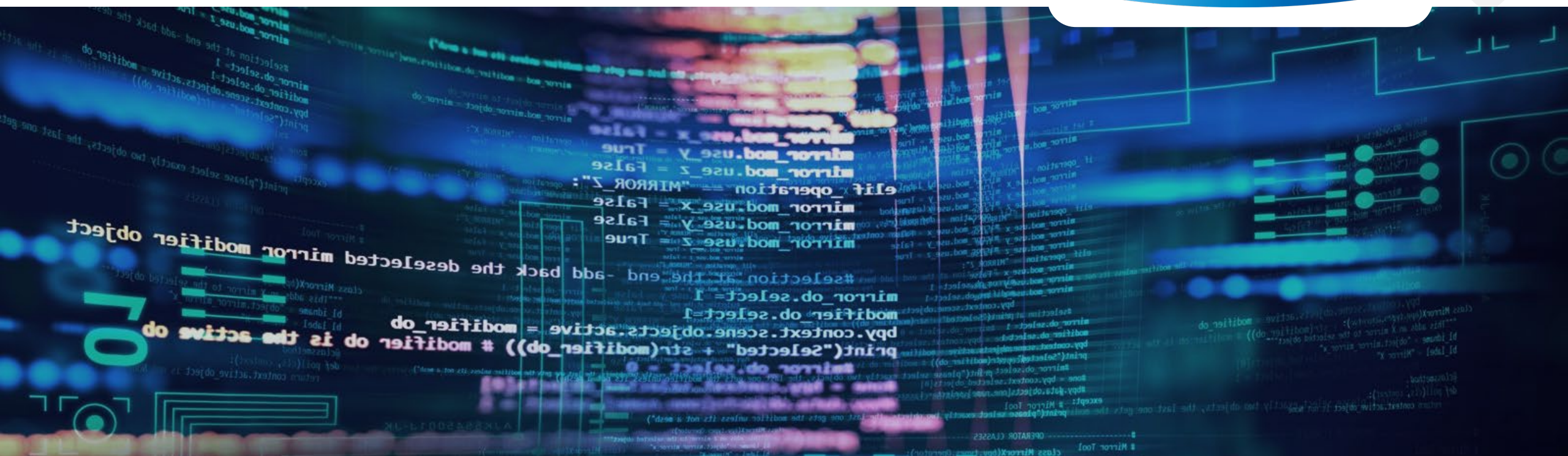


Lecture 8

사전과 집합

KAIST





- 파이썬에는 다양한 빌트인 자료형들이 있음을 배웠다
- 파이썬의 수많은 자료형 중에서도 자주 사용되는 자료형들을 선정하면 다음과 같다:
 - (부울) bool
 - (숫자) int, float, complex
 - (문자열) str
 - (시퀀스) str, list, tuple
 - (매핑) **dict**
 - (세트) **set**
- 직전 차시에서는 리스트와 튜플 자료형에 대해 살펴보았다
- 이번 차시에서는 그중에서도 **사전(dict; 딕셔너리)**와 **집합(set; 세트)** 자료형에 대해 배워보자



- 사전 자료형은 **dict**로 표기되며 **dictionary**의 준말이다
- 백과사전을 볼 때 **키워드와 내용**이 담겨있는 것처럼 사전 자료형도 **key:value** 형식을 띈다:

```
{key_1:val_1, key_2:val_2, ..., key_n:val_n}
```

- **key와 value의 한 쌍**이 곧 **사전을 이루는 원소단위**이다
- 리스트는 대괄호, 튜플은 소괄호였다면 사전과 집합은 **중괄호{}**로 원소들을 감싼다
- 우리가 백과사전에서 키워드로 검색해서 내용을 찾듯이 파이썬의 사전 자료형도 key값으로 검색하여 value값을 조회하는 것이 일반적
- 예제 코드를 통해 사전 자료형에 대해 알아보도록 하자



➤ 아래와 같이 사전 자료형 속 key와 value는 자료형으로 다양하게 들어갈 수 있다

[예제 1] 1) `my_first_dict = {"Name": "Nubzuki", "Birth Year": 2014, 4.9:{"Hi":{"Hello"}}}, ("Bonjour":["안녕"])`

➤ 하지만 **key**의 경우에는 자료형의 제약을 받게 되는데, 바로 **hashable(해시가능)**해야 한다

- hashable은 값을 해시함수에 넣어 hash(해시) 값으로 변환할 수 있다는 것을 의미한다

➤ hash는 **어떠한 특정 값에 대해서 유일한 값을 가져야 한다**

- 따라서 해시함수의 입력값이 변하면 출력값도 변한다
 - 백과사전을 펼쳤을 때 똑같은 키워드로 검색을 했는데 계속해서 다른 내용이 나오면 안되는 것과 동치
- **사전 자료형의 key 값은 hashable** 해야하고, 추후 수정/변경을 하지 못하게 **immutable**한 자료형으로 존재해야함
(지키지 않으면 TypeError 발생)

2) `yes_tuple_key = {(0):1} # 튜플 -> immutable`
3) `no_list_key = {[0]:1} # 리스트 -> mutable`

TypeError: unhashable type: 'list'



➤ 다시 돌아와서, **사전 자료형의 사용법**에 대해 더 알아보자

[예제 2] 1) Age_dict = {"Nubzuki": 9} # 딕셔너리 객체 생성
2) Age_dict["Yonghyun"] = 25 # 딕셔너리에 key-value 쌍 추가
3) print(Age_dict)
4) del Age_dict["Yonghyun"] # 특정 key 값에 해당되는 원소 제거
5) print(Age_dict)

{'Nubzuki': 9, 'Yonghyun': 25}
{'Nubzuki': 9}

➤ 사전 자료형의 **key값**들은 **hashable** 해야하기에 **중복된 값을 가질 수 없다**

만약에 동일한 key값을 가지지만 다른 value값을 가지도록 명령한다면 해당 key값에 대응되는 value가 새로운 값으로 갱신됨

6) Age_dict["Nubzuki"] = 10 # 이전 value 값인 9가 10으로 수정
7) print(Age_dict)

{'Nubzuki': 10}

➤ 사전 객체 생성 및 초기화시에도 동일한 로직이 적용된다

8) Age_dict = {"Nubzuki": 8, "Nubzuki": 9, "Nubzuki": 10}
9) print(Age_dict)

{'Nubzuki': 10}




➤ 사전 자료형의 대표적인 메소드 중 네 가지를 소개한다: **keys, values, items, get**

[예제 3]

```
1) Age_dict = {"Nubzuki": 9, "Yonghyun": 25, "KAIST": 52}
2) print(Age_dict.keys()) # 딕셔너리 객체의 key 값들을 모아 의사 리스트 형태로 만들 (리스트는 아님)
3) print(Age_dict.values()) # value 값들을 모아 의사 리스트 형태로 만들
4) print(Age_dict.items()) # 원소(key-value 쌍)들을 모아 의사 리스트 형태로 만들
5) print(Age_dict.get("Yonghyun", "No key"), end=", ") # 특정 값이 딕셔너리의 key값으로 존재하면 대응되는 value를 출력, 없으면
    두번째 파라미터의 값을 내뱉음 (디폴트값은 None)
6) print(Age_dict.get("Santa Claus", "No key"), end=", ")
```

➤ 파이썬의 in 연산자로 key가 존재하는 지를 확인할 수도 있다

```
7) print("Yonghyun" in Age_dict, end=", ") # 특정 값이 딕셔너리의 key값으로 존재하는가
8) print("Santa Claus" in Age_dict)
```



```
dict_keys(['Nubzuki', 'Yonghyun', 'KAIST'])
dict_values([9, 25, 52])
dict_items([('Nubzuki', 9), ('Yonghyun', 25), ('KAIST', 52)])
25, No key, True, False
```




➤ 사전 자료형의 병합은 다음과 같이 수행할 수 있다

[예제 4] 1) Nubzuki_dict= {"Name": "Nubzuki", "Age": 25}
2) Temp_dict = {"Trait": ['cute', 'adorable', 'smart'], "Phone Num": None}

방법 1 (파이썬 3.5 버전 이상부터 지원)

3) print(**Nubzuki_dict, **Temp_dict)

4) print(**Temp_dict, **Nubzuki_dict) # 파라미터 순서에 따라 달라지는 출력 결과

방법 2 - update 메소드 사용

5) print(Nubzuki_dict)

6) Nubzuki_dict.update(Temp_dict)

7) print(Nubzuki_dict)

```
{'Name': 'Nubzuki', 'Age': 25, 'Trait': ['cute', 'adorable', 'smart'], 'Phone Num': None}
{'Trait': ['cute', 'adorable', 'smart'], 'Phone Num': None, 'Name': 'Nubzuki', 'Age': 25}
{'Name': 'Nubzuki', 'Age': 25}
{'Name': 'Nubzuki', 'Age': 25, 'Trait': ['cute', 'adorable', 'smart'], 'Phone Num': None}
```



- 집합 자료형은 **set**으로 표기되며 수학에서 배운 집합과 유사하다
- 집합의 특징인 동일한 원소를 중복하여 가지고 있을 수 없고 해당 원소들은 특정한 순서를 띄지 않는다는 것을 그대로 반영 (동일한 값을 소지할 수 없는 주머니를 연상)
- 사전 자료형과 동일하게 **중괄호{}**로 원소들을 묶지만, 원소는 딕셔너리와 다르게 단일 값들을 가진다

[예제 5] 1) Nubzuki_dict= {"Name": "Nubzuki", "Age": 25}
2) my_first_set = {"element1", "element2", "element3"}
3) print(Nubzuki_dict)
4) print(my_first_set)
5) print(my_first_set[0]) # Not subscriptable (인덱스 X)

{'Name': 'Nubzuki', 'Age': 25}
{'element2', 'element1', 'element3'}

TypeError: 'set' object is not subscriptable

- 집합 내 원소는 사전 자료형의 key값과 동일한 자료형의 제약을 받게 되는데, 바로 **hashable(해시가능)**해야한다
- 잇따라 오는 예제 코드를 통해 집합 자료형에 대해 더 알아보자



➤ 집합 자료형의 대표적인 메소드 중 세 가지를 소개한다: **add, update, remove**

[예제 6]

```
1) hello_set = {"Hello"} # 집합 객체 초기화
2) hello2_set = set("Hello") # 문자열의 각 문자를 집합의 원소로 간주
3) print(hello_set)
4) print(hello2_set) # 집합 내 원소의 unordered 특성
5) hello2_set.add("d") # add: 집합에 원소 한개 추가
6) print(hello2_set)
7) hello2_set.update("e", "f") # update: 원소 여러개 추가
8) print(hello2_set)
9) hello2_set.remove("l") # remove: 원소 한개 제거
10) print(hello2_set)
```

```
{'Hello'}
{'o', 'l', 'H', 'e'}
{'o', 'l', 'H', 'e', 'd'}
{'o', 'l', 'H', 'e', 'f', 'd'}
{'o', 'H', 'e', 'f', 'd'}
```



수학에서의 집합 개념과 유사하게 파이썬의 집합 자료형도 다음과 같은 네가지의 집합 연산을 지원한다:

- **교집합** (intersection): **&** 연산자 혹은 **intersection** 메소드 사용
- **합집합** (union): **|** 연산자 혹은 **union** 메소드 사용
- **차집합** (difference): **-** 연산자 혹은 **difference** 메소드 사용
- **대칭차집합** (symmetric difference): **^** 혹은 **symmetric_difference** 메소드 사용

[예제 7]

```
1) set1 = set([1, 2, 3, 4, 5.0]) # 리스트 객체의 형변환
2) set2 = set([3, 4, 5.0, 5, 6, "7"])
3) print(set2) # 집합에서는 수학적으로 값이 같으면 동일 원소로 간주
4) print(set1 & set2) # 교집합 - 연산자
5) print(set1.intersection(set2)) # 교집합 - 메소드
6) print(set1 | set2) # 합집합 - 연산자
7) print(set1.union(set2)) # 합집합 - 메소드
```

```
{3, 4, 5.0, 6, '7'}
{3, 4, 5.0}
{3, 4, 5.0}
{1, 2, 3, 4, 5.0, 6, '7'}
{1, 2, 3, 4, 5.0, 6, '7'}
```




수학에서의 집합 개념과 유사하게 파이썬의 집합 자료형도 다음과 같은 네가지의 집합 연산을 지원한다:

- **교집합** (intersection): **&** 연산자 혹은 **intersection** 메소드 사용
- **합집합** (union): **|** 연산자 혹은 **union** 메소드 사용
- **차집합** (difference): **-** 연산자 혹은 **difference** 메소드 사용
- **대칭차집합** (symmetric difference): **^** 혹은 **symmetric_difference** 메소드 사용

[예제 8]

```
1) set1 = set([1, 2, 3, 4, 5.0]) # 리스트 객체의 형변환
2) set2 = set([3, 4, 5.0, 5, 6, "7"])
3) print(set1 - set2) # 차집합 - 연산자 (1)
4) print(set2 - set1) # 차집합 - 연산자 (2)
5) print(set1.difference(set2)) # 차집합 - 메소드 (1)
6) print(set2.difference(set1)) # 차집합 - 메소드 (2)
7) print(set1 ^ set2) # 대칭차집합 - 연산자
8) print(set1.symmetric_difference(set2)) # 대칭차집합 - 메소드
9) print((set1 | set2) - (set1 & set2)) # 위와 동치
```



```
{1, 2}
{6, '7'}
{1, 2}
{6, '7'}
{1, 2, 6, '7'}
{1, 2, 6, '7'}
{1, 2, 6, '7'}
```



- **집합 자료형**은 리스트와 튜플로의 **형변환이 가능**하다 (역으로 가능)
- 다만, **사전 자료형**은 리스트와 튜플로의 형변환 중 value 값을 손실하고, 역으로 **형변환이 불가**하다

[예제 9]

```
1) my_set = {1, (2,), 3.0, "a"}
2) my_dict = {1: (2), 3.0: "a"}
3) my_tuple = tuple(my_set) # 집합 -> 튜플
4) my_list = list(my_set) # 집합 -> 리스트
5) print(my_tuple)
6) print(my_list)
7) my_tuple = tuple(my_dict) # 사전 -> 튜플
8) my_list = list(my_dict) # 사전 -> 리스트
9) print(my_tuple)
10) print(my_list)
11) my_set_2 = set(my_tuple) # 튜플 -> 집합
12) print(my_set_2)
13) my_dict_2 = dict(my_tuple) # 튜플 -> 사전 (불가능)
```

(1, 'a', 3.0, (2,))
[1, 'a', 3.0, (2,)]
(1, 3.0)
[1, 3.0]
{1, 3.0}

**TypeError: cannot convert dictionary update
sequence element #0 to a sequence**