딕셔너리 자료형

딕셔너리란?

사람은 누구든지 "이름" = "홍길동", "생일" = "몇 월 며칠" 등으로 구별할 수 있다. 파이썬은 영리하게도 이러한 대응 관계를 나타낼 수 있는 자료형을 가지고 있다. 요즘 사용하는 대부분의 언어도 이러한 대응 관계를 나타내는 자료형을 갖고 있는데, 이를 연관 배열(Associative array) 또는 해시(Hash)라고 한다.

파이썬에서는 이러한 자료형을 딕셔너리(Dictionary)라고 하는데, 단어 그대로 해석하면 사전이라는 뜻이다. 즉 "people"이라는 단어에 "사람", "baseball"이라는 단어에 "야구"라는 뜻이 부합되듯이 딕셔너리는 Key와 Value를 한 쌍으로 갖는 자료형이다. 예컨대 Key가 "baseball"이라면 Value는 "야구"가 될 것이다.

딕셔너리는 리스트나 튜플처럼 순차적으로(sequential) 해당 요솟값을 구하지 않고 Key를 통해 Value를 얻는다. 이것이 바로 딕셔너리의 가장 큰 특징이다. baseball이라는 단어의 뜻을 찾기 위해 사전의 내용을 순차적으로 모두 검색하는 것이 아니라 baseball이라는 단어가 있는 곳만 펼쳐 보는 것이다.

딕셔너리는 어떻게 만들까?

다음은 기본 딕셔너리의 모습이다.

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

Key와 Value의 쌍 여러 개가 { }로 둘러싸여 있다. 각각의 요소는 Key : Value 형태로 이루어져 있고 쉼표(,)로 구분되어 있다.

> ※ Key에는 변하지 않는 값을 사용하고, Value에는 변하는 값과 변하지 않는 값 모두 사용할 수 있다.

다음 딕셔너리 예를 살펴보자.

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

위에서 Key는 각각 'name', 'phone', 'birth'이고, 각각의 Key에 해당하는 Value는 'pey', '0119993323', '1118'이 된다.

딕셔너리 dic의 정보

key	value
name	pey
phone	01199993323
birth	1118

다음 예는 Key로 정수 값 1, Value로 문자열 'hi'를 사용한 예이다.

```
>>> a = {1: 'hi'}
```

또한 다음 예처럼 Value에 리스트도 넣을 수 있다.

```
>>> a = { 'a': [1,2,3]}
```

딕셔너리 쌍 추가, 삭제하기

딕셔너리 쌍을 추가하는 방법과 삭제하는 방법을 살펴보자. 먼저 딕셔너리에 쌍을 추가하는 다음 예를 함께 따라 해 보자.

딕셔너리 쌍 추가하기

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
```

{1: 'a'} 딕셔너리에 a[2] = 'b'와 같이 입력하면 딕셔너리 a에 Key와 Value가 각각 2와 'b'인 2 : 'b'라는 딕셔너리 쌍이 추가된다.

```
>>> a['name'] = 'pey'
>>> a
{1: 'a', 2: 'b', 'name': 'pey'}
```

딕셔너리 a에 'name': 'pey'라는 쌍이 추가되었다.

```
>>> a[3] = [1,2,3]
>>> a
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

Key는 3, Value는 [1, 2, 3]을 가지는 한 쌍이 또 추가되었다.

딕셔너리 요소 삭제하기

```
>>> del a[1]
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

위 예제는 딕셔너리 요소를 지우는 방법을 보여 준다. del 함수를 사용해서 del a[key]처럼 입력하면 지정한 Key에 해당하는 {key : value} 쌍이 삭제된다.

딕셔너리를 사용하는 방법

"딕셔너리는 주로 어떤 것을 표현하는 데 사용할까?"라는 의문이 들 것이다. 예를 들어 4명의 사람이 있다고 가정하고, 각자의 특기를 표현할 수 있는 좋은 방법에 대해서 생각해 보자. 리스트나 문자열로는 표현하기가 상당히 까다로울 것이다. 하지만 파이썬의 딕셔너리를 사용한다면 이 상황을 표현하기가 정말 쉽다. 다음 예를 보자.

```
{"김연아":"피겨스케이팅", "류현진":"야구", "박지성":"축구", "귀도":"파이썬"}
```

사람 이름과 특기를 한 쌍으로 하는 딕셔너리이다. 정말 간편하지 않은가?

지금껏 우리는 딕셔너리를 만드는 방법에 대해서만 살펴보았는데 딕셔너리를 제대로 활용하기 위해서는 알아야 할 것이 더 있다. 이제부터 하나씩 알아보자.

딕셔너리에서 Key 사용해 Value 얻기

다음 예를 살펴보자.

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']
10
>>> grade['julliet']
99
```

리스트나 튜플, 문자열은 요솟값을 얻고자 할 때 인덱싱이나 슬라이싱 기법 중 하나를 사용했다. 하지만 딕셔너리는 단 한 가지 방법뿐이다. 바로 Key를 사용해서 Value를 구하는 방법이다. 위 예에서 'pey'라는 Key의 Value를 얻기 위해 grade['pey']를 사용한 것처럼 어떤 Key의 Value를 얻기 위해서는 `딕셔너리변수이름[Key]` 를 사용한다.

몇 가지 예를 더 보자.

```
>>> a = {1:'a', 2:'b'}
>>> a[1]
'a'
>>> a[2]
'b'
```

먼저 a 변수에 {1:'a', 2:'b'} 딕셔너리를 대입하였다. 위 예에서 볼 수 있듯이 a[1]은 'a' 값을 돌려준다. 여기에서 a[1]이 의미하는 것은 리스트나 튜플의 a[1]과는 전혀 다르다. 딕셔너리 변수에서 [ ] 안의 숫자 1은 두 번째 요소를 뜻하는 것이 아니라 Key에 해당하는 1을 나타낸다. 앞에서도 말했듯이 딕셔너리는 리스트나 튜플에 있는 인덱싱 방법을 적용할 수 없다. 따라서 a[1]은 딕셔너리 {1:'a', 2:'b'}에서 Key가 1인 것의 Value인 'a'를 돌려주게 된다. a[2] 역시 마찬가지이다.

이번에는 a라는 변수에 앞의 예에서 사용한 딕셔너리의 Key와 Value를 뒤집어 놓은 딕셔너리를 대입해 보자.

```
>>> a = {'a':1, 'b':2}
>>> a['a']
1
>>> a['b']
2
```

역시 a['a'], a['b']처럼 Key를 사용해서 Value를 얻을 수 있다. 정리하면, 딕셔너리 a는 a[Key]로 입력해서 Key에 해당하는 Value를 얻는다.

다음 예는 이전에 한 번 언급한 딕셔너리인데 Key를 사용해서 Value를 얻는 방법을 잘 보여 준다.

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> dic['name']
'pey'
>>> dic['phone']
'0119993323'
>>> dic['birth']
'1118'
```

딕셔너리 만들 때 주의할 사항

먼저 딕셔너리에서 Key는 고유한 값이므로 중복되는 Key 값을 설정해 놓으면 하나를 제외한 나머지 것들이 모두 무시된다는 점을 주의해야 한다. 다음 예에서 볼 수 있듯이 동일한 Key가 2개 존재할 경우 1:'a' 쌍이 무시된다.

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

이렇게 Key가 중복되었을 때 1개를 제외한 나머지 Key:Value 값이 모두 무시되는 이유는 Key를 통해서 Value를 얻는 딕셔너리의 특징에서 비롯된다. 즉 동일한 Key가 존재하면 어떤 Key에 해당하는 Value를 불러야 할지 알 수 없기 때문이다.

또 한 가지 주의해야 할 사항은 Key에 리스트는 쓸 수 없다는 것이다. 하지만 튜플은 Key로 쓸 수 있다. 딕셔너리의 Key로 쓸 수 있느냐 없느냐는 Key가 변하는(mutable) 값인지 변하지 않는(immutable) 값인지에 달려 있다. 리스트는 그 값이 변할 수 있기 때문에 Key로 쓸 수 없다. 다음 예처럼 리스트를 Key로 설정하면 리스트를 키 값으로 사용할 수 없다는 오류가 발생한다.

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

따라서 딕셔너리의 Key 값으로 딕셔너리를 사용할 수 없음은 당연하다. 단 Value에는 변하는 값이든 변하지 않는 값이든 상관없이 아무 값이나 넣을 수 있다.

딕셔너리 관련 함수들

딕셔너리를 자유자재로 사용하기 위해 딕셔너리가 자체적으로 가지고 있는 관련 함수를 사용해 보자.

Key 리스트 만들기(keys)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

a.keys()는 딕셔너리 a의 Key만을 모아서 dict_keys 객체를 돌려준다.

점프 투 파이썬

[파이썬 3.0 이후 버전의 keys 함수, 어떻게 달라졌나?]

파이썬 2.7 버전까지는 a.keys() 함수를 호출할 때 반환 값으로 dict_keys가 아닌 리스트를 돌려준다. 리스트를 돌려주기 위해서는 메모리 낭비가 발생하는데 파이썬 3.0 이후 버전에서는 이러한 메모리 낭비를 줄이기 위해 dict_keys 객체를 돌려준다. 다음에 소개할 dict_values, dict_items 역시 파이썬 3.0 이후 버전에서 추가된 것들이다. 만약 3.0 이후 버전에서 반환 값으로 리스트가 필요한 경우에는 `list(a.keys())` 를 사용하면 된다. dict_keys, dict_values, dict_items 등은 리스트로 변환하지 않더라도 기본적인 반복(iterate) 구문(예: for문)을 실행할 수 있다.

dict_keys 객체는 다음과 같이 사용할 수 있다. 리스트를 사용하는 것과 차이가 없지만, 리스트 고유의 append, insert, pop, remove, sort 함수는 수행할 수 없다.

```
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

> ※ print(k)를 입력할 때 들여쓰기를 하지 않으면 오류가 발생하니 주의하자. for문 등 반복 구문에 대해서는 03장에서 자세히 살펴본다.

dict_keys 객체를 리스트로 변환하려면 다음과 같이 하면 된다.

```
>>> list(a.keys())
['name', 'phone', 'birth']
```

Value 리스트 만들기(values)

```
>>> a.values()
dict_values(['pey', '0119993323', '1118'])
```

Key를 얻는 것과 마찬가지 방법으로 Value만 얻고 싶다면 values 함수를 사용하면 된다. values 함수를 호출하면 dict_values 객체를 돌려준다.

Key, Value 쌍 얻기(items)

```
>>> a.items()
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

items 함수는 Key와 Value의 쌍을 튜플로 묶은 값을 dict_items 객체로 돌려준다. dict_values 객체와 dict_items 객체 역시 dict_keys 객체와 마찬가지로 리스트를 사용하는 것과 동일하게 사용할 수 있다.

Key: Value 쌍 모두 지우기(clear)

```
>>> a.clear()
>>> a
{}
```

clear 함수는 딕셔너리 안의 모든 요소를 삭제한다. 빈 리스트를 [ ], 빈 튜플을 ( )로 표현하는 것과 마찬가지로 빈 딕셔너리도 { }로 표현한다.

Key로 Value얻기(get)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

get(x) 함수는 x라는 Key에 대응되는 Value를 돌려준다. 앞에서 살펴보았듯이 a.get('name')은 a['name']을 사용했을 때와 동일한 결괏값을 돌려받는다.

다만 다음 예제에서 볼 수 있듯이 a['nokey']처럼 존재하지 않는 키(nokey)로 값을 가져오려고 할 경우 a['nokey']는 Key 오류를 발생시키고 a.get('nokey')는 None을 돌려준다는 차이가 있다. 어떤것을 사용할지는 여러분의 선택이다.

> ※ 여기에서 None은 "거짓"이라는 뜻이라고만 알아두자.

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

딕셔너리 안에 찾으려는 Key 값이 없을 경우 미리 정해 둔 디폴트 값을 대신 가져오게 하고 싶을 때에는 get(x, '디폴트 값')을 사용하면 편리하다.

```
>>> a.get('foo', 'bar')
'bar'
```

a 딕셔너리에는 'foo'에 해당하는 값이 없다. 따라서 디폴트 값인 'bar'를 돌려준다.

해당 Key가 딕셔너리 안에 있는지 조사하기(in)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

'name' 문자열은 a 딕셔너리의 Key 중 하나이다. 따라서 'name' in a를 호출하면 참(True)을 돌려준다. 반대로 'email'은 a 딕셔너리 안에 존재하지 않는 Key이므로 거짓(False)을 돌려준다.