Dictionary, 딕셔너리

> 참고| **해시**
>
> 딕셔너리는 해시(Hash) 기법을 이용해서 데이터를 저장합니다. 보통 딕셔너리와 같은 키-값 형태의 자료형을 해시, 해시 맵, 해시테이블 등으로 부르기도 합니다.

정교수님은 학생들의 정보를 리스트에 정리하였습니다.

다음이 정교수님이 학생들의 정보를 취합한 방식입니다.

```
# 학번, 이름, 태도점수, 참여점수, 중간평가점수, 기말평가점수
python_student = [
    [1, '이익준', 'A', 'A+', 'A', 'A'],
    [2, '채송화', 'A+', 'A', 'A+', 'A+'],
    [3, '안정원', 'A+', 'A', 'B+', 'B']
]
```

정교수님이 작업을 하던 중에 인덱스에 위치하는 정보들이 무엇인지 까먹는 문제가 발생하였습니다.

정교수님은 인덱스에서 무엇을 의미하는지 알고 싶어합니다.

이럴때 우리는 사전형(dictionary) 자료형을 사용하여 해결할 수 있습니다.

사용법은 다음과 같습니다.

1. 딕셔너리 할당

빈 딕셔너리를 만들 때는 2가지 방법이 있습니다.

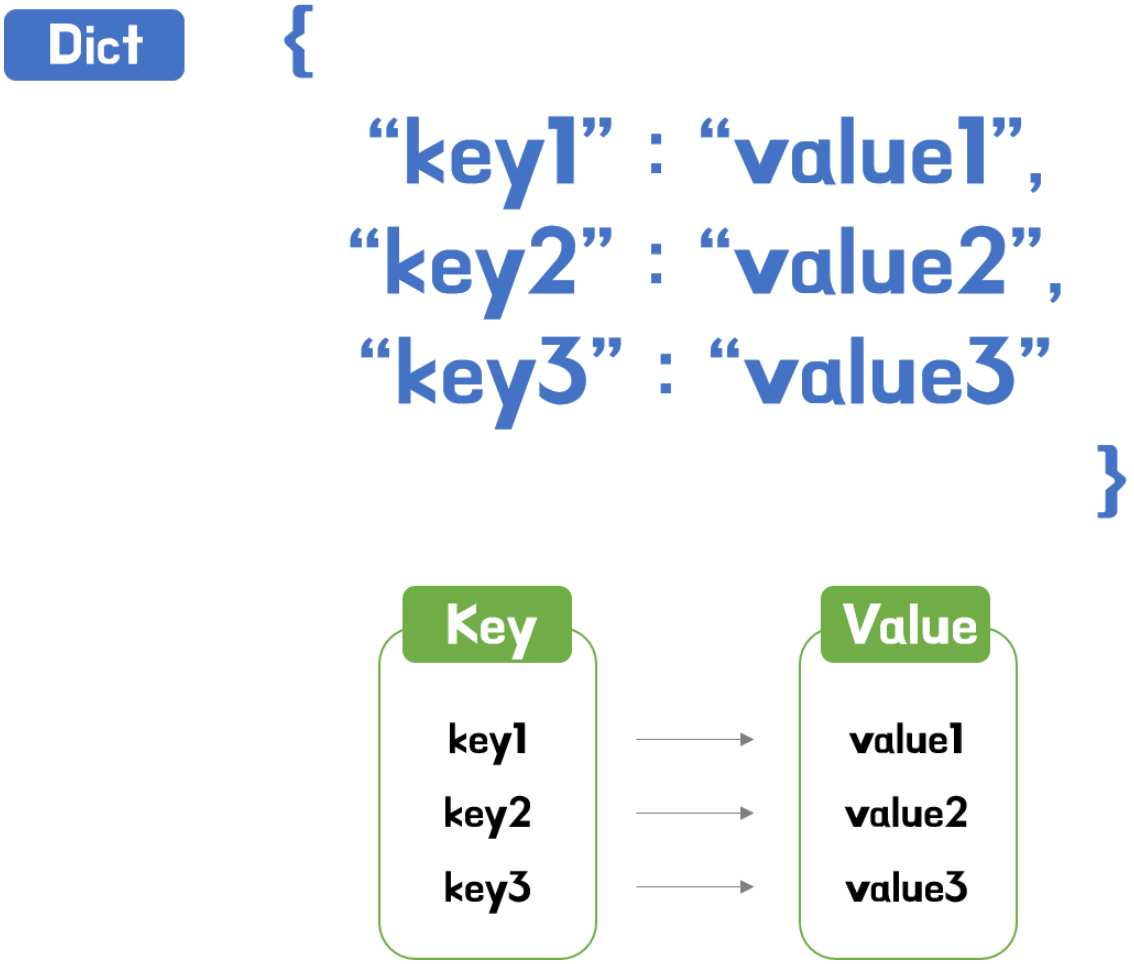
1. dict()
2. {}

`{}`는 이후에 배울 `set, 집합`에서도 사용됩니다.

key: value값이 있다면 딕셔너리 자료형이고, 값만 들어간다면 집합 자료형입니다.

```
dict_data = {
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
}

print(dict_data) # {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```



2. 딕셔너리 key, value 사용해보기

문자열과 리스트, 튜플의 경우에는 인덱싱을 통해서 값 하나하나에 접근할 수 있었습니다.

딕셔너리의 경우에는 인덱스에 키값을 넣어서 값을 가져올 수 있습니다.

사용법은 이와 `dict_data['key1']` 같습니다.

```
dict_data = {
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
}

# dict_data key값 key1에 value 가져와보기
print(dict_data["key1"]) # value1

# dict_data key값에 접근하여 value값 변경하기
dict_data["key2"] = 123
print(dict_data) # {'key1': 'value1', 'key2': 123, 'key3': 'value3'}

print(dict_data["val"]) # KeyError Traceback (most recent call last)
```

위의 결과로 보았을 때 알수있듯, dictionary는 수정이 가능(**mutable**)합니다.

주의]

> **Dictionary key값**
> dict 자료형에서 key값은 수정 불가한 객체(immutable)만 가능합니다.
> 키값이 중복되었을 때는 가장 아래에 쓰인 값으로 재할당됩니다.

```
dict_data = {
    1: "숫자로 키값 만들기",
    "1": "숫자(?)로 키값 만들기",
    "1": "문자로 키값 만들기",
    (1,2,3): "튜플로 키값 만들기"
}

print(dict_data) # {1: '숫자로 키값 만들기', '1': '문자로 키값 만들기', (1, 2, 3):
'튜플로 키값 만들기'}

dict_data = {
    [1,2,3]: "리스트는 mutable하기에 키값이 될수 없습니다."
} # #TypeError: unhashable type: 'list'

print(dict_data)
```

3. 딕셔너리 key, value 추가, 삭제

- 1) 추가

```
dict_data = {}
dict_data[1] = "일"
dict_data[2] = "이"
print(dict_data) # {1: '일', 2: '이'}
```

- 2) 삭제

```
dict_data = {
    1: "일",
    2: "이"
}

# 딕셔너리 요소 삭제
del dict_data[2] # 해당 2는 인덱스가 아닌 key값

print(dict_data) # {1: '일'}
```

4. 딕셔너리 관련 함수들

- 1) keys

`keys` 함수는 딕셔너리의 키값들을 반환합니다.

사용법은 `dict_data.keys()` 입니다.

```
dict_data = {
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

print(dict_data.keys()) # dict_keys(['python', 'list', 'dict'])
```

- 2) values

`values` 함수는 딕셔너리의 값들을 반환합니다.

사용법은 `dict_data.values()` 입니다.

```
dict_data = {
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

print(dict_data.values()) # dict_values(['파이썬', '리스트', '딕셔너리'])
```

- 3) items

`items` 함수는 딕셔너리의 키, 값들을 반환합니다.

사용법은 `dict_data.items()` 입니다.

```
dict_data = {
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

print(dict_data.items()) # dict_items([('python', '파이썬'), ('list', '리스트'),
('dict', '딕셔너리')])
```

- 4) clear

`clear` 함수는 딕셔너리 안의 모든 요소를 삭제합니다.

사용법은 `dict_data.clear()` 입니다.

```
dict_data = {
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

dict_data.clear()

print(dict_data) # {} : 빈 딕셔너리
```

- 5) get

`get` 함수는 딕셔너리 키값을 이용해서 값을 반환받는 함수입니다.

`dict_data['key값']` 으로 접근하는 방식과 동일한 값을 얻을 수 있습니다.

차이점은 해당 key값에 대한 값이 없을 때입니다.

`dict_data['key값']` 으로 접근할 때는 값이 없으면 Error가 발생하지만,

`dict_data.get('key값')` 으로 접근을 할 때 값이 없으면 None이 반환됩니다.

참고]

> **None**
> `None` 자료형은 아무것도 없는 상태를 나타내는 자료형입니다.

```
dict_data = {
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

python = dict_data.get('python')
print(python) # 파이썬

null_data = dict_data.get('Python')
print(null_data) # None
```

- 6) in

구성원 연산자에 있는 `in` 을 써서 해당 값이 있는지 없는지 판별할 수 있습니다.

결과값은 Bool자료형인 `True` / `False` 로 나오게 됩니다.

사용법은 `해당값 in iterable객체` 입니다.

기본적인 딕셔너리에서 해당값과 비교되는 요소는 key값들입니다.

```
dict_data = {
```

```
    "python": "파이썬",
    "list": "리스트",
    "dict": "딕셔너리"
}

# Key값 사용했을 때 결과
result = 'python' in dict_data
print(result) # True

# Value값 사용했을 때 결과
result = '파이썬' in dict_data
print(result) # False

# Key에 없는 값 사용했을 때 결과
result = '없음' in dict_data
print(result) # False
```