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

파이썬 프로그래밍

# 사전


한국기술교육대학교
온라인평생교육원

## ▣ 사전 메소드

### 1. 중요 메소드

- D.keys(): 사전 D에서 키들을 리스트로 반환
- D.values(): 사전 D에서 값들을 리스트로 반환
- D.items(): 사전 D에서 각 아이템을 튜플형태로 가져와 리스트로 반환
- key in D: 사전 D안에 key를 키값을 가진 아이템이 있는지 확인

```
phone = {'jack':9465215, 'jin':1111, 'Joseph':6584321}

print phone.keys() # 키의 리스트 반환
print phone.values() # 값들의 리스트 반환
print phone.items() # (키, 값)의 리스트 반환
print
print 'jack' in phone # 'jack'이 phone의 키에 포함되어 있는가?
print 'lee' in phone
```

```
['jin', 'Joseph', 'jack']
[1111, 6584321, 9465215]
[('jin', 1111), ('Joseph', 6584321), ('jack', 9465215)]

True
False
```

- phone.keys( ) → 리스트 안에 각각의 사전 키들만 반환
- 순서대로 반환 X → 내부에서 정한대로 반환
- 사전은 순서가 없기 때문에 임의의 순서대로 반환됨
- phone.values( ) → 리스트 안에 각각의 사전 값들만 반환
- phone.items( ) → (키, 값)으로 튜플이 원소가 되어 리스트 반환
- 'key' in 사전 이름 → 그 사전 안에 'key'가 있는지?
- in 이라는 키워드는 'key'에 대한 검색만 가능
- Phone.values( ) → 사전의 value을 의미하므로 in으로 value 검색 가능

## ▣ 사전 메소드

### 1. 중요 메소드

- D2 = D.copy(): 사전 D를 복사하여 D2 사전에 할당한다.

```
phone = {'jack':9465215, 'jin':1111, 'Joseph':6584321}
p = phone # 사전 레퍼런스 복사. 사전 객체는 공유된다.

phone['jack'] = 1234 # phone을 변경하면
print phone
print p # p도 함께 변경된다.
print

ph = phone.copy() # 사전복사. 별도의 사전 객체가 마련된다.
phone['jack'] = 1111 # phone을 바꿔도
print phone
print ph # ph는 바뀌지 않는다.
```

```
{'jin': 1111, 'Joseph': 6584321, 'jack': 1234}
{'jin': 1111, 'Joseph': 6584321, 'jack': 1234}

{'jin': 1111, 'Joseph': 6584321, 'jack': 1111}
{'jin': 1111, 'Joseph': 6584321, 'jack': 1234}
```

- phone 변수는 사전 객체를 레퍼런스하고 있는 참조 값을 지님
- p = phone → phone 자체가 가지고 있는 참조값을 p에 할당
- p와 phone은 동일한 사전 객체를 가리킴
- phone이 변경되면 p도 동일하게 변경됨
- phone.copy( ) → 기존 phone 을 그대로 복사해서 새로운 객체 생성
- phone과 ph는 서로 다른 객체를 가리킴

## ▣ 사전 메소드

### 1. 중요 메소드

- [주의] D.copy()는 Shallow Copy를 수행한다.

```
phone = {'a': [1,2,3], 'b': 4}
phone2 = phone.copy()
print phone
print phone2
print

phone['b'] = 100
print phone
print phone2
print

phone['a'][0] = 100
print phone
print phone2
```

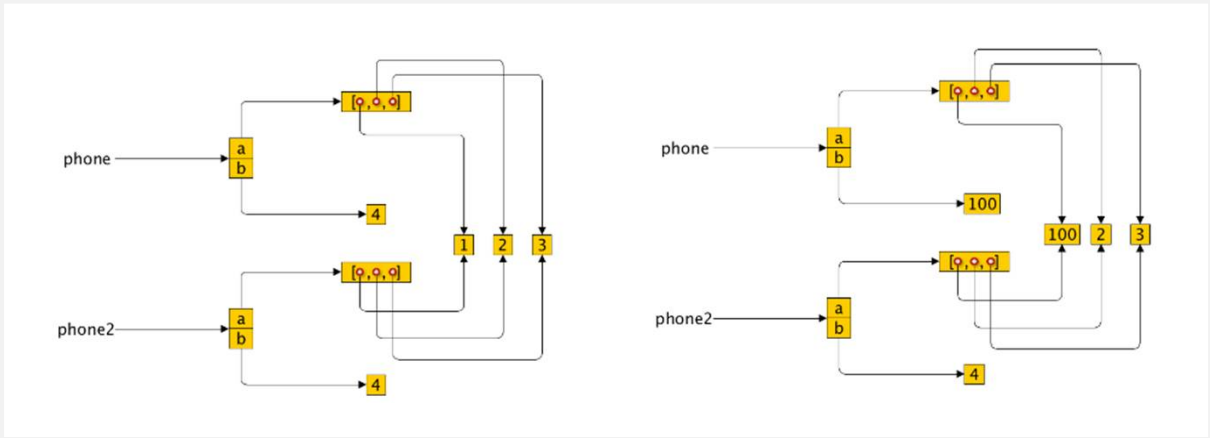
```
{'a': [1, 2, 3], 'b': 4}
{'a': [1, 2, 3], 'b': 4}

{'a': [1, 2, 3], 'b': 100}
{'a': [1, 2, 3], 'b': 4}

{'a': [100, 2, 3], 'b': 100}
{'a': [100, 2, 3], 'b': 4}
```

## ▣ 사전 메소드

### 1. 중요 메소드




- phone.copy( ) → shallow copy
- a key → 리스트, b key → 4
- 정수 값은 복사가 되면서 똑같지만 다른 새로운 객체 생성
- 리스트 안 존재하는 1, 2, 3 원소는 공유가 됨
- 1, 2, 3 자체가 copy되어 별도의 1, 2, 3 존재 X
- Shallow copy 반대 → Deep copy
- Deep copy → 공유된 리스트 1, 2, 3을 별도로 만들어 줌
- Shallow copy → 복사하려는 리스트 안 원소까지는 복사 X
- copy( ) → deep copy가 아니라 보통 shallow copy

## ▣ 사전 메소드

### 1. 중요 메소드

```
ph = {'jack':9465215, 'jin':1111, 'Joseph':6584321}

print ph.get('jack')  # 'jack'에 대한 값을 얻는다. ph['jack']과 같다.
print ph.get('gslee') # 'gslee'에 대한 값을 얻는다. 값이 없는 경우 None반환
```

```
9465215
None
```

- ph.get('jack') → jack 키에 있는 value 값 가져옴
- get( ) → key를 주면서 value를 검색하는 메소드
- ph['jack'] = ph.get('jack')

```
ph = {'jack':9465215, 'jin':1111, 'Joseph':6584321}
print ph['gslee'] # ph['gslee']는 키가 없는 경우 예외발생
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-11-00dc298a68ec> in <module>()
      1 ph = {'jack':9465215, 'jin':1111, 'Joseph':6584321}
----> 2 print ph['gslee']      # ph['gslee']는 키가 없는 경우 예외발생

KeyError: 'gslee'
```

- ph['gslee'] → 꺽쇠가로는 값이 없으면 error 발생
- 값이 없을 때 ph['a'] = ph.get('a')의 출력결과가 다름

## ▣ 사전 메소드

### 1. 중요 메소드

```
ph = {'jack':9465215, 'jin':1111, 'Joseph':6584321}
print ph.get('gslee', 5284) # 인수를 하나 더 제공하면 'gslee'가 없는 경우에 5284 리턴
print ph # 사전에는 변화가 없다
print

print ph.popitem()              # 임의의 아이템을 꺼낸다.
print ph
print

print ph.popitem()              # 임의의 아이템을 꺼낸다.
print ph
print

print ph.pop('jack')            # 키 값을 통해 해당 아이템을 지정하여 꺼낸다.
print ph
```

```
5284
{'jin': 1111, 'Joseph': 6584321, 'jack': 9465215}

('jin', 1111)
{'Joseph': 6584321, 'jack': 9465215}

('Joseph', 6584321)
{'jack': 9465215}

9465215
{}
```

- get 에서 인수를 하나 더 제공하면 gslee가 없을 경우 5284 리턴
- get은 본래 사전에 변화를 주는 메소드 X
- pop.item( ) → 임의의 item을 꺼내는 메소드
- pop('key') → key 값을 통해 해당 item을 지정하여 꺼냄
- pop이 돌려주는 것은 value 값이지만 실제는 item 반환

## ▣ 사전 메소드

### 1. 중요 메소드

```
phone = {'jack':9465215, 'jin':1111, 'Joseph':6584321}
ph = {'kim':12312, 'lee': 9090}
phone.update(ph) # 사전 phone의 내용을 ph으로 추가 갱신
print phone
print
phone.clear() # 사전의 모든 입력을 없앤다.
print phone
```

```
{'jin': 1111, 'Joseph': 6584321, 'jack': 9465215, 'kim': 12312, 'lee': 9090}

{}
```

- phone.update(ph ) → phone의 내용을 ph로 추가하여 업데이트
- clear 메소드 → 전체 사전 내용을 없앰

파이썬 프로그래밍

# 사전


한국기술교육대학교
온라인평생교육원

## ▣ 루프를 이용한 사전 내용 참조

### 1. 루프를 이용한 사전 내용 참조

- 사전의 모든 키값을 순차적으로 참조하는 방법

```
D = {'a':1, 'b':2, 'c':3}
for key in D.keys():
    print key, D[key]
```

```
a 1
c 3
b 2
```

- for~in 구문에 대해 학습
- D.keys( ) → key 값만 리스트 안에 담겨 리턴
- D[key] → key에 해당하는 value 값이 반환됨

- 사전 자체를 for루프에 활용하면 키에 대한 루프가 실행된다.

```
D = {'a':1, 'b':2, 'c':3}
for key in D:
    print key, D[key]
```

```
a 1
c 3
b 2
```

- for~in 구문 뒤에 사전을 넣을 시 사전의 key가 반환됨
- D.keys( ) = D

## ▣ 루프를 이용한 사전 내용 참조

### 1. 루프를 이용한 사전 내용 참조

- 키와 값을 동시에 참조하려면 D.items()를 활용한다.

```
for key, value in D.items():
    print key, value
```

```
a 1
c 3
b 2
```

- D.items( ) → 키와 값이 튜플형태로 반환 → 다시 언 패킹됨
- D.values( ) → value 값만 리스트로 반환

## ▣ 루프를 이용한 사전 내용 참조

### 1. 루프를 이용한 사전 내용 참조

- 사전에 입력된 아이템들은 일정한 순서가 없다.
- 키값에 대한 정렬은 아이템들을 리스트로 뽑은 다음에 해당 리스트에 있는 sort() 함수를 활용한다.

```
D = {'a':1, 'b':2, 'c':3}
items = D.items()
print items
print

items.sort()
print items
print

for k,v in items:
    print k, v
```

```
[('a', 1), ('c', 3), ('b', 2)]

[('a', 1), ('b', 2), ('c', 3)]

a 1
b 2
c 3
```

- sort 메소드 → 튜플의 첫번째 값을 기준으로 정렬