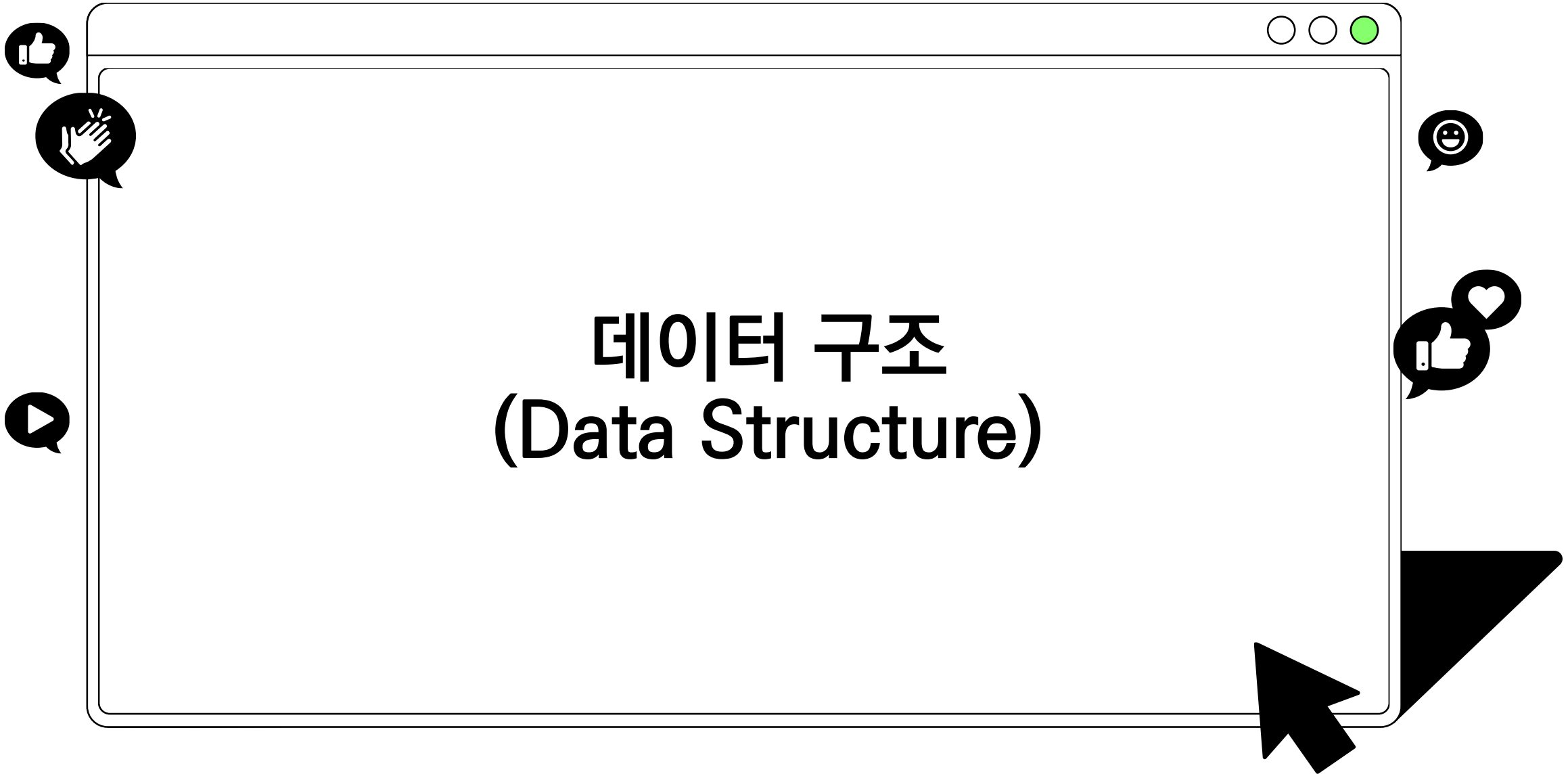
HAPPY HACKING



데이터 구조
(Data Structure)

컴퓨터(Computer)

Caculation + Remember

조작(계산)하고, 저장한다.

HAPPY HACKING

선언적 지식(declarative knowledge)

"사실에 대한 내용"

명령적 지식(imperative knowledge)

"How-to"

HAPPY HACKING

변수와 타입

int, float, complex, bool
str, list, tuple, range
set, dictionary

```
len('happy!')
```

HAPPY HACKING

len(‘happy!’)

```
word = 'happy!'
cnt = 0
for char in word:
    cnt += 1
```

함수(function)

decomposition, abstraction

input().split()

문자열.split()

```
[1, 2, 3].append(4)
```

리스트.append(4)

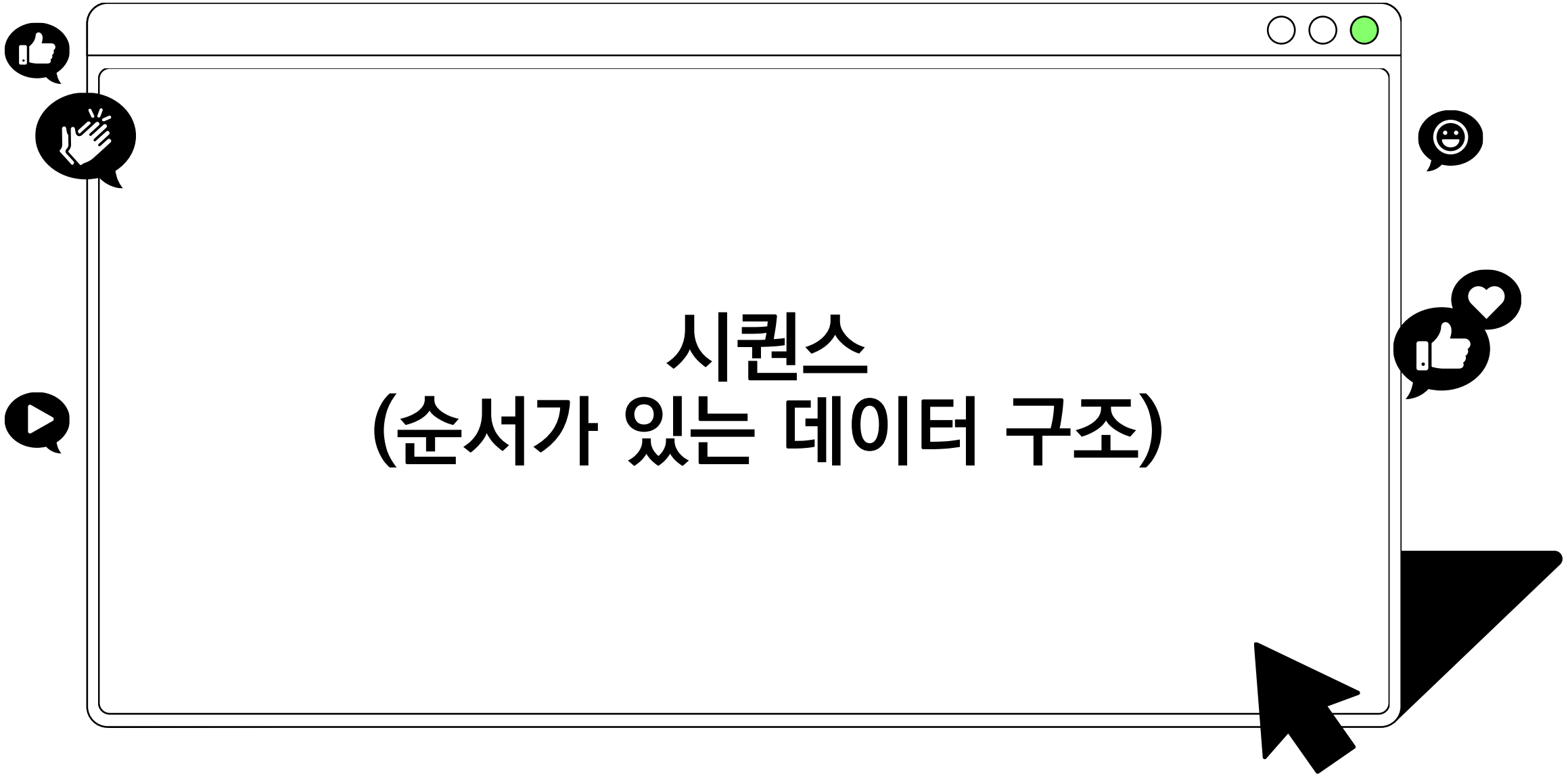
타입.메서드()

메서드(methods)

목차

HAPPY HACKING

HAPPY HACKING



시퀀스
(순서가 있는 데이터 구조)

HAPPY HACKING

문자열(String Type)

HAPPY HACKING

문자열

- 문자들의 나열(sequence of characters)
  - 모든 문자는 str 타입
- 문자열은 작은 따옴표(')나 큰 따옴표(")를 활용하여 표기
  - 문자열을 묶을 때 동일한 문장부호를 활용
  - PEP8에서는 소스코드 내에서 하나의 문장부호를 선택하여 유지하도록 함

```
print('hello')
```

```
hello
```

```
print('철수 "안녕"')
```

```
철수 "안녕"
```

```
print(type('hello'))
```

```
<class 'str'>
```

```
print("철수 '안녕'")
```

```
철수 '안녕'
```

HAPPY HACKING

문자열 탐색/검증

문법	설명
`s.find(x)`	x의 첫 번째 위치를 반환. 없으면, -1을 반환
`s.index(x)`	x의 첫 번째 위치를 반환. 없으면, 오류 발생
`s.isalpha()`	알파벳 문자 여부 *단순 알파벳이 아닌 유니코드 상 Letter (한국어도 포함)
`s.isupper()`	대문자 여부
`s.islower()`	소문자 여부
`s.istitle()`	타이틀 형식 여부

HAPPY HACKING

문자열 탐색

- .find(x)
  - x의 첫 번째 위치를 반환. 없으면, -1을 반환함.

```
print('apple'.find('p'))
# 1
print('apple'.find('k'))
# -1
```

HAPPY HACKING

문자열 탐색

- .index(x)
  - x의 첫 번째 위치를 반환. 없으면, 오류 발생

```
print('apple'.index('p’))
# 1
'apple'.index('k')
# ----------------------------------------
# ValueError Traceback (most recent call last)
# ----> 1 'apple'.index('k')
# ValueError: substring not found
```

문자열 관련 검증 메소드

```
print('abc'.isalpha())
# True
print('Ab'.isupper())
# False
print('ab'.islower())
# True
print('Title Title!'.istitle())
# True
```

HAPPY HACKING

문자열 관련 검증 메서드

- isdecimal() ⊆ .isdigit() ⊆ .isnumeric()
  - 아래의 예시를 암기할 필요는 없음

isdecimal()	isdigit()	isnumeric()	example
True	True	True	"O38", "๐๓๘", "０３８"
False	True	True	"⁰³⁸", "0.₃.₈.", "⓪③⑧"
False	False	True	"⅛⅘", "ⅠⅢⅧ", "⑩⑬㊿", "壹貳參"
False	False	False	"abc", "38.0", "-38"

HAPPY HACKING

문자열 변경

문법	설명
s.replace(*old*, *new*[,*count*])	바꿀 대상 글자를 새로운 글자로 바꿔서 반환
s.strip([*chars*])	공백이나 특정 문자를 제거
s.split(*sep=None*, *maxsplit=-1*)	공백이나 특정 문자를 기준으로 분리
'*separator*'.join([*iterable*])	구분자로 iterable을 합침
s.capitalize()	가장 첫 번째 글자를 대문자로 변경
s.title()	'나 공백 이후를 대문자로 변경
s.upper()	모두 대문자로 변경
s.lower()	모두 소문자로 변경
s.swapcase()	대↔ 소문자 서로 변경

HAPPY HACKING

문자열 변경

- .replace(old, new[,count])
  - 바꿀 대상 글자를 새로운 글자로 바꿔서 반환
  - count를 지정하면, 해당 개수만큼만 시행

```
print('coone'.replace('o', 'a’))
# caane
print('wooooowoo'.replace('o', '!', 2))
# w!!ooowoo
```

HAPPY HACKING

문자열 변경

- .strip([chars])
  - 특정한 문자들을 지정하면,
  - 양쪽을 제거하거나(strip), 왼쪽을 제거하거나(lstrip), 오른쪽을 제거(rstrip)
  - 문자열을 지정하지 않으면 공백을 제거함

```
print('    와우!\n'.strip())
# '와우!'
print('    와우!\n'.lstrip())
# '와우!\n'
print('    와우!\n'.rstrip())
# '    와우!'
print('안녕하세요????'.rstrip('?'))
# '안녕하세요'
```

문자열 변경

- .split(sep=None, maxsplit=-1)
  - 문자열을 특정한 단위로 나눠 리스트로 반환
    - sep이 None이거나 지정되지 않으면 연속된 공백문자를 단일한 공백문자로 간주하고, 선행/후행 공백은 빈 문자열에 포함시키지 않음.
    - maxsplit이 -1인 경우에는 제한이 없음.

```
print('a,b,c'.split('_'))
# ['a,b,c']
print('a b c'.split())
# ['a', 'b', 'c']
```

HAPPY HACKING

문자열 변경

- 'separator'.join([iterable])
  - 반복가능한(iterable) 컨테이너 요소들을 separator(구분자)로 합쳐 문자열 반환
    - iterable에 문자열이 아닌 값이 있으면 TypeError 발생

```
print(''.join(['3', '5’]))
# 35
```

기타 변경

- 문자열 변경 예시

```
msg = 'hI! Everyone.'
print(msg)
print(msg.capitalize())
print(msg.title())
print(msg.upper())
print(msg.lower())
print(msg.swapcase())

# hI! Everyone.
# Hi! everyone.
# Hi! Everyone.
# HI! EVERYONE.
# hi! everyone.
# Hi! eVERYONE.
```

HAPPY HACKING

리스트
(List)

HAPPY HACKING

리스트(List) 정의

- 변경 가능한 값들의 나열된 자료형
- 순서를 가지며, 서로 다른 타입의 요소를 가질 수 있음
- 변경 가능하며(mutable), 반복 가능함(iterable)
- 항상 대괄호 형태로 정의하며, 요소는 콤마로 구분

[0, 1, 2, 3, 4, 5]

리스트(List)

HAPPY HACKING

문법	설명
`L.append(x)`	리스트 마지막에 항목 x를 추가
`L.insert(i, x)`	리스트 인덱스 i에 항목 x를 삽입
`L.remove(x)`	리스트 가장 왼쪽에 있는 항목(첫 번째) x를 제거 항목이 존재하지 않을 경우, ValueError
`L.pop()`	리스트 가장 오른쪽에 있는 항목(마지막)을 반환 후 제거
`L.pop(i)`	리스트의 인덱스 i에 있는 항목을 반환 후 제거
`L.extend(m)`	순회형 m의 모든 항목들의 리스트 끝에 추가 (+=과 같은 기능)
`L.index(x, start, end)`	리스트에 있는 항목 중 가장 왼쪽에 있는 항목 x의 인덱스를 반환
`L.reverse()`	리스트를 거꾸로 정렬
`L.sort()`	리스트를 정렬 (매개변수 이용가능)
`L.count(x)`	리스트에서 항목 x가 몇 개 존재하는지 갯수를 반환

HAPPY HACKING

값 추가 및 삭제

- .append(x)
  - 리스트에 값을 추가함

```
cafe = ['starbucks', 'tomntoms', 'hollys’]
print(cafe)
# ['starbucks', 'tomntoms', 'hollys']
cafe.append('banapresso’)
print(cafe)
# ['starbucks', 'tomntoms', 'hollys',
'banapresso']
```

HAPPY HACKING

값 추가 및 삭제

- .extend(iterable)
  - 리스트에 iterable의 항목을 추가함

```
cafe = ['starbucks', 'tomntoms', 'hollys’]
print(cafe)
# ['starbucks', 'tomntoms', 'hollys']
cafe.extend(['cafe', 'test’])
print(cafe)
# ['starbucks', 'tomntoms', 'hollys', 'cafe', 'test]
```

HAPPY HACKING

값 추가 및 삭제

- .insert(i, x)
  - 정해진 위치 i에 값을 추가함

```
cafe = ['starbucks', 'tomntoms’]
print(cafe)
# ['starbucks', 'tomntoms']
cafe.insert(0, 'start’)
print(cafe)
# ['start', 'starbucks', 'tomntoms']
```

리스트 길이보다 큰 경우 맨 뒤

```
cafe = ['starbucks', 'tomntoms’]
print(cafe)
# ['starbucks', 'tomntoms']
cafe.insert(10000, 'end’)
print(cafe)
# ['starbucks', 'tomntoms', 'end']
```

HAPPY HACKING

값 추가 및 삭제

- .remove(x)
  - 리스트에서 값이 x인 것 삭제

```
numbers = [1, 2, 3, 'hi’]
print(numbers)
# [1, 2, 3, 'hi']
numbers.remove('hi’)
print(numbers)
# [1, 2, 3]
```

```
numbers.remove('hi')
# ----------------
# ValueError   Traceback (most recent call last)
# ----> 1 numbers.remove('hi')
# ValueError: list.remove(x): x not in list
```

없는 경우 ValueError

HAPPY HACKING

값 추가 및 삭제

- .pop(i)
  - 정해진 위치 i에 있는 값을 삭제하고, 그 항목을 반환함
  - i가 지정되지 않으면, 마지막 항목을 삭제하고 반환함

```
numbers = ['hi', 1, 2, 3]
# ['hi', 1, 2, 3]
pop_number = numbers.pop()
print(pop_number)
# 3
print(numbers)
# ['hi', 1, 2]
```

```
numbers = ['hi', 1, 2, 3]
# ['hi', 1, 2, 3]
pop_number = numbers.pop(0)
print(pop_number)
# 'hi'
print(numbers)
# [1, 2, 3]
```

값 추가 및 삭제

- .clear()
  - 모든 항목을 삭제함

```
numbers = [1, 2, 3]
print(numbers)
# [1, 2, 3]
print(numbers.clear())
# []
```

HAPPY HACKING

탐색 및 정렬

- .index(x)
  - x값을 찾아 해당 index 값을 반환

```
numbers = [1, 2, 3, 4]
print(numbers)
# [1, 2, 3, 4]
print(numbers.index(3))
# 2
print(numbers.index(100))
# ---------------------
# ValueError Traceback (most recent call last)
#       2 print(numbers)
#       3 print(numbers.index(3))
# ----> 4 print(numbers.index(100))

# ValueError: 100 is not in list
```

없는 경우 ValueError

탐색 및 정렬

- .count(x)
  - 원하는 값의 개수를 반환함

```
numbers = [1, 2, 3, 1, 1]
print(numbers.count(1))
# 3
print(numbers.count(100))
# 0
```

HAPPY HACKING

탐색 및 정렬

- .sort()
  - 원본 리스트를 정렬함. None 반환
  - sorted 함수와 비교할 것

```
numbers = [3, 2, 5, 1]
result = numbers.sort()
print(numbers, result)
# [1, 2, 3, 5] None
```

원본 변경

```
numbers = [3, 2, 5, 1]
result = sorted(numbers)
print(numbers, result)
# [3, 2, 5, 1] [1, 2, 3, 5]
```

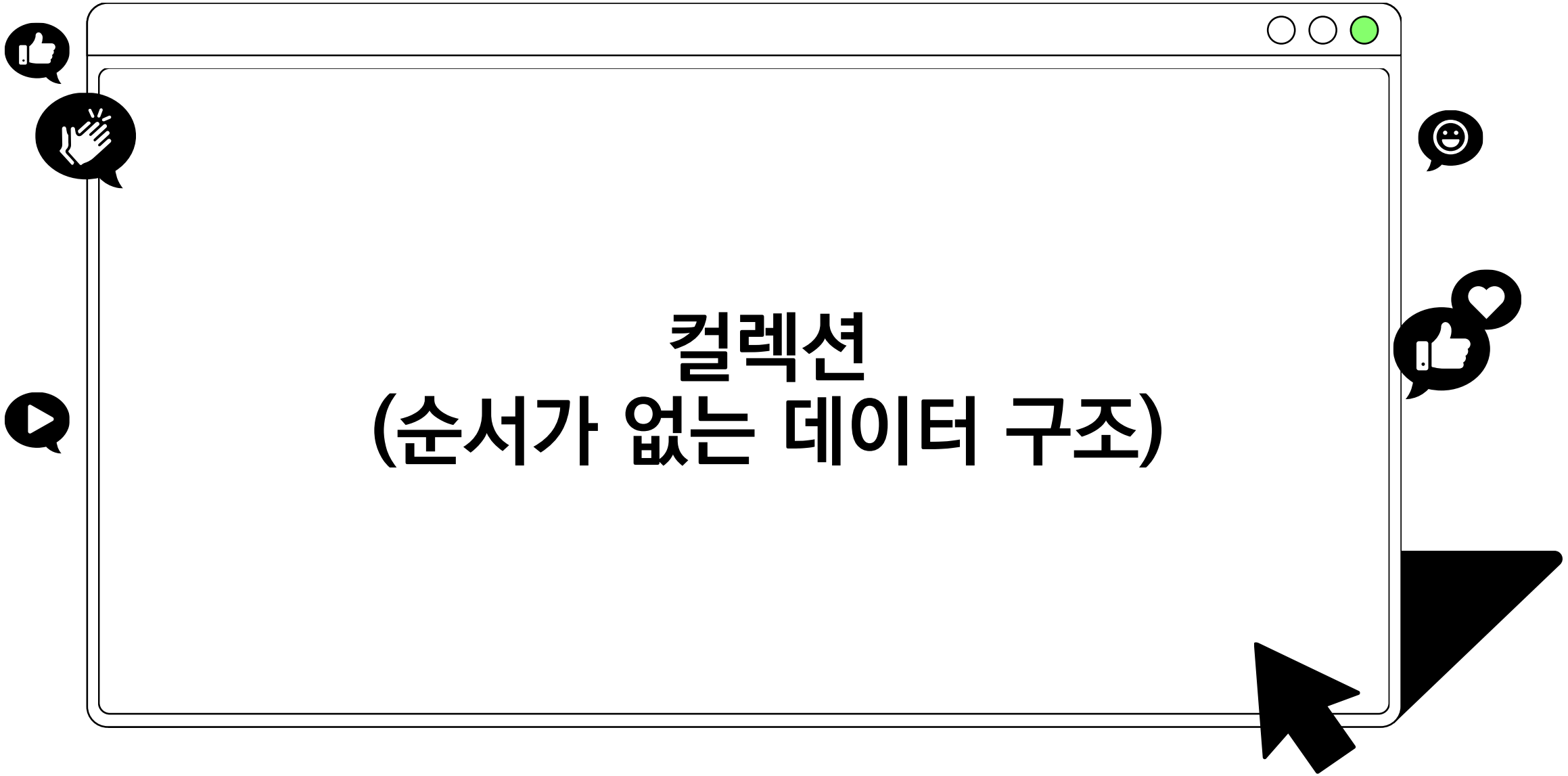
정렬된 리스트를 반환. 원본 변경 없음

탐색 및 정렬

- .reverse()
  - 순서를 반대로 뒤집음(정렬하는 것이 아님). None 반환.

```
numbers = [3, 2, 5, 1]
result = numbers.reverse()
print(numbers, result)
# [1, 5, 2, 3] None
```

HAPPY HACKING



컬렉션
(순서가 없는 데이터 구조)

세트
(Set)

HAPPY HACKING

세트(set)

- 유일한 값들의 모음(collection)
- 순서가 없고 중복된 값이 없음.
  - 수학에서의 집합과 동일한 구조를 가지며, 집합 연산도 가능
- 변경 가능하며(mutable), 반복 가능함(iterable)
  - 단, 세트는 순서가 없어 반복의 결과가 정의한 순서와 다를 수 있음

HAPPY HACKING

세트 메서드

문법	설명
`s.copy()`	세트의 얕은 복사본을 반환
`s.add(x)`	항목 x가 세트 s에 없다면 추가
`s.pop()`	세트s에서 랜덤하게 항목을 반환하고, 해당 항목을 제거 세트가 비어 있을 경우, KeyError
`s.remove(s)`	항목 x를 세트 s에서 삭제 항목이 존재하지 않을 경우, KeyError
`s.discard(x)`	항목 x가 세트 s에 있는 경우, 항목 x를 세트s에서 삭제
`s.update(t)`	세트 t에 있는 모든 항목 중 세트 s에 없는 항목을 추가
`s.clear()`	모든 항목을 제거
`s.isdisjoint(t)`	세트 s가 세트 t의 서로 같은 항목을 하나라도 갖고 있지 않은 경우, True반환
`s.issubset(t)`	세트 s가 세트 t의 하위 세트인 경우, True반환
`s.issuperset(t)`	세트 s가 세트 t의 상위 세트인 경우, True반환

딕셔너리
(Dictionary)

HAPPY HACKING

딕셔너리(Dictionary)

- 키-값(key-value) 쌍으로 이뤄진 모음(collection)
  - 키(key)
    - 불변 자료형만 가능 (리스트, 딕셔너리 등은 불가능함)
  - 값(values)
    - 어떠한 형태든 관계 없음
- 키와 값은 :로 구분 됩니다. 개별 요소는 ,로 구분됩니다.
- 변경 가능하며(mutable), 반복 가능함(iterable)
  - 딕셔너리는 반복하면 키가 반환됩니다.

```
students = {'홍길동': 30, '김철수': 25}
students['홍길동']
# 30
```

딕셔너리(Dictionary)

HAPPY HACKING

문법	설명
`d.clear()`	모든 항목을 제거
`d.keys()`	딕셔너리 d의 모든 키를 담은 뷰를 반환
`d.values()`	딕셔너리 d의 모든 값를 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
`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
`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

HAPPY HACKING

조회

- .get(key[,default])
  - key를 통해 value를 가져옴
  - KeyError가 발생하지 않으며, default 값을 설정할 수 있음(기본: None)

```
my_dict = {'apple': '사과', 'banana': '바나나'}
my_dict['pineapple']
# ----------------------------
# KeyError  Traceback (most recent call last)
#       1 my_dict = {'apple': '사과', 'banana':
'바나나'}
# ----> 2 my_dict['pineapple’]
# KeyError: 'pineapple'
```

```
my_dict = {'apple': '사과', 'banana': '바나나'}
print(my_dict.get('pineapple'))
# None
print(my_dict.get('apple'))
# 사과
print(my_dict.get('pineapple', 0))
# 0
```

추가 및 삭제

- .pop(key[,default])
  - key가 딕셔너리에 있으면 제거하고 해당 값을 반환
  - 그렇지 않으면 default를 반환
  - default값이 없으면 KeyError

```
my_dict = {'apple': '사과', 'banana': '바나나'}
data = my_dict.pop('apple')
print(data, my_dict)
# 사과 {'banana': '바나나'}
```

추가 및 삭제

- .pop(key[,default])
  - key가 딕셔너리에 있으면 제거하고 해당 값을 반환
  - 그렇지 않으면 default를 반환
  - default값이 없으면 KeyError

```
my_dict = {'apple': '사과', 'banana': '바나나'}
data = my_dict.pop('pineapple')
print(data, my_dict)
# ----------------
# KeyError  Traceback (most recent call last)
#       1 my_dict = {'apple': '사과', 'banana': '바나나'}
# ----> 2 data = my_dict.pop('pineapple')
#       3 print(data, my_dict)

# KeyError: 'pineapple'
```

추가 및 삭제

- .update([other])
  - 값을 제공하는 key, value로 덮어씁니다.

```
my_dict = {'apple': '사', 'banana': '바나나'}
my_dict.update(apple='사과')
print(my_dict)
# {‘apple’: ‘사과’, 'banana': '바나나'}
```