 22, 55, 33, 66, 77]
```

```
my_tuple = (11, 22, 33, 44, 55, 66)

# 주어진 튜플에서 44와 55의 값을 새로운 튜플에 할당해 보세요.
new_tuple = my_tuple[3:5]
print(new_tuple) # (44, 55)
```

```
my_dit = {'apple': '사과', 'pineapple': '파인애플', 'banana': '바나나', 'melon': '멜론'}
# my_list = ['apple', 'pineapple', 'banana', 'melon']
# for key in my_dict:
#     print(key, my_dict.get(key,'없음'))
```

```
my_dict = dict()
my_list = ['apple', 'pineapple', 'pineapple', 'pineapple', 'banana', 'banana']
for key in my_list:
    my_dict[key] = my_dict.setdefault(key, 0) + 1

print(my_dict)
#{'apple': 1, 'pineapple': 3, 'banana': 2}
```

얕은 복사와 깊은 복사



기존 변수 사용 과정에서의 문제점?

하나의 기억에 하나의 주소가 필요 == 100개 저장하려면 주소 100개 필요

연속적인 공간에 데이터가 저장되도록하여 `맨 처음 주소`만 킵

Shallow Copy & Deep Copy

할당(Assignment)

- 대입 연산자(=)

```
original_list = [1, 2, 3]
copy_list = original_list
print(original_list, copy_list) # [1, 2, 3] [1, 2, 3]

print(id(original_list)) # 2124310666688
print(id(copy_list)) # 2124310666688

copy_list[0] = 'hello'
print(original_list, copy_list) # ['hello', 2, 3] ['hello', 2, 3]
```

얕은 복사 (Shallow copy)

Slice 연산자를 활용하여 같은 원소를 가진 리스트지마 연산된 결과를 복사

```
a = [1, 2, 3]
b = a[:]
print(a, b) # [1, 2, 3] [1, 2, 3]
b[0] = 5
print(a, b) # [1, 2, 3] [5, 2, 3]
```

복사하는 리스트의 원소가 주소를 참조하는 경우

```
a = [1, 2, ['a', 'b']]
b = a[:]
print(a, b) # [1, 2, ['a', 'b']] [1, 2, ['a', 'b']]
b[2][0] = 0
print(a, b) # [1, 2, [0, 'b']] [1, 2, [0, 'b']]
```

깊은 복사(Deep copy)

- import copy
- copy.deepcopy()
- 메모리 용량 증가

```
import copy

a = [1, 2, ['a', 'b']]
b = copy.deepcopy(a)
print(a, b) # [1, 2, ['a', 'b']] [1, 2, ['a', 'b']]
b[2][0] = 0
print(a, b) # [1, 2, ['a', 'b']] [1, 2, [0, 'b']]
```

> **TIP**
>
> Q. 얕은 복사와 깊은 복사를 구분해서 사용하는 이유가 무엇인가요?
>
> - 깊은 복사(Deep Copy)는 '실제 값' 전체를 새로운 메모리 공간에 복사하는 것
> - 얕은 복사(Shallow Copy)는 '주소 값'을 복사하는 것

```
#deepcopy
def my_deepcopy(x):
    new = None # local에서 사용하기 위해
    # immutable(불변형)일 때 바로 반환
    if type(x) not in (list, set, dict):
        return x

    # list인 경우
    elif type(x) is list:
        # 새로운 리스트를 하나 생성해서
        new = []
        # 요소를 하나씩 복사(재귀적으로 진행)
        for i in x:
            new.append(my_deepcopy(i))

    # set인 경우
    elif type(x) is set:
        pass
    # dict인 경우
    elif type(x) is dict:
        pass
    return new

a = [1, 2, [3, [4, 5]]]
b = my_deepcopy(a)

a[2][1][1] = 10

print(a)
print(b)
```