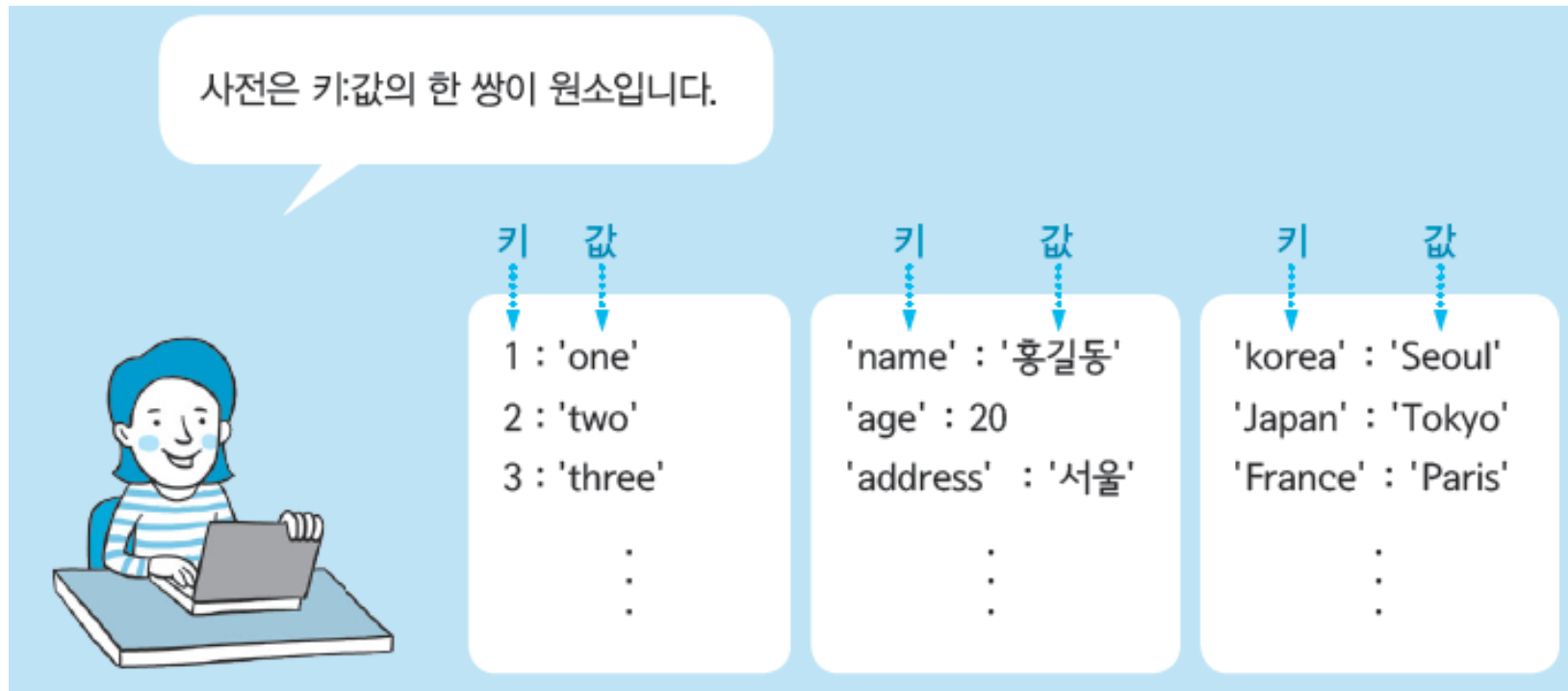


12. 사전 자료형

1. 사전 만들기
2. 사전은 '키'를 인덱스로 사용하고 슬라이스는 할 수 없습니다
3. 사전은 mutable 객체입니다
4. 사전에 in, not in 연산자 사용하기
5. 사전에 함수 적용하기
6. 사전 메소드
7. 사전과 for 반복문
8. 사전 내에서 for 반복문 사용하기 (Dictionary Comprehension)
9. 사전 출력하기 (pprint 모듈)
10. 사전의 값으로 mutable 자료형이 저장되는 경우
11. 정리

1. 사전 만들기

- ◆ 사전은 연관된 데이터를 하나의 쌍으로 묶어서 관리함
- ◆ 활용도가 아주 높은 mutable 자료형임



1. 사전 만들기

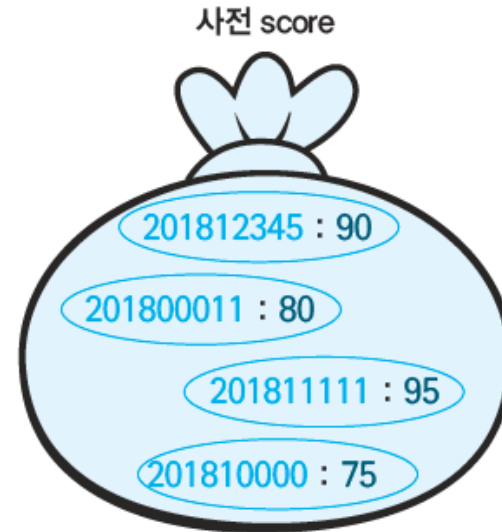
- ◆ 사전은 집합과 같이 중복이 없고, 순서 개념이 없음.
- ◆ 사전의 각 원소는 ‘키:값’의 쌍으로 구성되는데, 이 때 중복된 ‘키’가 없어야 함. 즉, 키의 중복이 없어야 함.
- ◆ 사전은 다음과 같이 생성함.

사전명 = { 키1:값1, 키2:값2, 키3:값3, ... }

1. 사전 만들기

◆ 사전의 예

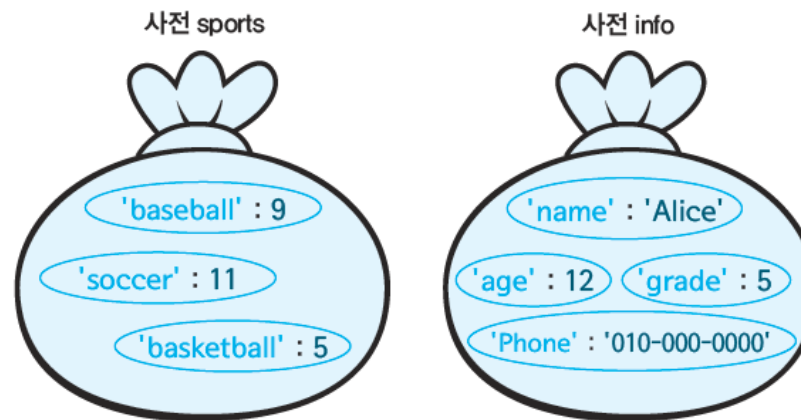
키	값
학번	성적
201812345	90
201800011	80
201811111	95
201810000	75



```
>>> score = {201812345:90, 201800011:80, 201811111:95, 201810000:75}  
>>> print(score)  
{201812345: 90, 201800011: 80, 201811111: 95, 201810000: 75}
```

1. 사전 만들기

◆ 사건의 예



```
>>> sports = {'baseball':9, 'soccer':11, 'basketball':5}
>>> print(sports)
{'baseball': 9, 'soccer': 11, 'basketball': 5}
```

```
>>> info = {'name':'Alice', 'age':12, 'grade':5, 'phone':'010-000-0000'}
>>> print(info['age'])           # 키 'age'에 해당하는 값을 출력합니다.
12
```

```
>>> print(info['height']) # 'height'는 사전 info에 없는 키입니다. keyError가 발생합니다.
```

Traceback (most recent call last):

```
File "<pyshell#13>", line 1, in <module>
    print(info['height'])
```

KeyError: 'height'

1. 사전 만들기



사전에서는 키가 인덱스 역할을 합니다.
하지만 사전은 순서 개념이 없기 때문에 시퀀스 자료형이 아니예요.
사전을 다음과 같이 생각해도 좋을 것 같아요.

	'name'	'age'	'grade'	'phone'
info 사전	alice	12	5	'010-000-0000'

1. 사전 만들기

◆ 빈 사전 만들기

- {}는 빈 사전임. (공집합은 set() 으로 표현해야 함)

공집합 기호 이용하기	dict() 함수 이용하기
<pre>>>> D = {} >>> type(D) <class 'dict'></pre>	<pre>>>> D = dict() >>> print(D) { >>> type(D) <class 'dict'></pre>

1. 사전 만들기

◆ 사전의 ‘키’를 만들 때 주의할 점

- ‘키’는 유일해야 함.

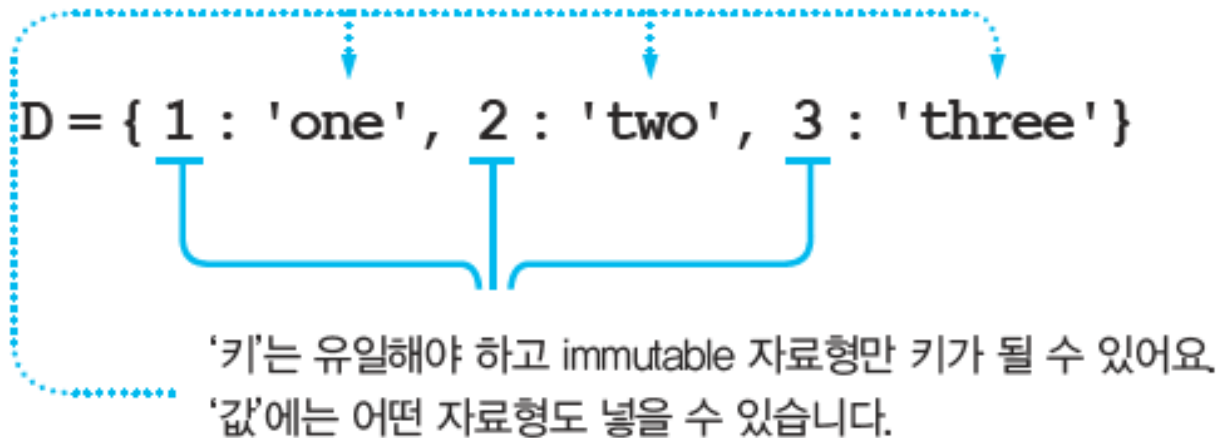
```
>>> number_of_students = {1:25, 2:30, 3:27, 4:29, 2:23}
>>> print(number_of_students)
{1: 25, 2: 23, 3: 27, 4: 29}
```

- immutable 자료형만 ‘키’로 사용 수 있음. mutable 객체를 ‘키’로 사용하면, TypeError 발생함.

```
>>> D = {[1,3,5,7]:'홀수', [2,4,6,8]:'짝수'}
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    D = {[1,3,5,7]:'홀수', [2,4,6,8]:'짝수'}
TypeError: unhashable type: 'list'
```


1. 사전 만들기

- ◆ 아홉 가지 자료형은 모두 사전의 '값'이 될 수 있습니다



1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

- 사전은 '키:값'의 쌍으로 이루어지기 때문에 변환이 간단하지 않음.
- 사전이 아닌 객체를 사전으로 변환하는 경우는 3가지로 정리할 수 있음.

① 크기가 2인 튜플 또는 리스트로 구성된 리스트는 사전으로 변환 가능함.

튜플과 리스트는 '키'와 '값'으로 사용할 데이터로 구성되어야 함.

```
>>> T = [('name', 'Alice'), ('age', 10), ('grade', 3)] # 크기 2인 튜플의 리스트
```

```
>>> D1 = dict(T)
```

```
>>> print(D1)
```

```
{'name': 'Alice', 'age': 10, 'grade': 3}
```

```
>>> area_code = [['서울', '02'], ['경기', '031'], ['인천', '032']] # 크기 2인 리스트의 리스트
```

```
>>> D2 = dict(area_code)
```

```
>>> print(D2)
```

```
{'서울': '02', '경기': '031', '인천': '032'}
```

1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

② 크기가 2인 튜플 또는 리스트로 구성된 튜플은 사전으로 변환 가능함.

```
>>> english = ((1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')) # 크기 2인 튜플의 튜플
>>> D3 = dict(english)
>>> print(D3)
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

```
>>> states = (['CA', 'California'], ['NY', 'New York']) # 크기 2인 리스트의 튜플
>>> D4 = dict(states)
>>> print(D4)
{'CA': 'California', 'NY': 'New York'}
```

1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

③ zip() 함수 이용하여 사전 생성하기

- zip() 함수는 인수로 두 개의 iterable 데이터를 받음. 첫 번째 인수는 '키'가 되고 두 번째 인수는 '값'이 됨.

```
>>> names = ['Alice', 'Bob', 'Paul', 'Cindy']
>>> ages = [10, 8, 12, 9]
>>> D = dict(zip(names, ages))  # dict 함수 안에 zip함수를 넣습니다.
>>> D
{'Alice': 10, 'Bob': 8, 'Paul': 12, 'Cindy': 9}
```

1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

③ zip() 함수 이용하여 사전 생성하기

- zip() 함수의 두 인수가 길이가 다르면 작은 크기에 맞춰서 사전이 생성됨.

```
>>> names = ['Alice', 'Bob', 'Paul'] # len(names) = 3 입니다.
>>> ages = [10, 9, 11, 13, 15] # len(ages) = 5 입니다.
>>> dict(zip(names, ages)) # 사전은 원소 3개를 갖게 됩니다.
{'Alice': 10, 'Bob': 9, 'Paul': 11}
>>> names = ['Alice', 'Bob', 'Paul', 'David', 'Carol'] # len(names) = 5 입니다.
>>> ages = [10, 9, 11] # len(ages) = 3 입니다.
>>> dict(zip(names, ages)) # 사전은 원소 3개를 갖게 됩니다.
{'Alice': 10, 'Bob': 9, 'Paul': 11}
```

1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

③ zip() 함수 이용하여 사전 생성하기

- zip() 함수 사용시에 집합이 인수가 되는 경우는 순서를 보장할 수 없음.

```
>>> S = {1, 2, 3, 4, 100, 1000} # 집합이기 때문에 어떤 순서로 '키'가 될지 알 수 없습니다.  
>>> T = ('one', 'two', 'three', 'four', 'hundred', 'thousand') # 튜플  
>>> D = dict(zip(S,T))  
>>> D  
{1: 'one', 2: 'two', 3: 'three', 100: 'four', 4: 'hundred', 1000: 'thousand'}
```

- range()함수도 zip() 함수의 인수로 사용하기 좋음.

```
>>> number = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']  
>>> D = dict(zip(range(1,11), number))  
>>> D  
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five', 6: 'six', 7: 'seven', 8: 'eight', 9: 'nine', 10: 'ten'}
```

1. 사전 만들기

◆ 다른 자료형의 데이터를 사전으로 변환하기

③ zip() 함수 이용하여 사전 생성하기

- 다음의 경우는 리스트때문에 에러가 발생함.

```
>>> L = [[1,3,5],[2,4,6]]
>>> dict(zip(L, ('odd', 'even')))          # 리스트는 '키'가 될 수 없음
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    dict(zip(L, ('odd', 'even')))
TypeError: unhashable type: 'list'
```

```
>>> L = [[1,3,5],[2,4,6]]
>>> dict(zip(L, ('odd', 'even')))
>>> dict(zip(('odd', 'even'), L))          # 튜플은 '키'가 될 수 있음
{'odd': [1, 3, 5], 'even': [2, 4, 6]}
```

1. 사전 만들기

◆ 두 사전이 같은지 비교하기

- 집합과 마찬가지로 원소의 개수가 같고, 같은 원소로 구성되어 있으면 같다고 판단함.
- 같은 키와 같은 값의 쌍이 저장되어 있으면 하나로 간주함.

```
>>> D = {1:'one', 100:'hundred', 3:'three', 10:'ten'}
>>> E = {3:'three', 10:'ten', 1:'one', 100:'hundred'}
>>> F = {1:'one', 1:'one', 100:'hundred', 3:'three', 10:'ten'} # 1:'one' 하나로 간주
>>> D == E, D == F, E == F
(True, True, True)
```


2. 사전은 '키'를 인덱스로 하고 슬라이스는 할 수 없습니다

```
>>> area_code = {'서울':'02', '경기':'031', '인천':'032'}
```

```
>>> print(area_code['경기']) # 사전 area_code에서 '키'가 '경기'인 값을 가져옵니다.
```

```
031
```

```
>>> print(area_code['서울']) # 사전 area_code에서 '키'가 '서울'인 값을 가져옵니다.
```

```
02
```

```
>>> print(area_code['대전']) # 없는 키를 넣으면 KeyError가 발생합니다.
```

```
Traceback (most recent call last):
```

```
File "<pyshell#84>", line 1, in <module>
```

```
    print(area_code['대전'])
```

```
KeyError: '대전'
```

3. 사전은 mutable 객체입니다

- ◆ 사전에 새로운 '키:값' 쌍의 원소를 추가할 수도 있고, 있던 데이터를 삭제할 수도 있음.
- ◆ 기존에 저장된 정보의 수정도 가능함. 이 경우는 '값'만 수정이 가능함.

3. 사전은 mutable 객체입니다

- ◆ 사전에 새로운 데이터를 추가하거나 기존의 데이터 수정하기

사전명[새로운 키] = 값

```
>>> books = {'파이썬':10, '아두이노':5, '자바':2, 'C언어':7}
```

```
>>> print(books)
```

```
{'파이썬': 10, '아두이노': 5, '자바': 2, 'C언어': 7}
```

```
>>> books['엔트리'] = 3          # 새로운 책 '엔트리'를 3권 추가합니다.
```

```
>>> print(books)
```

```
{'파이썬': 10, '아두이노': 5, '자바': 2, 'C언어': 7, '엔트리': 3}
```

```
>>> books['자바'] = 4           # 기존에 있는 '키'는 '값'을 업데이트합니다.
```

```
>>> print(books)
```

```
{'파이썬': 10, '아두이노': 5, '자바': 4, 'C언어': 7, '엔트리': 3}
```

3. 사전은 mutable 객체입니다

◆ 사전에서 원소 삭제하기 `del 사전명[키]`

```
>>> del books['아두이노']      # '키'가 '아두이노'인 원소를 삭제합니다.  
>>> print(books)  
{'파이썬': 10, '자바': 4, 'C언어': 7, '엔트리': 3}
```

◆ 사전을 통째로 삭제하기 `del 사전명`

```
>>> del books      # 사전을 통째로 삭제합니다.  
>>> print(books)  
... ..  
NameError: name 'books' is not defined
```

4. 사전에 in, not in 연산자 사용하기

◆ 사전에 +, *는 사용할 수 없음

◆ in / not in

```
>>> D1 = {1:'one', 5:'five', 10:'ten'}  
>>> 10 in D1      # 10은 D1에 있는 키입니다.  
True  
>>> 'ten' in D1  
False
```

4. 사전에 in, not in 연산자 사용하기

CODE 64 사전 ages는 이름을 '키'로, 나이를 '값'으로 한다. ages 사전을 만든 후 이름을 입력받아서 사전에 있는 이름인지 판단하는 코드.

```
ages = {} # 다음과 같이 사전 ages에 일일이 데이터를 저장합니다.
ages['Alice'] = 23; ages['Paul'] = 25; ages['Peter'] = 19; ages['Karen'] = 40;
ages['Andy'] = 25; ages['Cindy'] = 30; ages['David'] = 19; ages['Sally'] = 28;
ages['Tom'] = 22; ages['Sue'] = 32; ages['Bob'] = 31

name = input('Enter name : ') # 이름을 입력받습니다.
if name not in ages: # 사전에 이름이 있는지 간단히 in, not in으로 판단합니다.
    print(name, "is not in the 'ages' dictionary.")
else:
    print(name, 'is', ages[name], 'years old.')
```

[결과 1]

```
Enter name : Karen
Karen is 40 years old.
```

[결과 2]

```
Enter name : Jenny
Jenny is not in the 'ages' dictionary.
```

5. 사전에 함수 적용하기

'키'의 자료형	함수 적용 예 ('키'가 기준임)
정수(int)	<pre> >>> english = {3:'three', 5:'five', 1:'one', 4:'four', 2:'two'} >>> len(english) # 원소의 개수를 반환합니다. 5 >>> max(english) # '키'들 중에서 가장 큰 값을 반환합니다. 5 >>> min(english) # '키'들 중에서 가장 작은 값을 반환합니다. 1 >>> sum(english) # '키'들의 합을 반환합니다. 15 >>> sorted(english) # '키'들을 정렬한 리스트를 반환합니다. [1, 2, 3, 4, 5] </pre>
문자열(str)	<pre> >>> capital = {'Korea':'Seoul', 'Japan':'Tokyo', 'France':'Paris'} >>> len(capital) 3 >>> max(capital) # 아스키코드 값으로 비교합니다. 'Korea' >>> min(capital) 'France' >>> sum(capital) # 문자열이 키일 때, sum() 함수는 사용할 수 없어요. TypeError: unsupported operand type(s) for +: 'int' and 'str' >>> sorted(capital) ['France', 'Japan', 'Korea'] </pre>

6. 사전 메소드

◆ 사전에 적용할 수 있는 메소드는 모두 11개임.

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattr__', '__getitem__', '__gt__', '__hash__', '__
init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__',
 '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys',
 'pop', 'popitem', 'setdefault', 'update', 'values']
```


6. 사전 메소드

메소드	설명	반환값	발생 가능 예외
<code>clear()</code>	사전을 비웁니다(빈 사전을 만듦).	None	없음
<code>copy()</code>	사전을 복사하여 새로운 사전을 반환합니다.	있음	없음
<code>fromkeys(l)</code>	iterable 자료형 l에 있는 데이터들을 키로 사용한 새로운 사전을 만들어서 반환합니다.	있음	없음
<code>get(x)</code>	키 x에 해당하는 값을 반환합니다.	있음/ None	없음
<code>items()</code>	사전에 있는 데이터들을 (키, 값) 튜플의 리스트로 가져올 수 있습니다.	있음	없음
<code>keys()</code>	사전에 있는 키들을 리스트로 가져올 수 있습니다.	있음	없음
<code>pop(x)</code>	키가 x인 원소의 값을 반환합니다. 없는 키를 넣으면 <code>KeyError</code> 가 발생합니다.	있음	<code>KeyError</code>
<code>popitem()</code>	임의의 (키, 값)의 쌍을 반환합니다. 사전이 빈 경우는 <code>KeyError</code> 가 발생합니다.	있음	<code>KeyError</code>
<code>setdefault(x)</code>	x가 사전에 있는 키라면, 해당값을 반환합니다. x가 사전에 없는 키라면, 사전에 새로운 키로 추가합니다.	있음	없음
<code>update()</code>	두 사전을 합하여 줍니다.	None	없음
<code>values()</code>	사전에 있는 키들을 리스트로 가져올 수 있습니다.	있음	없음

6. 사전 메소드

◆ 사전 비우기 - clear() 메소드

```
>>> color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
>>> print(color_pencil)
{'red': 8, 'blue': 5, 'green': 6, 'purple': 4}
>>> a = color_pencil.clear()
>>> print(a)      # clear() 메소드의 반환값은 없어요 (따라서 None이 출력됩니다).
None
>>> print(color_pencil)
{}
```

6. 사전 메소드

◆ 사전 복사하기 - copy() 메소드

- 리스트와 마찬가지로 '='으로는 복사본이 생기지 않음.

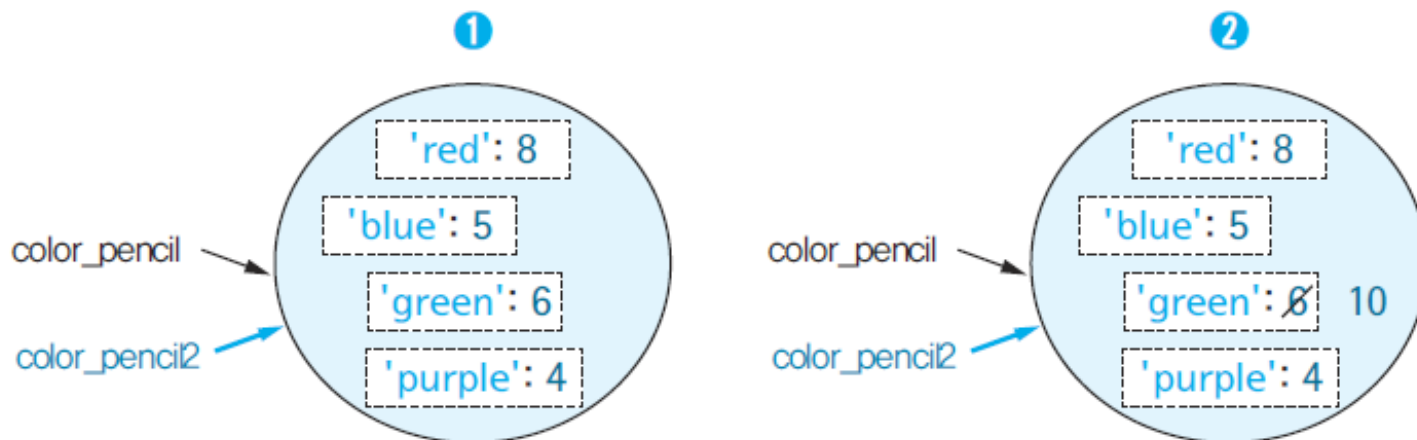
```
>>> color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
```

① >>> color_pencil2 = color_pencil # 아래 그림 참조(같은 객체를 가리키게 됩니다).

② >>> color_pencil2['green'] = 10 # 아래 그림 참조

```
>>> print(color_pencil)      # color_pencil을 출력합니다.
```

```
{'red': 8, 'blue': 5, 'green': 10, 'purple': 4}
```



6. 사전 메소드

◆ 사전 복사하기 - copy() 메소드

- copy() 메소드를 이용하면 얇은 복사본이 생성됨.

```
>>> color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
```

```
① >>> color_pencil2 = color_pencil.copy()
```

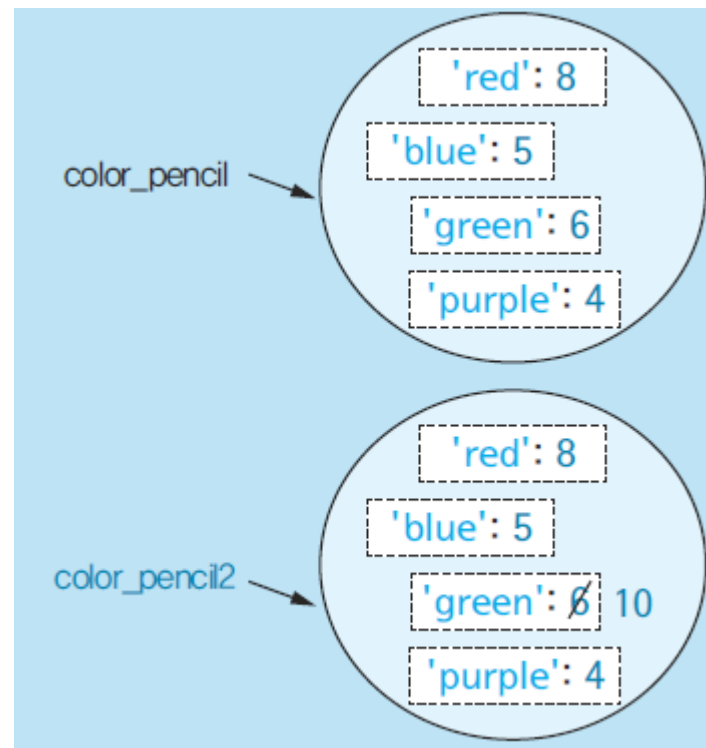
```
② >>> color_pencil2['green'] = 10
```

```
>>> print(color_pencil)
```

```
{'red': 8, 'blue': 5, 'green': 6, 'purple': 4}
```

```
>>> print(color_pencil2)
```

```
{'red': 8, 'blue': 5, 'green': 10, 'purple': 4}
```



NOTE mutable 자료형인 리스트, 집합, 사전만 `copy()` 메소드를 갖습니다.

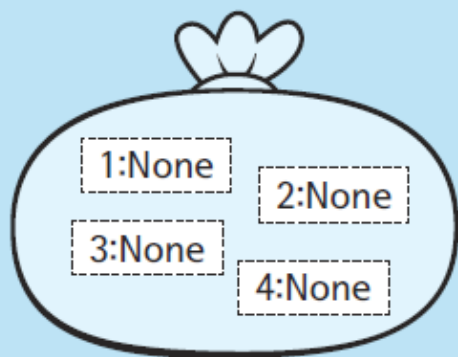
6. 사전 메소드

◆ iterable 객체를 '키'로 하는 사전 만들기 - fromkeys() 메소드

인수	1개	<code>dict.fromkeys(iterable)</code>
	2개	<code>dict.fromkeys(iterable, value)</code>
반환값	<p>인수가 한 개인 경우에는 iterable 데이터가 '키'가 되고, '값'은 모두 None으로 초기화된 사전이 반환됩니다.</p> <p>인수가 두 개인 경우에는 iterable 데이터가 '키'가 되고, '값'은 모두 두 번째 인수 value로 초기화된 사전이 반환됩니다.</p>	

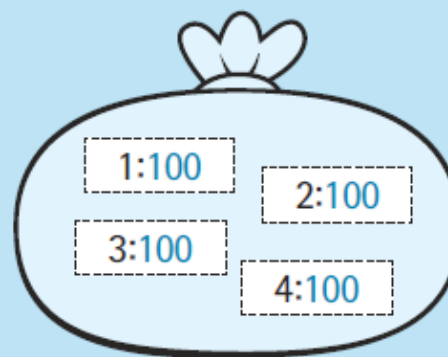
`D = dict.fromkey(iterable 객체)`

`D = dict.fromkey([1, 2, 3, 4])`



`D = dict.fromkey(iterable 객체, V)`

`D = dict.fromkey([1, 2, 3, 4], 100)`



6. 사전 메소드

첫 번째 인수 iterable 자료형	예제 코드
문자열	<pre> >>> string = 'wxyz' >>> D1 = dict.fromkeys(string) >>> print(D1) {'w': None, 'x': None, 'y': None, 'z': None} >>> string2 = 'hello' >>> D2 = dict.fromkeys(string2) # 'l'은 한 번만 들어갑니다. >>> print(D2) {'h': None, 'e': None, 'l': None, 'o': None} >>> D3 = dict.fromkeys(string2, 5) # '값'을 5로 초기화합니다. >>> print(D3) {'h': 5, 'e': 5, 'l': 5, 'o': 5} </pre>
리스트	<pre> >>> lang = ['python', 'java', 'html5'] >>> D4 = dict.fromkeys(lang) >>> print(D4) {'python': None, 'java': None, 'html5': None} >>> score = [80, 90, 77, 95] >>> D5 = dict.fromkeys(score) >>> print(D5) {80: None, 90: None, 77: None, 95: None} >>> D6 = dict.fromkeys(score, 0) >>> print(D6) {80: 0, 90: 0, 77: 0, 95: 0} </pre>

6. 사전 메소드

첫 번째 인수 iterable 자료형	예제 코드
튜플	<pre>>>> temp = (23.5, 17.2, 20.7) >>> D7 = dict.fromkeys(temp) >>> print(D7) {23.5: None, 17.2: None, 20.7: None} >>> D8 = dict.fromkeys(temp, 1) >>> print(D8) {23.5: 1, 17.2: 1, 20.7: 1}</pre>
집합	<pre>>>> fruits = {'apple', 'melon', 'grape'} >>> D9 = dict.fromkeys(fruits) >>> print(D9) {'apple': None, 'grape': None, 'melon': None} >>> D10 = dict.fromkeys(fruits, 3) >>> print(D10) {'apple': 3, 'grape': 3, 'melon': 3}</pre>

6. 사전 메소드

첫 번째 인수 iterable 자료형	예제 코드
사전	<pre>>>> english = {100:'hundred', 10:'ten', 1:'one', 5:'five'} >>> D11 = dict.fromkeys(english) # '키'만 가져옵니다. >>> print(D11) {100: None, 10: None, 1: None, 5: None} >>> D12 = dict.fromkeys(english, 7) >>> print(D12) {100: 7, 10: 7, 1: 7, 5: 7}</pre>
range() 함수	<pre>>>> D13 = dict.fromkeys(range(5)) >>> print(D13) {0: None, 1: None, 2: None, 3: None, 4: None} >>> D14 = dict.fromkeys(range(2,10,3), 10) >>> print(D14) {2: 10, 5: 10, 8: 10}</pre>

6. 사전 메소드

◆ 사전에서 어떤 '키'의 '값' 가져오기 - get() 메소드

인수	1개	사전.get(키)
	2개	사전.get(키, 키가 없을 경우에 반환하고자 하는 값)
반환값	인수로 들어오는 '키'가 사전에 있는 '키'라면, 해당하는 '값'을 반환합니다. 만약에 인수가 한 개이고 '키'가 없을 경우에는 None을 반환합니다. 만약에 인수가 두 개이고 '키'가 없을 경우에는 두 번째 인수가 반환됩니다.	

사전.get(키)



사전에 있는 키라면, 해당값을 반환
 사전에 없는 키라면, None 반환

사전.get(키, v)



사전에 있는 키라면, 해당값을 반환
 사전에 없는 키라면, v 반환

6. 사전 메소드

◆ 사전에서 어떤 '키'의 '값' 가져오기 - get() 메소드

```
>>> D = {1:'one', 2:'two', 4:'four'}    # 숫자가 키, 영어가 값입니다.
>>> v = D.get(2)                        # 2는 사전 D에 있는 키입니다. 해당 값을 반환합니다.
>>> print(v)
two
>>> D                                    # 사전은 그대로입니다.
{1: 'one', 2: 'two', 4: 'four'}
>>> v = D.get(5)                        # 5는 사전 D에 없는 키입니다. None을 반환합니다.
>>> print(v)
None
>>> v = D.get(5, 'five')               # 5가 D에 없다면, 'five'를 반환합니다.
>>> print(v)
five
>>> D                                    # 사전은 그대로입니다.
{1: 'one', 2: 'two', 4: 'four'}
```

6. 사전 메소드

◆ 사전에서 어떤 '키'의 '값' 가져오기 - get() 메소드

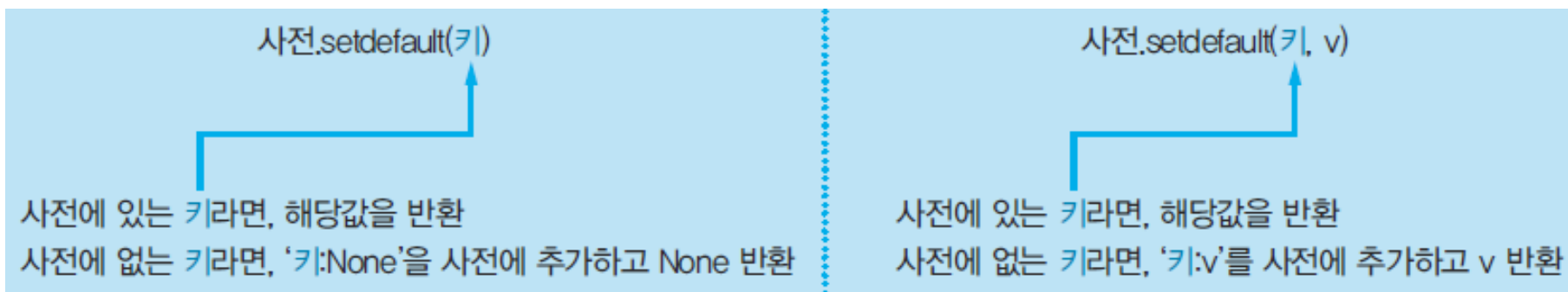
참고 get() 메소드는 사전[키]로 사용하는 것과 같습니다. 그런데, 없는 '키'가 입력되면 get() 메소드는 None을 반환하는 반면에 사전[키]는 KeyError를 발생시킵니다.

```
>>> D = {'name':'Alice', 'age':10, 'height':130}
>>> D.get('grade')    # 'grade'는 없는 '키'이지만 에러가 없습니다.
>>> D['grade']        # [ ] 기호를 사용할 때는 반드시 사전에 있는 '키'여야 합니다.
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    D['grade']
KeyError: 'grade'
```

6. 사전 메소드

- ◆ 사전에서 값을 가져오거나 새로운 원소를 추가하는 메소드 - `setdefault()` 메소드

인수	1개	사전.setdefault(키)
	2개	사전.setdefault(키, 초기값)
반환값	키가 사전에 있다면 해당하는 값을 반환합니다. 키가 사전에 없다면 사전에 키를 추가해 줍니다. 이때, 인수가 1개이면 None이 값으로 들어가고, 초기값이 있다면 그 초기값이 값으로 들어갑니다.	



6. 사전 메소드

- ◆ 사전에서 값을 가져오거나 새로운 원소를 추가하는 메소드 - `setdefault()` 메소드

```
>>> players = {'baseball':9, 'basketball':5}
>>> n = players.setdefault('basketball')    # 'basketball'은 있는 키입니다.
>>> print(n)                                # 키가 있다면, 해당 값을 반환합니다.
5
>>> players                                # 사전은 그대로입니다.
{'baseball': 9, 'basketball': 5}
>>> n = players.setdefault('soccer')         # 'soccer'는 없는 키입니다.
>>> players                                  # 사전에 'soccer':None 이 추가됩니다.
{'baseball': 9, 'basketball': 5, 'soccer': None}
>>> print(n)                                # 없는 키는 None을 반환합니다.
None
>>> n = players.setdefault('volleyball', 6)  # 'volleyball'은 없는 키입니다.
>>> players                                  # 'volleyball'을 키로 하고 두 번째 인수 6이 값으로 추가됩니다.
{'baseball': 9, 'basketball': 5, 'soccer': None, 'volleyball': 6}
>>> print(n)                                # 6이 반환됩니다.
6
```

6. 사전 메소드

◆ 사전에서 원소를 삭제하기 - pop(), popitem() 메소드

▪ pop(x) 메소드

인수	1개	사전.pop(키)
	2개	사전.pop(키, v)
반환값	키가 사전에 있다면 해당하는 값을 반환하고 사전에서 삭제합니다. 인수가 한 개일 때, 키가 사전에 없다면 KeyError가 발생합니다. 인수가 두 개일 때, 키가 사전에 없다면 v가 반환됩니다.	

사전.pop(키)

사전에 있는 키라면, 해당값을 반환하고 사전에서 '키:값' 삭제
사전에 없는 키라면, KeyError 발생

사전.pop(키, v)

사전에 있는 키라면, 해당값을 반환하고 사전에서 '키:값' 삭제
사전에 없는 키라면, v 반환

6. 사전 메소드

◆ 사전에서 원소를 삭제하기 - pop(), popitem() 메소드

- pop(x) 메소드

```
>>> states = {'CA':'California', 'NY':'New York'}
>>> s = states.pop('NY')
>>> states                                # 사전에서 키가 'NY'인 아이템이 삭제됩니다.
{'CA': 'California'}
>>> print(s)                             # 키 'NY'의 값이 반환됩니다.
New York
>>> s = states.pop('OH') # 없는 키를 넣으면 KeyError가 발생합니다.
.....
KeyError: 'OH'
```

```
>>> states = {'CA':'California', 'NY':'New York'}
>>> s = states.pop('OH', 'Ohio') # 사전에 'OH'는 없습니다. 'Ohio'를 반환합니다.
>>> states                        # 사전은 그대로입니다.
{'CA': 'California', 'NY': 'New York'}
>>> print(s)
Ohio
```

6. 사전 메소드

◆ 사전에서 원소를 삭제하기 - pop(), popitem() 메소드

▪ popitem() 메소드

```
>>> states = {'CA':'California', 'NY':'New York', 'OH':'Ohio'}
>>> result = states.popitem()      # 임의의 아이템을 선택하여 반환합니다.
>>> states                        # 'OH':'Ohio'가 삭제되었습니다.
{'CA': 'California', 'NY': 'New York'}
>>> print(result)                 # 삭제된 아이템은 (키,값)의 튜플로 반환됩니다.
('OH', 'Ohio')
>>> k, v = states.popitem()       # 반환값이 (키,값) 튜플이므로 두 변수를 이용합니다.
>>> print(k, v)
NY New York
>>> states
{'CA': 'California'}
>>> D = {}
>>> result = D.popitem()          # 빈 사전에 popitem() 메소드는 KeyError 발생합니다.
....
KeyError: 'popitem(): dictionary is empty'
```


6. 사전 메소드

◆ 사전 합하기 - update() 메소드

```
>>> D1 = {'Korea':'Seoul', 'Japan':'Tokyo', 'France':'Paris'}
>>> D2 = {'USA':'Washington D.C.'}
>>> a = D1.update(D2)
>>> print(D1) # D1 = D1 U D2
{'Korea': 'Seoul', 'Japan': 'Tokyo', 'France': 'Paris', 'USA': 'Washington D.C.'}
>>> D2                                     # D2는 그대로입니다.
{'USA': 'Washington D.C.'}
>>> print(a)                               # 반환값은 없습니다.
None

>>> v = {'name':'Alice', 'age':10}
>>> v.update(grade=3)   # 괄호 안에 grade=3을 키워드 인수라고 부릅니다.(13장)
>>> print(v)
{'name': 'Alice', 'age': 10, 'grade': 3}
>>> v.update(address='Seoul', height='130')
>>> v
{'name': 'Alice', 'age': 10, 'grade': 3, 'address': 'Seoul', 'height': '130'}
```

6. 사전 메소드

- ◆ for 반복문에서 사용하면 유용한 메소드 - items(), keys(), values()
 - 사전에 있는 모든 원소를 (키,값)의 쌍으로 가져오기 - items()

```
>>> voca = {'컵':'cup', '커피':'coffee', '컴퓨터':'computer'}
>>> result = voca.items()
>>> print(result)
dict_items([('컵', 'cup'), ('커피', 'coffee'), ('컴퓨터', 'computer')])
>>> type(result)      # items() 메소드의 반환값은 dict_items 자료형이에요.
<class 'dict_items'>
>>> iter(result)      # dict_items는 iterable 자료형임을 알 수 있습니다.
<dict_itemiterator object at 0x021CD6F0>
```

6. 사전 메소드

- ◆ for 반복문에서 사용하면 유용한 메소드 - items(), keys(), values()
 - 사전에서 키들만 가져오기 - keys()

```
>>> score_of_students = {201812345:90, 201811111:88, 201800100:92}
>>> result = score_of_students.keys()
>>> print(result)
dict_keys([201812345, 201811111, 201800100])
>>> type(result)          # keys( ) 메소드의 반환값은 dict_keys 자료형이에요.
<class 'dict_keys'>
>>> iter(result)          # dict_keys는 iterable 자료형임을 알 수 있습니다.
<dict_keyiterator object at 0x021CD8A0>
```

6. 사전 메소드

- ◆ for 반복문에서 사용하면 유용한 메소드 - items(), keys(), values()
 - 사전에서 키들만 가져오기 - values()

```
>>> info = {'name':'Alice', 'age':10, 'grade':3, 'address':'Seoul'}
>>> result = info.values()
>>> print(result)
dict_values(['Alice', 10, 3, 'Seoul'])
>>> type(result)      # values() 메소드의 반환값의 자료형은 dict_values입니다.
<class 'dict_values'>
>>> iter(result)      # dict_values는 iterable 자료형입니다.
<dict_valueiterator object at 0x021D4840>
```

6. 사전 메소드

참고 어떤 객체가 iterable 객체인지 쉽게 알 수 있는 방법이 있어요. `iter()` 함수에 적용해서 어떤 정보가 나오면(iterator라는 정보) iterable 객체이고, 에러가 발생하면 iterable 객체가 아니에요.

```
>>> L = [1, 2, 4, 5]; T = (2, 4, 6); name = 'Alice'
>>> iter(L)          # 리스트는 iterable 객체
<list_iterator object at 0x02401190>
>>> iter(T)          # 튜플도 iterable 객체
<tuple_iterator object at 0x02401390>
>>> iter(name)        # 문자열도 iterable 객체
<str_iterator object at 0x02401370>
>>> S = {1, 3, 5}; D = {'name':'Kim', 'age':20, 'address':'Seoul'}
>>> iter(S)           # 집합도 iterable 객체
<set_iterator object at 0x023FFFA8>
>>> iter(D)           # 사전도 iterable 객체
<dict_keyiterator object at 0x02400840>
>>> iter(range(10))    # range() 함수의 결과도 iterable 객체
<range_iterator object at 0x021695C0>
>>> iter(reversed(L))  # reversed() 함수의 결과도 iterable 객체
<list_reverseiterator object at 0x024013F0>
>>> x = 10
>>> iter(x)           # 정수는 iterable 객체가 아닙니다.
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    iter(x)
TypeError: 'int' object is not iterable
```

7. 사전과 for 반복문

◆ keys() 메소드 - '키'에 대해서 for 반복문 수행하기

```
color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
for color in color_pencil.keys():
    print(color)
```

[결과]
red
blue
green
purple

참고 for 반복문에서 in 다음에 사전 이름만 넣어도 알아서 사전의 '키'만을 가져옵니다.

```
color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
for color in color_pencil:      # 사전명만 넣었음.
    print(color)
```

[결과]
red
blue
green
purple

7. 사전과 for 반복문

- ◆ values() 메소드 - '값'에 대해서 for 반복문 수행하기

```
color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4}
for count in color_pencil.values():
    print(count)
```

[결과]

8
5
6
4

7. 사전과 for 반복문

◆ items() 메소드 - '키'와 '값'에 대해서 for 반복문 수행하기

- items() 메소드는 튜플을 반환함.

<pre>color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4} for color, count in color_pencil.items(): print('There are', count, color, 'pencils.')</pre>	<p>[결과]</p> <p>There are 8 red pencils. There are 5 blue pencils. There are 6 green pencils. There are 4 purple pencils.</p>
<pre>color_pencil = {'red':8, 'blue':5, 'green':6, 'purple':4} for pencil in color_pencil.items(): print(pencil)</pre>	<p>[결과]</p> <p>('red', 8) ('blue', 5) ('purple', 4) ('green', 6)</p>

7. 사전과 for 반복문

CODE 65 하나의 단어를 입력받아서 그 단어에 모음이 각각 몇 개 있는지를 출력하는 코드.

```
word = input('Enter one word : ')
vowel = dict.fromkeys('aeiou', 0)
```

```
for c in word:
    if c in vowel.keys():
        vowel[c] += 1
```

```
for c,n in vowel.items():
    print('vowel', c, ': ', n)
```

[결과 1]

```
Enter one word : computer
vowel a : 0
vowel e : 1
vowel i : 0
vowel o : 1
vowel u : 1
```

[결과 2]

```
Enter one word : mississippi
vowel a : 0
vowel e : 0
vowel i : 4
vowel o : 0
vowel u : 0
```

7. 사전과 for 반복문

CODE 66 리스트 vocabulary에는 여러 개의 단어가 저장되어 있다. 각 단어가 몇 번 중복되어 있는지를 사전을 이용해서 출력하는 코드.

```
vocabulary = ['paper', 'rose', 'pencil', 'book', 'desk',  
              'computer', 'book', 'erase', 'computer',  
              'computer', 'rose', 'apple', 'rose', 'book',  
              'computer', 'paper', 'usb', 'chair', 'usb',  
              'paper', 'shoe', 'spoon']
```

```
voca = {}  
for v in vocabulary:  
    if v in voca:      # 이미 v가 voca에 있으면 해당 값을 1 증가  
        voca[v] += 1  
    else:              # v가 아직 voca에 없으면 새로 추가  
        voca.setdefault(v, 1)  
  
for k, v in voca.items():  
    print(k, ': ', v)
```

[결과]

```
paper : 3  
rose : 3  
pencil : 1  
book : 3  
desk : 1  
computer : 4  
erase : 1  
apple : 1  
usb : 2  
chair : 1  
shoe : 1  
spoon : 1
```

7. 사전과 for 반복문

CODE 67 사전 score에는 이름과 성적이 저장되어 있습니다. 이름은 문자열이고 성적은 [국어, 영어, 수학]의 리스트로 저장되어 있어요. 사전에서 데이터를 읽어서 학생 이름과 평균을 계산하여 출력하는 프로그램을 작성해 보세요. 사전을 연습하기 위해서 사전 score에 있는 데이터를 읽어서 average라는 이름의 사전을 다시 만들도록 해 보세요. average 사전은 이름이 키가 되고, 평균 값이 되도록 합니다. 그리고 최종 average 사전을 출력합니다

```
score = {'Alice': [80, 90, 88],  
        'Paul': [77, 92, 90],  
        'David': [60, 70, 80],  
        'Cindy': [80, 92, 95],  
        'Tom': [85, 65, 70]}
```

```
average = {}      # average 사전을 만듭니다.  
for name, scores in score.items():  
    average[name] = sum(scores) / len(scores)  
  
for name, avg in average.items():  
    print("{:7} {:10.2f}".format(name, avg))
```

[결과]

```
Paul 86.33  
David 70.00  
Cindy 89.00  
Tom 73.33
```

7. 사전과 for 반복문

CODE 68 하나의 단어를 입력받아서 암호화된 단어로 바꾸는 문제입니다. 사전 crypt_code를 보고 키에 해당하는 알파벳을 값에 해당하는 알파벳으로 바꾸는 코드를 작성해 보세요.

```
crypt_code = {'a':'g', 'b':'r', 'c':'q', 'd':'i', 'e':'u',
              'f':'e', 'g':'w', 'h':'n', 'i':'d', 'j':'l',
              'k':'v', 'l':'t', 'm':'f', 'n':'s', 'o':'o',
              'p':'a', 'q':'k', 'r':'x', 's':'m', 't':'p',
              'u':'y', 'v':'b', 'w':'j', 'x':'z', 'y':'c',
              'z':'h'}
```

```
original_msg = input('Enter word : ')
crypted_msg = ''      # 빈 문자열을 만들어 둡니다.
for ch in original_msg:
    crypted_msg += crypt_code[ch]
print(original_msg, '->', crypted_msg)
```

[결과 1]

Enter word : python
python -> acpnos

[결과 2]

Enter word : programming
programming -> axowxgffds

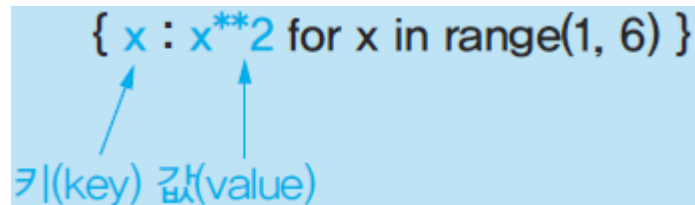
[결과 3]

Enter word : computer
computer -> qofaypux

8. 사전 내에서 for 반복문 사용하기 (Dictionary Comprehension)

- ◆ 리스트, 집합처럼 사전 내에서도 for 반복문을 사용할 수 있음

```
>>> squares = { x : x**2 for x in range(1, 6) }  
>>> squares  
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```



The diagram shows the syntax `{ x : x**2 for x in range(1, 6) }` on a light blue background. Two blue arrows point from the text below to parts of the code: one from '키(key)' to the variable 'x' and another from '값(value)' to the expression 'x**2'.

- 이차 함수 $y = x^2 + 1$ 의 x 와 y 값을 사전으로 저장하는 예제

(정의역 $-3 \leq x \leq 3$)

```
>>> f = { x : x**2 + 1 for x in range(-3, 4) }  
>>> print(f)  
{-3: 10, -2: 5, -1: 2, 0: 1, 1: 2, 2: 5, 3: 10}
```

8. 사전 내에서 for 반복문 사용하기 (Dictionary Comprehension)

- ◆ 리스트, 집합처럼 사전 내에서도 for 반복문을 사용할 수 있음
 - 리스트에는 도시명과 기온이 번갈아 저장되어 있다. 이를 사전으로 변환하는 dictionary comprehension 코드.

```
>>> T = ['서울', 15.2, '인천', 16.0, '대전', 17.3, '부산', 19]  
>>> temperature = { T[i] : T[i+1] for i in range(0, len(T), 2) }  
>>> print(temperature)  
{ '서울': 15.2, '인천': 16.0, '대전': 17.3, '부산': 19 }
```

```
T = ['서울', 15.2, '인천', 16.0, '대전', 17.3, '부산', 19]  
temperature = { T[i] : T[i+1] for i in range(0, len(T), 2) }
```

키(key) 값(value)

9. 사전 출력하기 (pprint 모듈)

- ◆ pprint 모듈을 이용하면 사전을 보기 좋게 출력할 수 있음

```
>>> T = {'서울':15.2, '인천':16.0, '대전':17.3, '부산':19.2}
>>> import pprint
>>> pprint.pprint(T)
{'대전': 17.3, '부산': 19.2, '서울': 15.2, '인천': 16.0}
>>> pprint.pprint(T, width=20)  # 'width=양의정수'를 넣으면 다음과 같이 출력됩니다.
{'대전': 17.3,
'부산': 19.2,
'서울': 15.2,
'인천': 16.0}
>>> pprint.pprint(T, width=20, indent=10)  # indent 조정할 수 있습니다.
{'대전': 17.3,
'부산': 19.2,
'서울': 15.2,
'인천': 16.0}
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 9가지 자료형 모두 사전의 값이 될 수 있음
- ◆ 사전의 값으로 mutable 자료형이 저장될 때 복사(copy)에 유의해야 함
- ◆ mutable 객체는 객체 자체를 복사하지 않고, 객체 참조 정보를 복사하기 때문에 유의해야 함

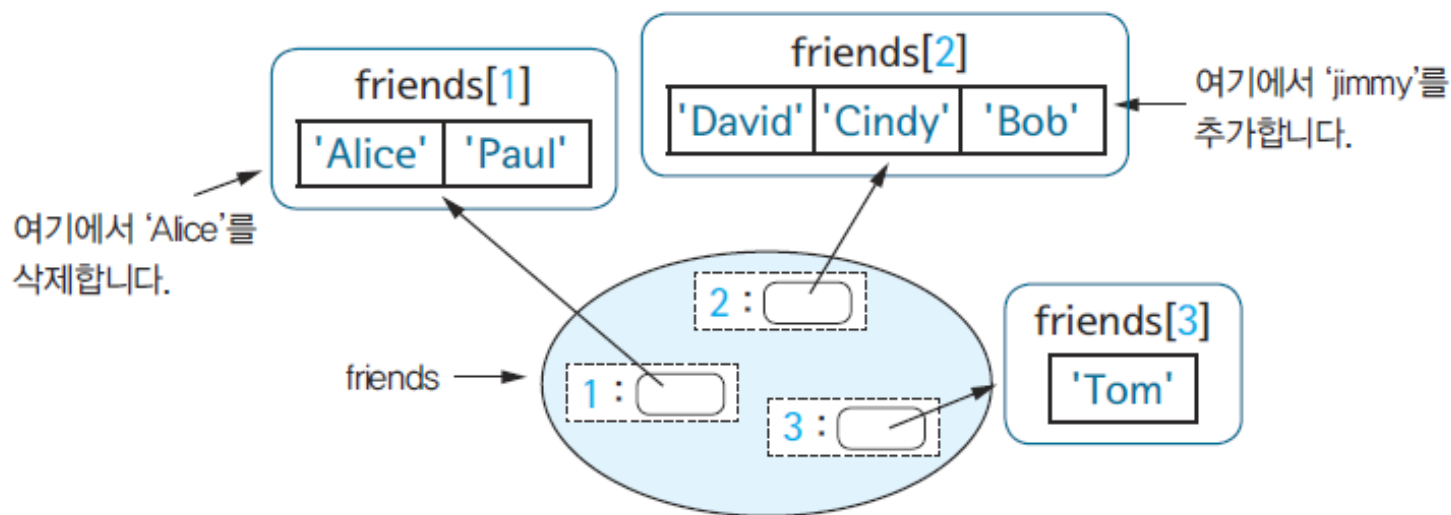
10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제

친구 목록은 변할 수 있어서 리스트로 만들었습니다.

```
>>> friends = {1:['Alice', 'Paul'], 2:['David', 'Cindy', 'Bob'], 3:['Tom']}
```

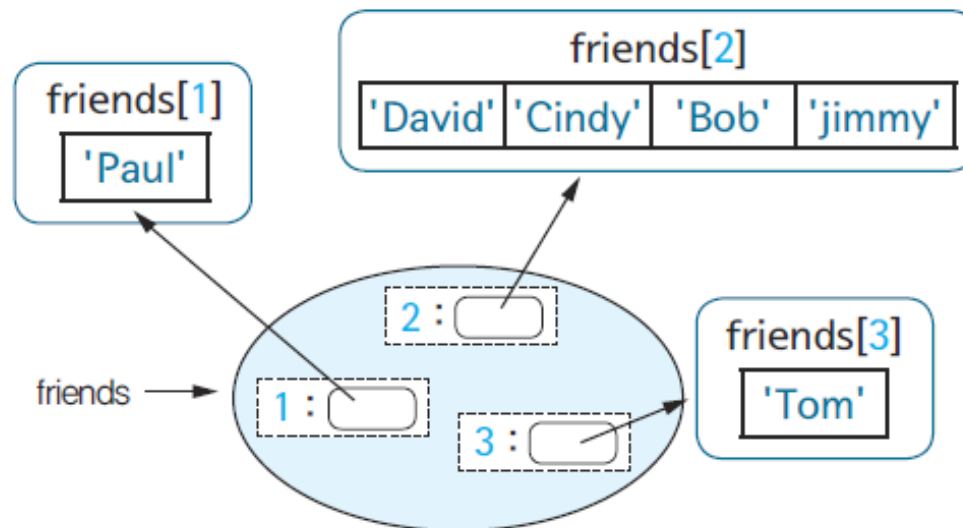
- 위의 사전에서 2번에 친구 'Jimmy'를 추가하고, 1번에 있는 친구 'Alice'를 삭제하려고 함.



10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제

```
>>> friends[2].append('Jimmy')    # 2반에 'Jimmy' 추가
>>> friends
{1: ['Alice', 'Paul'], 2: ['David', 'Cindy', 'Bob', 'Jimmy'], 3: ['Tom']}
>>> friends[1].remove('Alice')    # 1반에서 'Alice' 삭제
>>> friends
{1: ['Paul'], 2: ['David', 'Cindy', 'Bob', 'Jimmy'], 3: ['Tom']}
```



10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제

- 얕은 복사와 깊은 복사

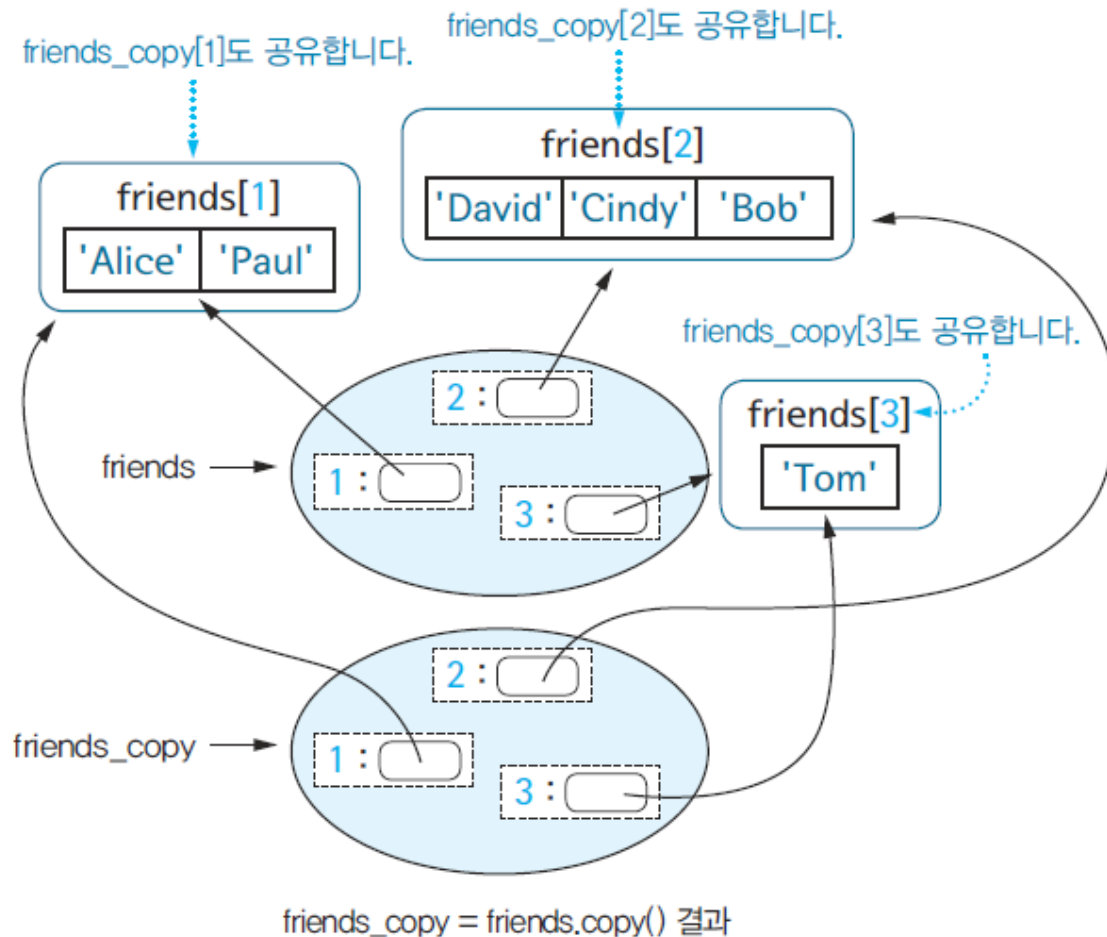
얕은 복사	사전의 copy() 메소드, copy 모듈의 copy() 함수
깊은 복사	copy 모듈의 deepcopy() 함수

- 사전의 copy() 메소드

```
>>> friends = {1:['Alice', 'Paul'], 2:['David', 'Cindy', 'Bob'], 3:['Tom']}
>>> friends_copy = friends.copy()
>>> friends_copy[2].append('Jimmy')    # friends_copy 사전 2반에 'Jimmy' 추가
>>> friends_copy
{1: ['Alice', 'Paul'], 2: ['David', 'Cindy', 'Bob', 'Jimmy'], 3: ['Tom']}
>>> friends    # friends_copy를 수정했는데, friends 사전도 수정되었습니다.
{1: ['Alice', 'Paul'], 2: ['David', 'Cindy', 'Bob', 'Jimmy'], 3: ['Tom']}
```

10. 사전의 값으로 mutable 자료가 저장되는 경우

- ◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제
 - 사전의 `copy()` 메소드



10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제
 - copy 모듈의 `copy()` 함수를 이용한 얇은 복사

```
>>> friends = {1:['Alice', 'Paul'], 2:['David', 'Cindy', 'Bob'], 3:['Tom']}
>>> import copy
>>> F = copy.copy(friends)
>>> F[2].append('Carol')
>>> F[3].remove('Tom')
>>> F
{1: ['Alice', 'Paul'], 2: ['David', 'Cindy', 'Bob', 'Carol'], 3: [] }
>>> friends
{1: ['Alice', 'Paul'], 2: ['David', 'Cindy', 'Bob', 'Carol'], 3: [] }
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제

- copy 모듈의 **deepcopy()** 함수를 이용한 깊은 복사

```
>>> books = {'children': ['Peter Pan', 'Snow White'],
             'computer': ['python', 'java', 'html5'],
             'travel': ['Asia', 'Europe', 'Africa']}

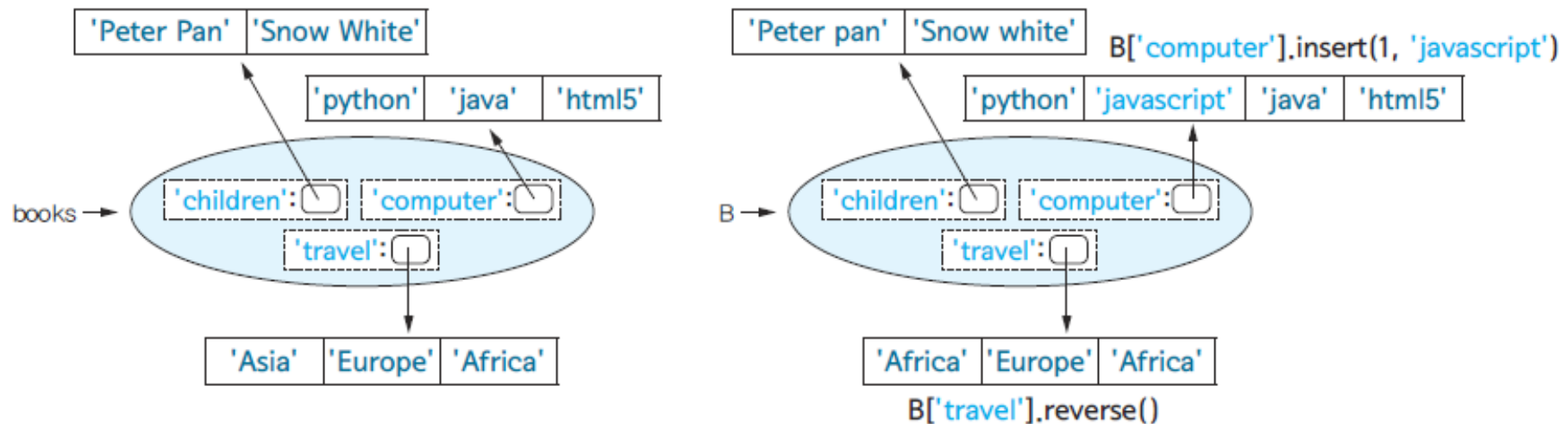
>>> import copy
>>> B = copy.deepcopy(books)                # 독립된 복사본이 생김.
>>> B['computer'].insert(1, 'javascript')    # B['computer']에만 'javascript' 추가.
>>> B['travel'].reverse()                   # B['travel']만 역순으로 바뀜.
>>> import pprint
>>> pprint.pprint(books, width=100, indent=5)
{  'children': ['Peter Pan', 'Snow White'],
   'computer': ['python', 'java', 'html5'],
   'travel':  ['Asia', 'Europe', 'Africa']}

>>> pprint.pprint(B, width=100, indent=5)
{  'children': ['Peter Pan', 'Snow White'],
   'computer': ['python', 'javascript', 'java', 'html5'],
   'travel':  ['Africa', 'Europe', 'Asia']}
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 리스트가 저장되는 경우의 복사 예제
 - copy 모듈의 **deepcopy()** 함수를 이용한 깊은 복사

B = copy.deepcopy(books) 결과 독립된 사전 생성됨.



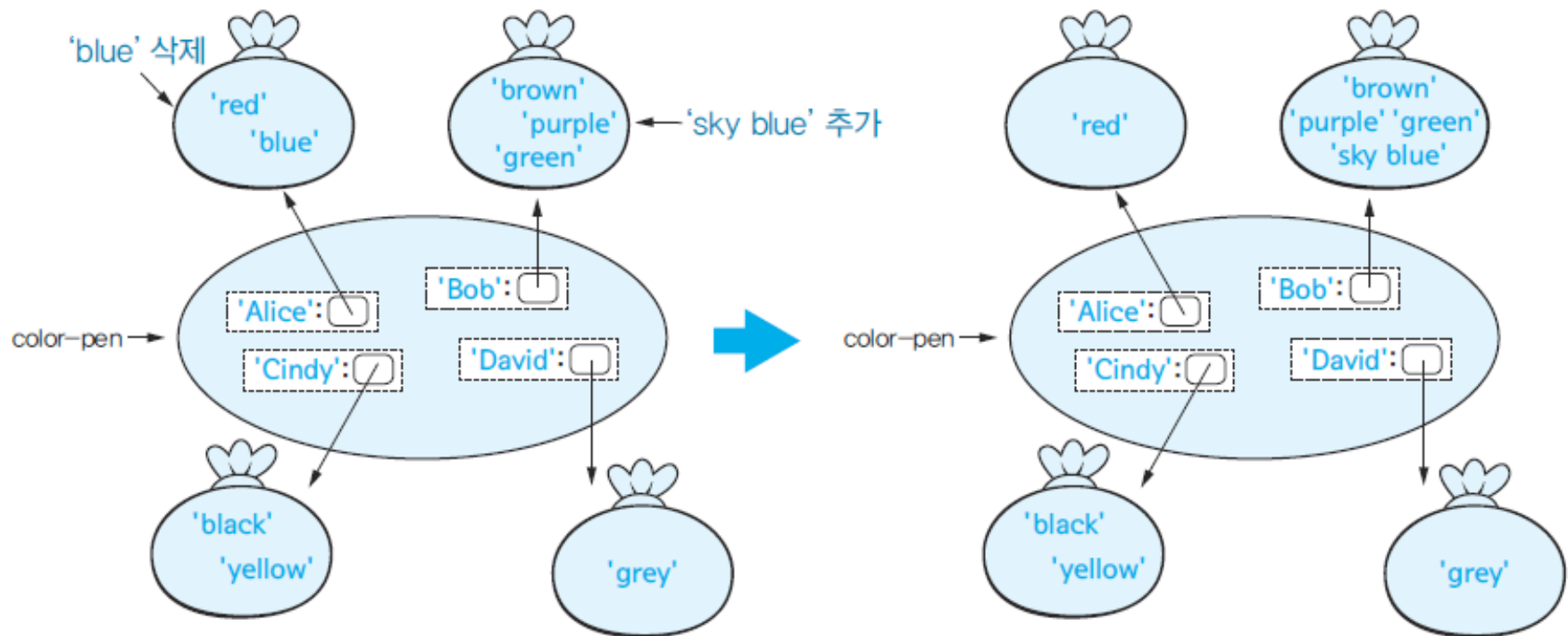
10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 집합이 저장되는 경우의 복사 예제

```
>>> color_pen = {'Alice': {'red', 'blue'},
                  'Bob': {'brown', 'purple', 'green'},
                  'Cindy': {'black', 'yellow'},
                  'David': {'grey'}}
>>> color_pen['Bob'].add('sky blue')    # 집합 메소드 add()를 사용합니다.
>>> color_pen['Alice'].discard('blue')  # 집합 메소드 discard()를 사용합니다.
>>> import pprint
>>> pprint.pprint(color_pen, width=100, indent=4)
{ 'Alice': {'red'},
  'Bob': {'brown', 'sky blue', 'purple', 'green'},
  'Cindy': {'black', 'yellow'},
  'David': {'grey'}}
```


10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 집합이 저장되는 경우의 복사 예제



10. 사전의 값으로 mutable 자료형이 저장되는 경우

◆ 사전의 값으로 집합이 저장되는 경우의 복사 예제

- 집합의 `copy()` 메소드, `copy` 모듈의 `copy()` 함수를 이용한 얇은 복사

```
>>> fruits = {'Alice':{'apple', 'kiwi'},
              'Bob':{'melon', 'orange', 'blueberry'},
              'Carol':{'grape', 'strawberry'}}
```

사전의 `copy()` 메소드

```
>>> F = fruits.copy()
>>> F['Bob'].remove('blueberry')
>>> import pprint
>>> pprint.pprint(F)
{'Alice': {'kiwi', 'apple'},
 'Bob': {'melon', 'orange'},
 'Carol': {'strawberry', 'grape'}}
>>> pprint.pprint(fruits)
{'Alice': {'kiwi', 'apple'},
 'Bob': {'melon', 'orange'},
 'Carol': {'strawberry', 'grape'}}
```

`copy` 모듈의 `copy()` 함수

```
>>> import copy
>>> F = copy.copy(fruits)
>>> F['Bob'].remove('blueberry')
>>> import pprint
>>> pprint.pprint(F)
{'Alice': {'apple', 'kiwi'},
 'Bob': {'melon', 'orange'},
 'Carol': {'strawberry', 'grape'}}
>>> pprint.pprint(fruits)
{'Alice': {'apple', 'kiwi'},
 'Bob': {'melon', 'orange'},
 'Carol': {'strawberry', 'grape'}}
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 집합이 저장되는 경우의 복사 예제
 - copy 모듈의 deepcopy() 함수를 이용한 깊은 복사

```
>>> fruits = {'Alice':{'apple', 'kiwi'},
              'Bob':{'melon', 'orange', 'blueberry'},
              'Carol':{'grape', 'strawberry'}}
>>> import copy
>>> F = copy.deepcopy(fruits)
>>> F['Bob'].remove('blueberry')
>>> import pprint
>>> pprint.pprint(F)
{'Alice': {'kiwi', 'apple'},
 'Bob': {'melon', 'orange'},
 'Carol': {'strawberry', 'grape'}}
>>> pprint.pprint(fruits)
{'Alice': {'kiwi', 'apple'},
 'Bob': {'melon', 'blueberry', 'orange'},
 'Carol': {'strawberry', 'grape'}}
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 사전이 저장되는 경우의 복사 예제
 - 사전의 copy 메소드

```
>>> D = {1:{15:['Alice', 'Paul'], 20:['David']},  
        2:{17:['Cindy'], 25:['Helen', 'Tom', 'Jenny']}}  
>>> E = D.copy()  
>>> E[2][17].append('Jessica')  
>>> D  
{1: {15: ['Alice', 'Paul'], 20: ['David']}, 2: {17: ['Cindy', 'Jessica'], 25: ['Helen',  
'Tom', 'Jenny']}}  
>>> E  
{1: {15: ['Alice', 'Paul'], 20: ['David']}, 2: {17: ['Cindy', 'Jessica'], 25: ['Helen',  
'Tom', 'Jenny']}}
```

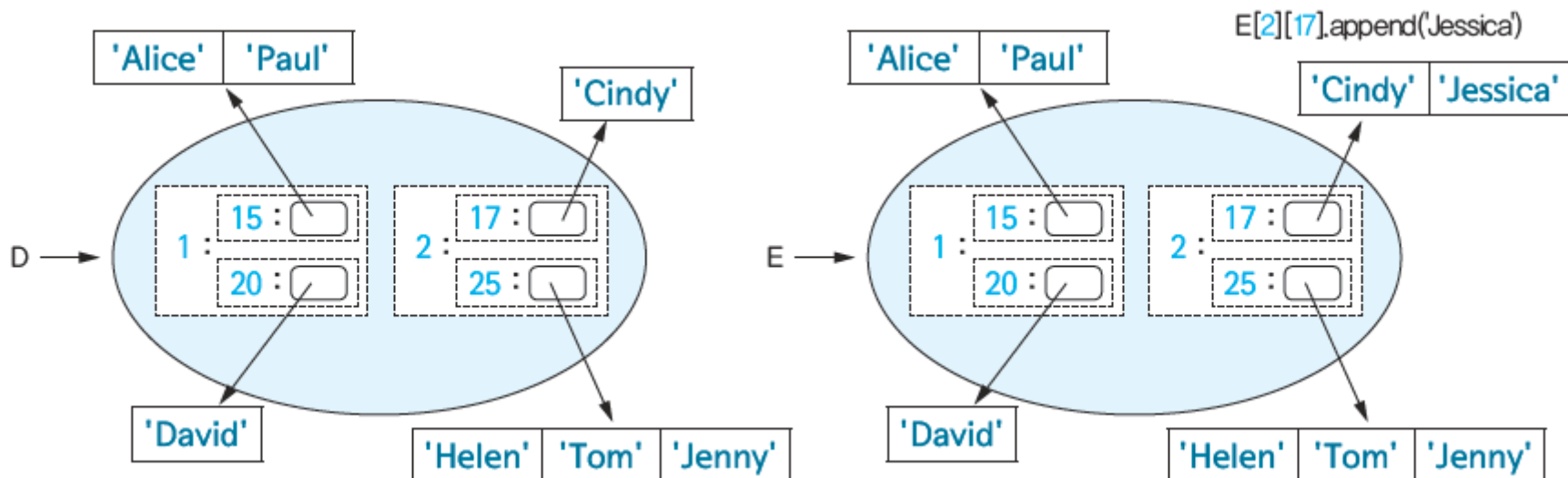
10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 사전이 저장되는 경우의 복사 예제
 - copy 모듈의 deepcopy() 함수

```
>>> D = {1:{15:['Alice', 'Paul'], 20:['David']},  
         2:{17:['Cindy'], 25:['Helen', 'Tom', 'Jenny']}}  
>>> import copy  
>>> E = copy.deepcopy(D)  
>>> E[2][17].append('Jessica')  
>>> D  
{1: {15: ['Alice', 'Paul'], 20: ['David']}, 2: {17: ['Cindy'], 25: ['Helen', 'Tom',  
'Jenny']}}  
>>> E  
{1: {15: ['Alice', 'Paul'], 20: ['David']}, 2: {17: ['Cindy', 'Jessica'], 25: ['Helen', 'Tom',  
'Jenny']}}
```

10. 사전의 값으로 mutable 자료형이 저장되는 경우

- ◆ 사전의 값으로 사전이 저장되는 경우의 복사 예제
 - copy 모듈의 deepcopy() 함수



11. 정리

- ◆ 사전의 원소는 키와 값의 쌍으로 구성됨.
- ◆ 사전의 형태는 복잡하지만, 활용도가 높은 자료형임.
- ◆ 사전은 mutable 자료형임.
- ◆ 사전의 키로 mutable 자료형은 사용할 수 없음.
- ◆ 사전이 제공하는 메소드들을 잘 활용할 수 있어야 함.