LLM 활용 인공지능 서비스 개발자 양성과정

도봉 SeSAC 캠퍼스 X Saltlux

강사 최동혁

소개

- 사전 자료형은 dict로 표기되며 dictionary의 준말이다.
- 백과사전을 볼 때 키워드와 내용이 담겨있는 것처럼 사전 자료형도 key:value 형식을 띈다:

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

key, value

- key와 value의 한 쌍이 곧 사전을 이루는 단위이다.
- 리스트는 대괄호, 튜플은 소괄호였다면 사전과 집합은 중괄호{}로 원소들을 감싼다.
- 우리가 백과사전에서 키워드로 검색해서 내용을 찾듯이 파이썬의 사전 자료형도 key값으로 검색하여 value값을 조회하는 것이 일반적이다.

```
my_first_dict = {"Name": "Hoi", "Birth Year": 2023, 4.9:{"Hi":{"Hello"}}, ("Bonjour"):["안녕"]}
```

- 위와 같이 사전 자료형 속 key와 value는 자료형으로 다양하게 들어갈 수 있다.

Hashable

- 하지만 key의 경우에는 자료형의 제약을 받게 되는데, 바로 hashable(해시 가능)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 동치이다.

Hashable - 예제

따라서 사전 자료형의 key 값은 hashable 해야하고 추후 수정/변경을 하지 못하게 immutable한 자료형으로 존재해야 한다 (이를 지키지 않으면 TypeError 발생).

```
yes_tuple_key = {(0):1} # 튜플 -> immutable
no_list_key = {[0]:1} # 리스트 -> mutable
age_dict = {"Hoi": 9} # 딕셔너리 객체 생성
age_dict['Kui'] = 4 # 딕셔너리에 key-value 쌍 추가
print(age_dict)
del age_dict['Hoi'] # 특정 key 값에 해당되는 원소 제거
print(age_dict)
사전 자료형의 key값들은 hashable 해야하기에 중복된 값을 가질 수 없다.
```

잠깐! Hash 함수란?

- 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수이다.
- 매핑 전 원래 데이터의 값을 키(key), 매핑 후의 데이터의 값을 해시값(hash value), 매핑하는 과정 자체를 해싱(hashing)이라고 한다.
- 해시함수는 해쉬값의 개수보다 대개 많은 키값을 해쉬값으로 변환(many-to-one) 하기 때문에 해시함수가 서로 다른 두 개의 키에 대해 동일한 해시값을 내는 해시충돌(collison)이 발생하게 된다.
- 보다 자세한 내용은 링크1, 링크2 참조

value 갱신

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

```
age_dict['Hoi'] = 10 # 이전 value 값인 9가 10으로 수정
print(age_dict)
```

- 사전 객체 생성 및 초기화시에도 동일한 로직이 적용된다.

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

메소드

사전 자료형의 대표적인 메소드 중 네 가지를 소개한다.

`keys`, `values`, `items`, `get`

```
age_dict = {"Hoi": 9, "Kui": 25, "Daro": 52}
print(age_dict.keys()) # 딕셔너리 객체의 key 값들을 모아 의사 리스트 형태로 만듦 (리스트는 아님)
print(age_dict.values()) # value 값들을 모아 의사 리스트 형태로 만듦
print(age_dict.items()) # 원소(key-value 쌍)들을 모아 의사 리스트 형태로 만듦
print(age_dict.get("Kui", "No key"), end=", ") # 특정 값이 딕셔너리의 key값으로 존재하면 대응
```

메소드

value를 출력, 없으면 두번째 파라미터의 값을 내뱉음 (디폴트값은 None)

```
print(age_dict.get("Santa Claus", "No key"), end=", ")
```

in 연산자로 key가 존재하는 지를 확인할 수도 있다.

```
print("Kui" in age_dict, end=", ") # 특정 값이 딕셔너리의 key값으로 존재하는가
print("Santa Claus" in age_dict)
```

사전의 병합

사전 자료형의 병합은 다음과 같이 수행할 수 있다

```
kui_dict= {"name": "Kui", "Age": 25}
temp_dict = {"trait": ['cute', 'adorable', 'smart'], "phone": None}
```

방법 1 (파이썬 3.5 버전 이상부터 지원)

```
print({**kui_dict, **temp_dict})
print({**temp_dict, **kui_dict}) # 파라미터 순서에 따라 달라지는 출력 결과
```

사전의 병합

방법 2 - update 메소드 사용

```
print(kui_dict)
kui_dict.update(temp_dict)
print(kui_dict)
```

소개

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

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

```
korea_university_mascort= {"name": "Hwanho", "age": 25}
my_first_set = {"element1", "element2", "element3"}
print(korea_university_mascort)
print(my_first_set)
print(my_first_set[0])  # Not subscriptable (인덱스 X)
```

메소드

- 집합 자료형의 대표적인 메소드 중 세 가지를 소개한다: `add`, `update`, `remove`

```
hello_set = {"Hello"} # 집합 객체 초기화
hello2_set = set("Hello") # 문자열의 각 문자를 집합의 원소로 간주
print(hello_set)
print(hello2_set) # 집합 내 원소의 unordered 특성
hello2_set.add("d") # add: 집합에 원소 한개 추가
print(hello2_set)
```

메소드

- 집합 자료형의 대표적인 메소드 중 세 가지를 소개한다: `add`, `update`, `remove`

```
hello_set = {"Hello"} # 집합 객체 초기화
hello2_set.update("e", "f") # update: 원소 여러개 추가
print(hello2_set)
hello2_set.remove("l") # remove: 원소 한개 제거
print(hello2_set)
```

연산

- 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

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

연산

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

기타

- 집합 자료형은 리스트와 튜플로의 형변환이 가능하다 (역으로 가능).
- 다만, 사전 자료형은 리스트와 튜플로의 형변환 중 value 값을 손실하고, 역으로 형변환이 불가하다.

```
my_set = {1, (2,), 3.0, "a"}
my_dict = {1: (2), 3.0: "a"}
my_tuple = tuple(my_set) # 집합 -> 튜플
my_list = list(my_set) # 집합 -> 리스트
my_tuple = tuple(my_dict) # 사전 -> 튜플
my_list = list(my_dict) # 사전 -> 리스트
my_set_2 = set(my_tuple) # 튜플 -> 집합
my_dict_2 = dict(my_tuple) # 튜플 -> 사전 (불가능)
```