파이썬 프로그래밍

# 사전


한국기술교육대학교
온라인평생교육원

## ▣ 사전 활용법

### 1. 사전의 특징

- 집합적 자료형
- 자료의 순서를 정하지 않는 매핑(Mapping)형
  - 키(Key)를 이용하여 값(Value)에 접근
  - 시퀀스 자료형은 아님
- 키와 값의 매핑 1개를 아이템(item)이라고 부름

```
member = {'basketball':5, 'soccer':11, 'baseball':9}
print member['baseball'] # 검색
```

```
9
```

- 사전 = 리스트, 튜플과 함께 가장 많이 활용되는 내장자료형
- 집합적 자료형 = 사전이라는 자료 내 여러 개의 객체 존재 가능
- 매핑형 자료형은 사전이 유일
- 콤마(,) 단위가 사전의 원소 개수
- 콤마 단위의 원소 = 아이템
- 아이템 → '키 : 값' 으로 구성
- 인덱싱 X, 사전의 검색 연산

## ▣ 사전 활용법

### 1. 사전의 특징

- 값을 저장할 시에 키를 사용
  - 키가 없다면 새로운 키와 값의 아이템이 생성
  - 키가 이미 존재한다면 그 키에 해당하는 값이 변경

```
member = {'basketball':5, 'soccer':11, 'baseball':9}
member['volleyball'] = 7 # 새로운 아이템 설정
member['volleyball'] = 6 # 변경
print member
print len(member)        # 아이템의 개수 반환
```

```
{'soccer': 11, 'basketball': 5, 'baseball': 9, 'volleyball': 6}
4
```

- 새로운 아이템 설정 → member['새로운 key'] = '새로운 value'
- 아이템 변경 → member['기존 key'] = '변경할 value'
- len(member) → member 안에 존재하는 item 개수

## ▣ 사전 활용법

### 2. 해쉬 기법

- 사전을 출력하면 각 아이템들이 임의의 순서로 출력된다.
- 새로운 아이템이 들어오면 키 내용에 따라 그 순서가 달라진다.
- 내부적으로 키 내용에 대해 해쉬(Hash) 기법을 사용
  - 검색 속도가 매우 빠름
  -[참고]: http://www.laurentluce.com/posts/python-dictionary-implementation/
- 키와 값 매핑에 대한 아이템을 삭제할 때에는 del과 함께 키값 명시

```
member = {'basketball':5, 'soccer':11, 'baseball':9}
del member['basketball'] # 항목 삭제
print member
```

```
{'soccer': 11, 'baseball': 9}
```

- 사전 내 아이템 순서는 존재하지 않음
- 내부적으로 멤버를 파이썬에 구현할 때는 해쉬 기법 사용
- 해쉬 방법 → 각각의 키에 내부적으로 존재하는 인덱스를 붙임
- 해쉬 값 = 인덱스 값
- 내부적으로 자료를 저장하는 방법이 있음 → 그 순서대로 출력
- 임의의 순서대로 저장되어 있는 값을 확인 가능
- baseball 키를 해쉬함수로 돌려서 해쉬 값을 가지고 value를 찾는 것
- Key와 value에 설정 가능한 자료형은?

- [중요] 키는 변경 불가능 (Immutable) 자료만 가능
  - 문자열, 숫자, 튜플은 가능
  - 리스트, 사전은 키가 될 수 없음
- 반면에 사전에 입력되는 값은 임의의 객체

## ▣ 사전 활용법

### 2. 해쉬 기법

```
d = {}
d['str'] = 'abc'
d[1] = 4
d[(1,2,3)] = 'tuple'
d[[1,2,3]] = 'list' # 리스트는 키가 될 수 없다.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4f2e8eb1eca7> in <module>()
      3 d[1] = 4
      4 d[(1,2,3)] = 'tuple'
----> 5 d[[1,2,3]] = 'list' # 리스트는 키가 될 수 없다.

TypeError: unhashable type: 'list'
```

- 'str' = key, 'abc' = value
- Key와 value에는 문자열 가능
- Key와 value에 정수 가능
- Key 와 value에 tuple 가능
- [1, 2, 3] → list로 변경이 가능하여 해쉬함수로 돌릴 수 X
- 변경 가능한 자료형 = 리스트, 사전

```
d[{1:2}] = 3 # 사전은 키가 될 수 없다.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-de0c91623d77> in <module>()
----> 1 d[{1:2}] = 3 # 사전은 키가 될 수 없다.

TypeError: unhashable type: 'dict'
```

- 리스트와 사전은 key가 될 수 없음

## ▣ 사전 활용법

### 2. 해쉬 기법

- 함수 이름은 사전의 키나 값으로 사용 가능함

```
def add(a, b):
    return a + b

def sub(a, b):
    return a - b

action = {0: add, 1: sub}   # 함수 이름을 사전의 값으로 사용
print action[0](4, 5)
print action[1](4, 5)
print
action2 = {add: 1, sub: 2}  # 함수 이름을 사전의 키로 사용
print action2[add]
```

```
9
-1

1
```

- 함수 이름은 사전의 key나 value로 사용 가능
- add → 함수가 객체로 인식 → 객체의 레퍼런스 값을 add가 가짐
- 파이썬에서는 모든 것이 객체
- action[0] → add라는 value가 반환됨
- add는 함수를 가리킴 → (4,5)는 함수의 인자로 함수 호출
- action[1] → sub value가 반환됨
- sub는 함수를 가리킴 → (4,5)는 함수의 인자로 함수 호출
- action2는 key 자리에 함수를 가짐
- 검색을 add로 하여 add의 value 값 1 출력

## ▣ 사전 활용법

### 3. dict 내장함수

- 사전을 생성하는 다른 방법: 내장함수 dict() 사용

```
d = dict()
print type(d)
print

print dict(one=1, two=2)
print dict([('one', 1), ('two', 2)])
print dict({'one':1, 'two':2})
```

```
<type 'dict'>

{'two': 2, 'one': 1}
{'two': 2, 'one': 1}
{'two': 2, 'one': 1}
```

- dict( ) → 비어 있는 공백 사전 함수가 나옴
- one=1 → one은 key, 1은 value로 들어감
- 안에 있는 item 끼리 순서가 없음 → 임의의 순서로 저장
- 리스트 안 각각의 원소는 튜플
- 튜플 첫번째 원소 → key, 두번째 원소 → value

## ▣ 사전 활용법

### 3. dict 내장함수

```
keys = ['one', 'two', 'three']
values = (1, 2, 3)
print zip(keys, values) # zip(): 두 개의 자료를 순서대로 쌍으로 묶은 튜플들의
                                리스트 반환
print dict(zip(keys, values))
```

```
[('one', 1), ('two', 2), ('three', 3)]
{'three': 3, 'two': 2, 'one': 1}
```

- 내장함수 zip
- zip의 원소로 시퀀스 자료형 2개 사용
- 두 개의 자료를 순서대로 쌍으로 묶은 튜플들의 리스트 반환
- zip 함수를 dict 함수의 원소로 사용 가능