이 세상이나 세상에 있는 것들을 사랑하지 말라 누구든지 세상을 사랑하면 아버지의 사랑이 그 안에 있지 아니하니 이는 세상에 있는 모든 것이 육신의 정욕과 안목의 정욕과 이생의 자랑이니 다 아버지께로부터 온 것이 아니요 세상으로부터 온 것이라 이 세상도, 그 정욕도 지나가되 오직 하나님의 뜻을 행하는 자는 영원히 거하느니라 (요일2:15-17)

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

Chapter 8. Dictionary: 딕셔너리

학습 목표

- 딕셔너리의 데이터 구조를 익힌다
- 딕셔너리를 다루는 다양한 코딩 기법을 익힌다
- 딕셔너리를 사용하여 문제해결 역량을 갖춘다

학습 내용

1. 딕셔너리 데이터 구조의 소개
   - 딕셔너리 정의하기
   - 딕셔너리 특성
   - 딕셔너리 정의하기와 생성하기
   - 딕셔너리 자료 접근과 조작
2. 딕셔너리 메소드와 순회
3. 딕셔너리 컴프리헨션
   - 정렬
   - 익명함수(lambda function)

딕셔너리 데이터 구조 소개

Dictionary는 파이썬에서 상당히 중요한 위치를 차지합니다. 파이썬이란 언어 자체가 딕셔너리로 구성되어 있다고 해도 과언이 아닙니다. 예를 들면, Modules, classes, objects와 같은 것들이 사실상 딕셔너리로 되어 있습니다.

딕션너리 정의

파이썬의 공식적인 문서에는 Dictionary를 다음과 같이 정의합니다.

> An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods.

딕션너리 특성

딕션너리가 **연관 배열(associative array)** 이라는 말은 배열의 인덱스를 정수뿐만이 아니라 다른 다양한 타입으로 설정할 수 있는 배열을 의미합니다. 연관 배열을 파이썬에서는 딕셔너리 데이터 구조로 구현한 것이며, 딕셔너리는 키-값(key-value)으로 이루어진 데이터의 콜렉션이라 볼 수 있습니다.

이러한 연관 배열을 사용하면 **키** 즉 **배열의 인덱스** 에 좀 더 명확한 의미를 부여할 수 있습니다. 예를 들면, 주민번호 혹은 전화번호를 키 즉 배열의 인덱스로 사용할 수 있다는 것입니다. 그래서, 다음과 같은 두 가지를 기억해두어야 합니다.

1. 딕셔너리는 배열의 인덱스 역할을 하는 키를 변환(map)하여 배열이나 콜렉션에 자료를 저장합니다.
2. 딕셔너리의 키 값은 **hashable** 이어야 합니다. 다른 말로 키 값은 해시값이 있어야 하고, 또한 삭제하지 않는 한 바꿀 수 없습니다.

파이썬 시퀀스(sequence) 자료형(list, tuple)은 인덱스 0부터 시작하여 순서대로 자료가 저장되어 있고, 정수 인덱스로 접근하지만, 딕셔너리는 키(key)로 접근 가능하다는 것입니다.

파이썬의 set은 해시값(hash value)이 있고, 객체들이 중복되지 않아야 하는 것처럼 한 딕셔너리의 키 값들도 하나의 set처럼 중복하지 않고, 고유한 해시값이 있어야 합니다. 그러므로, 리스트같은 mutable한 객체는 키로 사용할 수 없지만 튜플은 키로 사용할 수 있습니다. 그러나 딕셔너리의 value에 대해서는 제약이 없이 어떠한 객체들도 사용할 수 있습니다.

딕셔너리가 파이썬 2.7 버전까지는 순서가 없는 unordered 데이터 구조였으나, 파이썬 3.7 이후부터는 순서가 있는 ordered 구조입니다.

딕셔너리는 리스트처럼 가변(mutability) 데이터 타입이기 때문에 사전을 선언 후에 자유롭게 새로운 키에 값을 추가하거나 갱신이 가능하고, 또한 기존 키와 값을 삭제할 수 있습니다.

동적 언어인 파이썬에서는 키로 해시가 가능한(hashable) 모든 데이터를 사용할 수 있고, 값에 대해서는 아무런 제한없이 어떤 데이터도 보관할 수 있습니다. 자바와 같이 동일한 타입의 키와 값만 사용하도록 제한하는 다른 언어와 상당히 대조되는 부분입니다.

딕션너리 정의하기

파이썬의 딕셔너리는 리스트와 튜플과 마찬가지로 객체의 컬렉션을 저장하지만, 객체들을 순서대로 배열에 저장하는 대신, 딕셔너리는 키-값 쌍(key-value pairs)라고 불리는 방식으로 정보를 저장합니다. 즉, 딕셔너리를 정의하는 문법은 다음과 같이 Key와 Value의 쌍 여러 개가 `{ }` 로 둘러싸여 있습니다. 각각의 요소는 `Key : Value` 형태로 이루어져 있고 쉼표(,)로 구분되어 집니다.

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

딕션너리 생성하기

주로 다음 세 가지 방법으로 생성할 수 있습니다.

1. key-value 쌍의 값들로 구성된 항목들을 `{ }` 묶음으로 생성할 수 있습니다.
2. key-value 쌍의 튜플들로 구성된 리스트로 `dict()` construct 호출합니다.

3. key-value 쌍의 값들이 단순한 문자열이면, 이를 keyword argument로 삼아서 `dict()` 를 호출합니다.

Method 1:

아래와 같이 key-value 쌍으로 값을 지정하고 대괄호 curly braces `{}` 로 묶어주는 방법입니다.

```
nickname = { 'Alabama': 'Cotton State', 'Alaska': 'The Last Frontier',
'Arizona': 'Grand Canyon State' }
teams = { 'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado':
'Rockies' }
```

In [1]:
```
nickname = { 'Alabama': 'Cotton State', 'Alaska': 'The Last Frontier', 'Arizona': 'Gra
print(nickname)
teams = { 'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies' }
print(teams)
```

```
{'Alabama': 'Cotton State', 'Alaska': 'The Last Frontier', 'Arizona': 'Grand Canyon St
ate'}
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
```

Method 2: key-pair 튜플로 구성된 항목들의 리스트로 `dict()` construct를 호출함으로 생성할 수 있습니다.

```
teams = dict([ ('Atlanta', 'Braves'), ('Boston', 'Red Sox'),
('Colorado', 'Rockies') ])
```

In [2]:
```
teams = dict([ ('Atlanta', 'Braves'), ('Boston', 'Red Sox'), ('Colorado', 'Rockies') ]
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
```

Method 3:

key-value 쌍의 값들이 단순한 문자열이라면, 이를 keyword argument로 삼아서 `dict()` construct를 호출합니다.

```
teams = dict(Atlanta = 'Braves', Boston = 'Red Sox', Colorado =
'Rockies')
```

In [3]:
```
teams = dict(Atlanta = 'Braves', Boston = 'Red Sox', Colorado = 'Rockies')
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
```

딕셔너리 항목을 추가, 삭제, 갱신하기

딕셔너리의 한 항목은 언제나 key-value 쌍으로 이루어져 있습니다.

추가하기

기존의 딕셔너리에 새로운 항목을 추가하기 위해, 새로운 key를 인덱스로 주고, 새로운 value를 추가할 수 있습니다.

Sample Run:

```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado':
'Rockies'}
```

Expected Output:

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies',
'Toronto': 'Blue Jay'}
```

In [2]:
```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
None
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies', 'Toronto': 'Blue Ja
y'}
```

갱신하기

이미 존재하는 항목의 키는 수정할 수 없지만, 값(value)은 덮어쓰기 형식으로 진행합니다.

다음 예제는 이미 딕셔너리에 존재하는 'Toronto'의 value를 'Blue Jays'로 갱신합니다.

Expected Output

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies',
'Toronto': 'Blue Jays'}
```

In [4]:
```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
None
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies'}
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies', 'Toronto': 'Blue Jay
s'}
```

삭제하기

`del` 함수를 사용해서 `del teams[key]` 처럼 입력하면 지정한 키에 해당하는 `{key : value}` 항목이 삭제됩니다.

'Colorado' 항목(key and value)을 삭제하십시오.

Expected Output

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays'}
```

In [6]:
```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Colorado': 'Rockies', 'Toronto':
del teams['Colorado']
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays'}
```

딕셔너리 키로 값(value) 구하기

딕셔너리 값(value)은 키(key)를 배열의 인덱스처럼 간주하여 값을 찾을수 있습니다.

teams 딕셔너리에서 'Atlanta' key의 value을 출력하십시오.

Expected Output

```
Braves
```

In [5]:
```
None
```

```
Braves
```

딕셔너리에 없는 객체, 예를 들면 `teams['Seattle']` , 토론토 팀 이름을 찾으려 하면 아래와 같이 오류가 발생합니다.

```
teams['Seattle']

---------------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
~\AppData\Local\Temp/ipykernel_22312/3662211662.py in <module>
----> 1 teams['Seattle']

KeyError: 'Seattle'
```

딕셔너리 연산자와 함수

딕셔너리를 자유자재로 사용하기 위해 딕셔너리가 자체적으로 가지고 있는 관련 함수와 메소드에 익숙해야 합니다. 몇 가지를 다루어 봅시다.

in, not in 연산자 - membership test

이러한 연산자는 딕셔너리의 키에 대하여 작동합니다. `in` 연산자는 지정한 키가 딕셔너리에 존재하면 True, 아니면 False를 반환합니다.

현재 teams를 출력하고, `Toronto` 팀이 있으면, True를 출력하십시오. 또한 `Seattle` 팀이 존재하지 않으면, 삽입하고 아래와 같이 출력하십시오.

Expected Output

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays'}
True
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays',
'Seattle': 'Mariner'}
```

In [8]:
```
teams= {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays'}
None
print(teams)
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays'}
True
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays', 'Seattle': 'Marine
r'}
```

데이터 순회

`in` 연산자를 통해서 딕셔너리에 있는 모든 데이터를 `for` 루프문으로 순회할 수 있습니다. 기본적으로는 키만 얻어지기 때문에, 값은 대괄호를 이용해서 접근해야 합니다.

현재 teams 딕셔너리를 출력하고, for 문을 사용하여 각 팀의 key와 value를 아래와 같이 출력하십시오.

Sample Run:

```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue
Jays', 'Seattle': 'Mariner'}
```

Expected Output

```
Atlanta: Braves
Boston: Red Sox
Toronto: Blue Jays
Seattle: Mariner
```

In [9]:
```
teams = {'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays', 'Seattle':
None
```

```
{'Atlanta': 'Braves', 'Boston': 'Red Sox', 'Toronto': 'Blue Jays', 'Seattle': 'Marine
r'}
Atlanta: Braves
Boston: Red Sox
Toronto: Blue Jays
Seattle: Mariner
```

Alternative:

하지만 `items()` 메서드를 활용하면 키와 값을 큐플의 형태로 한 번에 얻을 수 있어서 좀 더 깔끔하게 루프문을 작성할 수 있습ㄴ다.

In [10]:
```
None
```

```
Atlanta: Braves
Boston: Red Sox
Toronto: Blue Jays
Seattle: Mariner
```

딕셔너리 병합

여러 개의 사전을 합쳐야할 때는 ** 연산자를 사용하여, 중괄호 안에 합칠 사전들을 쉼표(,)로 구분하여 나열하면 됩니다.

파이썬 3.9부터는 ** 대신에 | 연산자를 사용하여 좀 더 편하게 사전을 병합할 수 있습니다. 예를 들면, `dic3 = {dic1 | dic2}` 코드가 가능합니다.

In [11]:
```
dic1 = {'apple': 1, 'banana': 2}
dic2 = {'banana': 5, 'lemon': 3, 'orange': 4}

dic3 = {**dic1, **dic2}
print(dic3)
```

```
{'apple': 1, 'banana': 5, 'lemon': 3, 'orange': 4}
```

len() 함수

딕셔너리의 key-value쌍으로 이루어진 항목의 갯수를 반환합니다.

In [12]:
```
len(teams)
```

Out[12]: 4

딕셔너리 메소드와 순회

딕셔너리 메소드

- `d.clear()` clears a dictionary
- `d.get(<key>[, <default>])` returns the value of a key if it exists
- `d.items()` returns a list of key-value pairs in a dict.
- `d.keys()` returns a list of keys in a dict.
- `d.values()` returns a list of values in a dict.
- `d.pop(<key>[, <default>])` removes a key from a dict, if present, and returns its value
- `d.popitem()` removes the last key-value pair from a dict and returns it as a tuple
- `d.update(<obj>)` merges a dict with another dict or with an iterable of key-value pair

딕셔너리 순회(Iterating)

딕혀너리의 항목들을 모두 순회하는데 다음과 같은 방법들이 있습니다.

- `d __iter()__` 를 호출하여 키들을 우히나 **반복자(iterator)** 를 반환한다.
- `d.items()` 딕셔너러의 키와 값으로 구성된 새로운 view를 리스트로 반환한다.
- `d.keys()` 딕셔너리의 키들로 구성된 리스트를 반환한다
- `d.values()` 딕셔너리의 값들로 구성된 리스트를 반환한다.

keys를 통해 직접 순회하기

딕셔너리는 파이썬의 mapping 클래스에 속하는 객체입니다. mapping 클래스로부터 상속받는 특별한 메소드들이 있습니다. 이러한 객체의 메소드와 속성을 `dir()` 로 찾아볼 수 있습니다. 아래와 같이 `dir()` 의 인자로 빈 딕셔너리 즉 `{}` 를 전달하는 식으로 코드 셀에서 호출하면 됩니다.

```
dir({})

['__class__', '__contains__', '__delattr__', 'delitem', ... ,
'__iter__', ...]
```

In [13]:
```
dir({})
```

Out[13]:
```
['__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
```

```
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

위의 `dir({})` 의 출력 결과의 리스트를 자세히 보면 `__iter__` 이 있습니다. 이것은 바로 딕셔너리, 리스트, 셋 같은 container 객체들이 가지고 있는 특별한 메소드들 중에 하나입니다. `__iter__` 메소드는 container에 있는 모든 항목들을 순회할 수 있도록 하는 iterator 객체를 반환합니다.

딕셔너리의 `__iter__` 메소드는 반환하는 iterator는 `key` 로 순회할 수 있도록 구현되어 있습니다. 아래의 예제에서 살펴볼 수 있습니다.

In [14]:
```
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
for key in fruits:
    print(key)
```

```
kiwi
mango
apple
banana
```

d.items()로 순회하기

딕셔너리를 다루려면, 대개 키와 값에 대하여 함께 작업할 때가 많습니다. 그러므로, 상당히 유용할 때가 많은 `.items()` 메소드는 딕셔너리 항목들에 대한 새로운 view 객체인 리스트를 반환해 줍니다.

예를 들면, 아래와 같이 각 항목을 **튜플** 로 전환하여 만든 튜플들의 리스트를 반환합니다.

In [15]:
```
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
ditems = fruits.items()
print(ditems)
```

```
dict_items([('kiwi', 1), ('mango', 9), ('apple', 2), ('banana', 8)])
```

그러므로, 튜플을 해체(unpacking)해야 각 항목의 키와 값을 다룰 수 있습니다. 다음과 같이 하면 간단히 각 키와 값을 튜플의 두 원소를 접근할 수 있습니다.

In [16]:
```
for key, value in fruits.items():
    print(key, '->', value)
```

```
kiwi -> 1
mango -> 9
apple -> 2
banana -> 8
```

d.keys()로 순회하기

`.keys()` 메소드는 딕셔너리의 키들로만 구성된 새로운 view 객체인 리스트를 반환해 줍니다.

In [17]:
```
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
ditems = fruits.keys()
print(ditems)
```

```
dict_keys(['kiwi', 'mango', 'apple', 'banana'])
```

`.keys()` 메소드로 키를 찾으며, 그 키를 사용하면, 그 키에 해당하는 값은 바로 구할 수 있습니다.

In [18]:
```
for key in fruits.keys():
    print(key, '->', fruits[key])
```

```
kiwi -> 1
mango -> 9
apple -> 2
banana -> 8
```

d.values()로 순회하기

`.values()` 메소드는 딕셔너리의 값들로만 구성된 새로운 view 객체인 리스트를 반환해 줍니다.

In [19]:
```
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
ditems = fruits.values()
print(ditems)
```

```
dict_values([1, 9, 2, 8])
```

In [20]:
```
for value in fruits.values():
    print(value)
```

```
1
9
2
8
```

`d.update(<obj>)`

> Merges a dictionary with another dictionary or with an iterable of key-value pairs.

만약 `<obj>` 이 딕셔너리이면, `d.update(<obj>)` 는 `<obj>` 항목들을 d에 합병합니다.

`<obj>` 에 있는 각 키에 대하여,

- 만약, 키가 `d`에 존재하지 않으면, `<obj>`의 key-value쌍은 `d`에 추가 됩니다.
- 만약, 키가 `d`에 존재하고 있다면, `<obj>`의 value로 갱신(update)됩니다.

In [21]:
```
d1 = {'a': 10, 'b': 20, 'c': 30}
d2 = {'b': 200, 'd': 400}

d1.update(d2)
d1
```

Out[21]:
```
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

Dictionary Comprehension

리스트 컴프리헨션처럼 딕셔너리 컴프리헨션도 간결한 코딩으로 작업을 할 수 있고 결과로 딕셔너리를 생성합니다. 예를 들어, 다음과 같은 두 리스트로 하나의 딕셔너리로 만들고 싶다면 다음과 같이 Dictionary Comprehension으로 코딩할 수 있습니다.

In [22]:
```
fruits = ['apple', 'banana', 'kiwi']
colors = ['red', 'yellow', 'brown']
d = {key: value for key, value in zip(fruits, colors)}
d
```

Out[22]:
```
{'apple': 'red', 'banana': 'yellow', 'kiwi': 'brown'}
```

여기서 `zip()` 은 두 개의 iterables 즉 `fruits` 와 `colors` 를 인자로 받아서 한 묶음으로 반복자 (`iterator`)를 만들어 튜플로 반환한 것을 key와 value로 각각 풀어서 새로운 딕셔너리를 생성하는데 사용한 것입니다.

딕셔너리 정렬과 순회(Sorting and Iterating)

파이썬 3.7 부터, 딕셔너리가 생성될 때 항목이 추가된 순서를 유지하므로, 항목들을 정렬하여 딕셔너를 새로 구성하면 정렬이 된 딕션너리가 산출이 됩니다.

Sorted by Keys

아래 코드는 `sorted()` 함수를 사용하여, **키로 정렬되어** 반환되는 항목들을 순서대로 받아서 딕셔너리로 구성합니다. 딕션너리를 만들지 않고, 다른 작업도 가능합니다.

In [23]:
```
# Python 3.7 이상
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
sorted_fruits = {k: fruits[k] for k in sorted(fruits)}
print(sorted_fruits)
```

```
{'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

In [24]:
```
for key in sorted(fruits):
    print(f"{key}: {fruits[key]}")
```

```
apple: 2
banana: 8
kiwi: 1
mango: 9
```

Soted by Values

딕셔너리의 항목들을 아래와 같이 값(value)을 기준으로 정렬하려면 어떻게 해야 할까요?

```
kiwi -> 1
apple -> 2
banana -> 8
mango -> 9
```

여기서도 키로 정렬할 때 사용한 `sorted()` 함수를 사용할 수 있습니다. 다만, 키가 아니라 다른 것을 기준으로 정렬하라는 명령할 수 있는 함수의 인자에 설정해야 합니다. 이러한 내용을 좀 더 자세히 살펴보려면, `help(sorted)` 명령을 실행해보면, 다음과 같은 설명을 볼 수 있습니다.

```
help(sorted)

Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in
ascending order.

    A custom key function can be supplied to customize the sort order,
and the
    reverse flag can be set to request the result in descending order.
```

설명에 제시된 것처럼, "a custom key function"을 인자로 설정하면 사용자가 원하는대로 정렬이 가능하다고 합니다. sorted 함수가 비교할 것(즉 여기 예제는 딕셔너리의 value)을 정해주는 함수를 만들고, 그 함수의 이름을 인자로 넘겨줍니다.

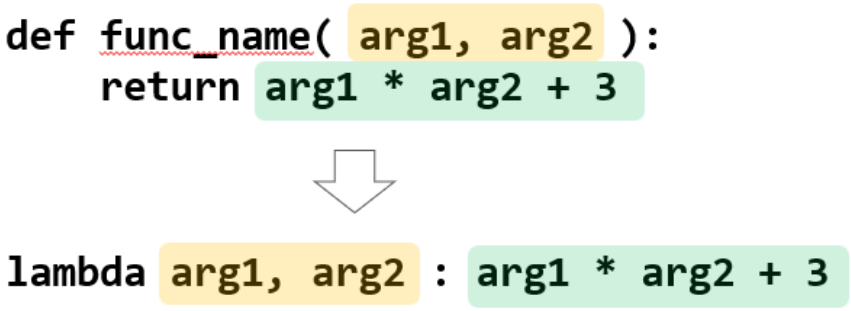
이 예제에서 우리가 비교할 것은 딕셔너리 (key, value) 튜플의 두 번째 것이므로, 이것을 반환하는 함수를 만들면 됩니다.

In [25]:
```
fruits = { 'mango': 9, 'kiwi': 1, 'apple': 2, 'banana': 8}

def by_value(item):
    return item[1]

for k, v in sorted(fruits.items(), key=by_value):
    print(k, '->', v)
```

```
kiwi -> 1
apple -> 2
banana -> 8
mango -> 9
```

Using Lambda function 익명함수

위의 `by_value()` 와 같은 함수는 여기서만 사용하고 다시 사용하지 않을 것으로 보입니다. 이런 경우, 이런 간단한 함수는 **익명함수** (**lambda함수**)로 처리하면 편리합니다. 람다함수는 함수이름이 없으며, `return` 을 사용하지 않고 자동으로 결과를 반환할 수 있습니다. 아래 그림을 참조하십시오.



예를 들면, x를 인자로 받아 5를 더해서 반환하는 이름없는 lambda함수는 다음과 같이 정의할 수 있습니다.

```
lambda x: x + 5
```

람다함수는 정의하면서 동시에 사용할 수 있으며, 아래와 같이 람다 함수에 3을 입력한 것이고 8이 출력됩니다.

```
(lambda x: x + 5)(3)
8
```

람다함수도 객체이기 때문에 정의와 동시에 변수에 저장할 수 있습니다. 다시 사용할 수 있습니다.

```
func = lambda x: x + 5
func(3)
8
```

In [26]:
```
fruits = { 'mango': 9, 'kiwi': 1, 'apple': 2, 'banana': 8}

for k, v in sorted(fruits.items(), key = lambda item: item[1]):
    print(k, '->', v)
```

```
kiwi -> 1
apple -> 2
banana -> 8
mango -> 9
```

Reversed

딕셔너리의 항목들을 거꾸로 정렬하면, `sorted()` 함수의 매개변수 `reverse = True` 로 설정하면 됩니다.

In [27]:
```
fruits = {'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
reversed = {k: fruits[k] for k in sorted(fruits, reverse=True)}
print(reversed)
print(fruits)
```

```
{'mango': 9, 'kiwi': 1, 'banana': 8, 'apple': 2}
{'kiwi': 1, 'mango': 9, 'apple': 2, 'banana': 8}
```

한 가지 유의할 점은 sorted() 함수는 딕셔너리 원자료는 변형하지 않고, 새로운 복사본을 반환하고 있다는 것입니다.

Complex Dictionary - 중첩 딕셔너리

파이썬에서 리스트 안에 리스트를, 튜플 안에 튜플을 중첩할 수 있는 것처럼 중첩 딕셔너리 또한 생성할 수 있습니다.

아래와 같이 주(state) 이름을 키로 하고, 주도(capital)와 주 별명을 값으로 하는 딕셔너리이지만, 그 값 자체가 또 다른 딕셔너리입니다.

In [28]:
```
states = {
    "Alabama": {"capital": "Montgomery", "nickname": "Cotton State"},
    "Arizona": {"capital": "Phoenix", "nickname": "Grand Canyon State" },
    "California": {"capital": "Sacramento", "nickname": "Golden State"},
    "New York": {"capital": "Albany", "nickname": "Empire State" },
    "Texas": {"capital": "Austin",  "nickname": "Lone Star State" }
}

states
```

Out[28]:
```
{'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany', 'nickname': 'Empire State'},
 'Texas': {'capital': 'Austin', 'nickname': 'Lone Star State'}}
```

다음과 같이 각 키의 값은 딕셔너리입니다.

In [29]:
```
states["Texas"]
```

Out[29]:
```
{'capital': 'Austin', 'nickname': 'Lone Star State'}
```

텍사스의 주 별명은 다음과 같이 얻을 수 있습니다.

In [30]:
```
states["Texas"]["nickname"]
```

Out[30]:
```
'Lone Star State'
```

In [31]:
```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
    "Texas": "Austin"
}
```

In [32]:
```
nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "New York": "Empire State",
```

```
    "Texas": "Lone Star State"
}
```

Exercises

리스트로 딕셔너리 생성하기

다음과 같이 두 개의 원소로 구성된 튜플들의 리스트는 `dict()` construct를 사용하여 바로 딕셔너리로 전환할 수 있습니다.

Sample Run:

```
alist = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]
```

Expected Outptut:

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

In [34]:
```
alist = ([('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)])
store = dict(alist)
print(store)
```

```
{'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

함수로 딕셔너리 생성하기

주어진 영어 문자열(단어)에서 각 알파벳 글자의 수를 세어서 딕셔너리로 반환하는 함수 `count()` 를 구현하고 테스트 하십시오.

Sample Run:

```
if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

Expected Output:

```
{'t': 1, 'e': 4, 'n': 2, 's': 2}
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

Solution:

In [35]:
```
def count(word):
    counter = {}
    for letter in word:
        if letter not in counter:
            counter[letter] = 0
        counter[letter] += 1
    return counter

if __name__ == '__main__':
    print(count('tennessee'))
```

```
    print(count('mississippi'))
```

```
{'t': 1, 'e': 4, 'n': 2, 's': 2}
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

함수로 딕셔너리 생성하기 - sorted()

주어진 영어 문자열(단어)에서 각 알파벳 글자의 수를 세고, **알파벳 순으로 정렬한** 딕셔너리로 반환하는 함수 `count()` 를 구현하고 테스트 하십시오.

Sample Run:

```
if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

Expected Output:

```
{'e': 4, 'n': 2, 's': 2, 't': 1}
{'i': 4, 'm': 1, 'p': 2, 's': 4}
```

Solution:

In [36]:
```
def count(word):
    counter = {}
    for letter in word:
        if letter not in counter:
            counter[letter] = 0
        counter[letter] += 1
    return { k: v for k, v in sorted(counter.items()) }

if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

```
{'e': 4, 'n': 2, 's': 2, 't': 1}
{'i': 4, 'm': 1, 'p': 2, 's': 4}
```

함수로 딕셔너리 생성하기 - lambda function

주어진 영어 문자열(단어)에서 각 알파벳 글자의 수를 세고, **키와 값을 교환하고, 낮은 갯수부터 정렬한** 딕셔너리로 반환하는 함수 `count()` 를 구현하고 테스트 하십시오.

Sample Run:

```
if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

Expected Output:

```
{1: 't', 2: 's', 4: 'e'}
{1: 'm', 2: 'p', 4: 's'}
```

Solution:

In [37]:
```
def count(word):
    counter = {}
    for letter in word:
        if letter not in counter:
            counter[letter] = 0
        counter[letter] += 1
    return { v: k for k, v in sorted(counter.items(), key=lambda item: item[1]) }

if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

```
{1: 't', 2: 's', 4: 'e'}
{1: 'm', 2: 'p', 4: 's'}
```

참고 사항 using setdefualt()

In [38]:
```
def count(word):
    counter = {}
    for letter in word:
        counter.setdefault(letter, 0)
        counter[letter] += 1
    return counter

if __name__ == '__main__':
    print(count('tennessee'))
    print(count('mississippi'))
```

```
{'t': 1, 'e': 4, 'n': 2, 's': 2}
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

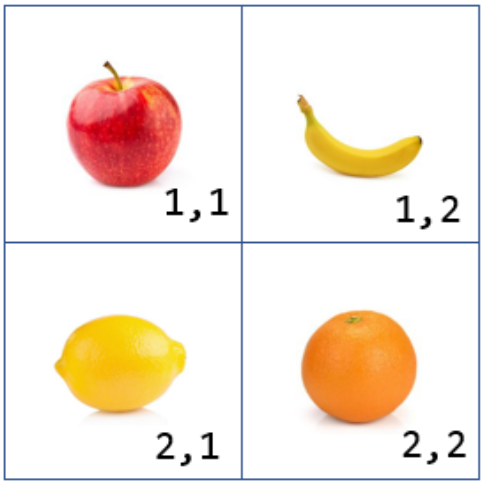
참고 사항 using defualtdict()

In [39]:
```
from collections import defaultdict

def count(word):
    counter = defaultdict(int)
    for letter in word:
        counter[letter] += 1
    return counter

if __name__ == '__main__':
    print(count("tennessee"))
    print(count("mississippi"))
```

```
defaultdict(<class 'int'>, {'t': 1, 'e': 4, 'n': 2, 's': 2})
defaultdict(<class 'int'>, {'m': 1, 'i': 4, 's': 4, 'p': 2})
```

딕셔너리 생성하기

아래 이미지와 같은 정보가 있다고 가정할 때, 각 박스에 지정되어 있는 두 숫자(key)와 과일의 이름(value)을 key-value 쌍으로 하는 딕셔너리를 생성하고, 각 과일의 key로 값을 출력하십시오. 과일의 이름은 apple, banana, lemon, and orange입니다.



Hint: 딕셔너리의 key로 사용하려면, 그 객체는 immutable해야 하고, hashable해야 합니다. 예를 들어, 리스트 `[1, 2]` 가 hashable인지 아닌지 테스트하려면, `hash([1, 2])` 실행해보면 됩니다.

In [40]:
```
hash((1, 2))
```

Out[40]:
```
-3550055125485641917
```

Solution:

In [41]:
```
# keys should be immutable obejcts. Tuple is ok, but not list.
fruits = { (1, 1):'apple', (1, 2): 'banana', (2, 1): 'lemon', (2, 2): 'orange' }

print(fruits[(1, 2)])
```

```
banana
```

키와 값을 교환한 딕셔너리 생성하기

주어진 딕셔너리 mydict에 있는 모든 항목의 키와 값을 서로 교환하여 새로운 딕셔너리 urdit를 생성하십시오.

Sample Run:

```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
```

Expected Output:

```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = {1: 'one', 2: 'two', 3: 'thee', 4: 'four'}
```

Using for loop

In [42]:
```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = {}
for key, value in mydict.items():
    urdict[value] = key
```

```
urdict
```

Out[42]:
```
{1: 'one', 2: 'two', 3: 'thee', 4: 'four'}
```

Using Dictionary Comprehension

In [43]:
```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = { value: key for key, value in mydict.items() }
urdict
```

Out[43]:
```
{1: 'one', 2: 'two', 3: 'thee', 4: 'four'}
```

이렇게 식으로 간결하고 효율적인 코딩을 하여 가독성을 높이는 것을 바람직한 일입니다. 이런 코딩 방식을 흔히 Pythonic Code라고 합니다.

조건에 맞는 항목들로 딕셔너리 생성하기

주어진 딕셔너리에 있는 항목들 중에 어떤 조건에 맞는 항목들로을 찾아서 새로운 딕셔너리를 만들려고 합니다. 다음에 주어진 딕셔너리 mydict에서 값이 짝수인 항목들로 구성한 딕셔너리 urdict를 생성하십시오.

Sample Run:

```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
```

Expected Output:

```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = {'two': 2, 'four': 4}
```

Using for-loop

In [44]:
```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = {}
for key, value in mydict.items():
    if value % 2 == 0:
        urdict[key] = value
print(urdict)
```

```
{'two': 2, 'four': 4}
```

Using Dictionary Comprehension

In [45]:
```
mydict = {'one': 1, 'two': 2, 'thee': 3, 'four': 4}
urdict = {k: v for k, v in mydict.items() if v % 2 == 0 }
print(urdict)
```

```
{'two': 2, 'four': 4}
```

계산하기

주어진 딕셔너리에 있는 항목들 중에 어떤 조건에 맞는 항목들로을 찾아서 새로운 딕셔너리를 만들려고 합니다. 다음에 주어진 딕셔너리 mydict에서 값이 짝수인 항목들로 구성한 딕셔너리 urdict를 생성하십시오.

Sample Run:

```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9,
'kiwi':49}
```

Expected Output:

```
99
```

Using for-loop

In [46]:
```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9, 'kiwi':49}
counts = 0
for value in fruits.values():
    counts += value

print(counts)
```

```
99
```

Using **list comprehension** & sum()

In [47]:
```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9, 'kiwi':49}
counts = sum([value for value in fruits.values()])
counts
```

Out[47]:
```
99
```

Using generator expression & sum()

아래 코드는 리스트 컴프리헨션을 사용한 것 같지만, 자세히 보면 `[]` 를 사용하는 대신에 `()` 를 사용하여 generator expression를 사용한 것입니다.

Generator는 다음과 같은 특징이 있습니다.

- Generator expression은 iterator를 반환하는 코드입니다.
- 반환 받은 iterator로 for-loop로 순회하거나 sum()함수에 넣어 합을 구할 수 있습니다.
- iterator 객체는 한번만 순회하는데 사용할 수 있습니다.
- 더 자세한 것은 여기를 참조하세요.

In [48]:
```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9, 'kiwi':49}
counts = (value for value in fruits.values())
print(counts)  # iterator object
for x in counts:
    print(x)
sum(counts)    # for loop에서 사용했기 때문에 더 이상 순회할 것이 없음.
```

```
<generator object <genexpr> at 0x0000019B78B34040>
12
21
```

```
8
9
49
```

Out[48]: 0

In [49]:
```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9, 'kiwi':49}
sum(value for value in fruits.values())
```

Out[49]: 99

Using sum() simply

In [50]:
```
fruits = {'apple': 12, 'banana': 21, 'lemon': 8, 'orange': 9, 'kiwi':49}
sum(fruits.values())
```

Out[50]: 99

두 개의 리스트로 딕셔너리 생성하기

다음과 같이 크기가 같은 두 개의 리스트로 하나의 딕셔너리를 생성하십시오.

Sample Run:

```
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]
```

Expected Output:

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Hint:

- Use `zip()` to combine fruit and count as a pair.
- Use list comprehension to make a list of those pairs.
- Use `dict()` construct to convert the list to a dict data type

Solution 1: Using for-loop

`(apple, 2)` 와 같은 튜플들로 구성된 리스트를 for-loop로 만들고 난 후, 그 결롸를 딕셔너리로 전환합니다.

In [51]:
```
#converting two lists or sets into one dict type object
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]
alist = []
for f, c in zip(fruit, count):
    alist.append((f, c))
store = dict(alist)
print(store)
```

```
{'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Solution 2: Using list-comprehension

리스트 컴프리헨션으로 튜를들로 이루어진 리스트를 딕셔너리로 전환합니다.

In [52]:
```
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]

store = dict([(f, c) for f, c in zip(fruit, count)])
print(store)
```

```
{'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Create the following dict type data set.

```
data = {'a': 1, 'b': 2, 'c': 3, ..., 'x': 24, 'y': 25, 'z': 26}
```

- You may import `string` to get ASCII lowercase as shown below:

```
from string import ascii_lowercase as lowers
```

Solution 1: Using dict() function and zip().

In [53]:
```
from string import ascii_lowercase as lowers

data = dict( zip(lowers, range(1, len(lowers) + 1)) )
print(data)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k':
11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26}
```

In [54]:
```
import string

data = dict( zip(string.ascii_lowercase, range(1, len(lowers) + 1)) )
print(data)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k':
11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26}
```

Solution 2: Using dict comprehension

In [55]:
```
data = {ch: num for ch, num in zip(lowers, range(1, 27))}
print(data)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k':
11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26}
```

딕셔너리 항목들을 한 줄로 출력하기(1)

주어진 딕셔너리의 내용을 한 줄로 출력하길 원합니다.

Hint:

- Use for-loop and items().
- Use some options in print().

Sample Run:

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Expected Output:

```
apple:2, banana:8, kiwi:1, mango:9,
```

In [56]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}

for key, value in store.items():
    print(key, value, sep = ':', end = ', ')
```

```
apple:2, banana:8, kiwi:1, mango:9,
```

딕셔너리 항목들을 한 줄로 출력하기(2)

주어진 딕셔너리의 내용을 끝에 extra comma없이 한 줄로 깔끔하게 출력하길 원합니다.

Hint:

- Use list comprehension and join()

Sample Run:

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Expected Output:

```
apple:2, banana:8, kiwi:1, mango:9
```

In [57]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
print(', '.join(k + ':' + str(v) for k, v in store.items()))
```

```
apple:2, banana:8, kiwi:1, mango:9
```

Soted by key's length using a custom function

주어진 딕셔너리의 항목들을 아래와 같이 키(key) 문자열의 길이를 기준으로 정렬하려고 합니다.
여기서는 `def by_length(item)` 함수를 정의하고, 이를 사용하십시오.

Sample Run:

```
fruits = { 'apple':2, 'kiwi': 5, 'pineapple': 3, 'banana': 8}
```

Expected Output:

```
kiwi -> 5
apple -> 2
banana -> 8
pineapple -> 3
```

여기서도 value로 정렬할 때 사용한 `sorted()` 함수와 `sorted()` 함수의 인자로 비교/정렬하기 원하는 값을 함수로 지정해주는 방법이 있습니다. 본 단원에서 제공하는 예제를 살펴보십시오. 좀 더 자세히 살펴보려면, `help(sorted)` 명령을 실행해보면, 다음과 같은 설명을 볼 수 있습니다.

Solution:

In [58]:
```
fruits = { 'apple':2, 'kiwi': 5, 'pineapple': 3, 'banana': 8}

def by_length(item):
    return len(item[0])

for k, v in sorted(fruits.items(), key=by_length):
    print(k, '->', v)
```

```
kiwi -> 5
apple -> 2
banana -> 8
pineapple -> 3
```

Soted by key's length using a lambda function

주어진 딕셔너리의 항목들을 아래와 같이 키(key) 문자열의 길이를 기준으로 정렬하려고 합니다. 여기서는 익명함수(lambda function)을 사용하십시오. 그러면, 별도의 함수를 정의하지 않고도, for-loop에서 직접 처리할 수 있습니다. 이러한 코딩 기법을 가리켜 Pythonic way of coding, 간단히 Pythonic Coding이라고 합니다.

Sample Run:

```
fruits = { 'apple':2, 'kiwi': 5, 'pineapple': 3, 'banana': 8}
```

Expected Output:

```
kiwi -> 5
apple -> 2
banana -> 8
pineapple -> 3
```

In [59]:
```
fruits = { 'apple':2, 'kiwi': 5, 'pineapple': 3, 'banana': 8}

for k, v in sorted(fruits.items(), key=lambda item: len(item[0])):
    print(k, '->', v)
```

```
kiwi -> 5
apple -> 2
banana -> 8
pineapple -> 3
```

딕셔너리 생성하기(1)

다음과 같이 각각 State 이름을 키로 사용하는 딕셔너리 두 개가 있다고 가정합시다. 하나는 State의 capital을 value로 저장하고 있고, 다른 하나는 State의 별명(nickname)을 value로 가지고 있습니다. 두 딕셔너리에는 같은 State들의 정보들이 각각 나열되어 있다고 가정합니다.

두 딕셔너리 곧 `capitals` 와 `nicknames` 를 합병하여, 키는 state가 되고, 값은 capital과 nickname이 되도록 `states` 딕셔너리를 생성하십시오.

Hint:

`states` 딕셔너리의 값은 `{ }` 으로 즉 set으로 정의됨으로 그 안에 원소들의 순서는 우리가 조정할 수 없습니다.

Sample Run:

```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
    "Texas": "Austin"
}

nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "New York": "Empire State",
    "Texas": "Lone Star State"
}
```

Expected Outptut:

```
states = {
 'Alabama': {'Montgomery', 'Cotton State'},
 'Arizona': {'Phoenix', 'Grand Canyon State'},
 'California': {'Sacramento', 'Golden State'},
 'New York': {'Albany', 'Empire State'},
 'Texas': {'Austin', 'Lone Star State'}
 }
```

In [60]:
```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
    "Texas": "Austin"
}
```

In [61]:
```
nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "New York": "Austin",
    "Texas": "Lone Star State"
}
```

In [62]:
```
states = {}
for key, value in capitals.items():
    states[key] = { value, nicknames[key] }

states
```

Out[62]:
```
{'Alabama': {'Cotton State', 'Montgomery'},
 'Arizona': {'Grand Canyon State', 'Phoenix'},
 'California': {'Golden State', 'Sacramento'},
 'New York': {'Albany', 'Austin'},
 'Texas': {'Austin', 'Lone Star State'}}
```

In [63]:
```
states = { key: {value, nicknames[key]} for key, value in capitals.items() }
states
```

Out[63]:
```
{'Alabama': {'Cotton State', 'Montgomery'},
 'Arizona': {'Grand Canyon State', 'Phoenix'},
 'California': {'Golden State', 'Sacramento'},
 'New York': {'Albany', 'Austin'},
 'Texas': {'Austin', 'Lone Star State'}}
```

중첩 딕셔너리 생성하기(2)

다음과 같이 각각 State 이름을 키로 사용하는 딕셔너리 두 개가 있다고 가정합시다. 하나는 State의 capital을 value로 저장하고 있고, 다른 하나는 State의 별명(nickname)을 value로 가지고 있습니다. 두 딕셔너리에는 같은 State들의 정보들이 각각 나열되어 있다고 가정합니다.

두 딕셔너리 곧 `capitals` 와 `nicknames` 를 병합하여 `states` 중첩 딕셔너리를 생성하십시오.

Hint:
`states` 딕셔너리의 값은 `{ key-value }` 로 즉 딕셔너리로 정의됨으로 그 안에 원소들은 더 해지는 순서로 정해집니다.

Sample Run:

```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
    "Texas": "Austin"
}

nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "New York": "Empire State",
    "Texas": "Lone Star State"
}
```

Expected Outptut:

```
states = {
 'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany', 'nickname': 'Empire State'},
 'Texas': {'capital': 'Austin', 'nickname': 'Lone Star State'}
 }
```

In [64]:
```
capitals = {
```

```
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
    "Texas": "Austin"
}
```

In [65]:
```
nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "New York": "Austin",
    "Texas": "Lone Star State"
}
```

In [66]:
```
states = {}
for key, value in capitals.items():
    states[key] = {'capital': value, 'nickname': nicknames[key] }

states
```

Out[66]:
```
{'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany', 'nickname': 'Austin'},
 'Texas': {'capital': 'Austin', 'nickname': 'Lone Star State'}}
```

Step 3:

위의 코드를 딕셔너리 컴프리헨션으로 변환하십시오.

In [67]:
```
states = { key: {'capital': value, 'nickname': nicknames[key]} for key, value in capi
states
```

Out[67]:
```
{'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany', 'nickname': 'Austin'},
 'Texas': {'capital': 'Austin', 'nickname': 'Lone Star State'}}
```

중첩 딕셔너리 생성하기 (3)

다음과 같이 각각 State 이름을 키로 사용하는 딕셔너리 두 개가 있다고 가정합시다. 하나는 State의 `capital` 키에, 주도를 value로 저장하고 있고, 다른 하나는 State의 별명(nickname)을 value로 가지고 있습니다. 두 딕셔너리에 있는 주들이 갯수나 이름들이 일치하지 않을 수도 있습니다. 예를 들면, 'New York'주는 `capitals` 딕셔너리에는 존재하지만, `nicknames` 딕셔너리에는 존재하지 않습니다. `Texas` 주는 그 반대의 경우입니다.

이러한 두 딕셔너리(`capitals, nicknames`)를 병합하여 새로운 중첩 딕셔너리 `states` 를 생성하십시오.

Sample Run:

```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
```

```
    "California": "Sacramento",
    "New York": "Albany",
}

nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "Texas": "Lone Star State"
}
```

Expected Outptut:

```
states = {
 'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany'},
 'Texas': {'nickname': 'Lone Star State'}
 }
```

In [68]:
```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
}
```

In [69]:
```
nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "Texas": "Lone Star State"
}
```

In [70]:
```
one={'a': [1, 2], 'c': [5, 6], 'b': [3, 4]}
two={'a': [2.4, 3.4], 'c': [5.6, 7.6], 'b': [3.5, 4.5]}
three={'a': 1.2, 'c': 3.4, 'b': 2.3, 'x': 5}

{key: value + two[key] for key, value in one.items() }
```

Out[70]:
```
{'a': [1, 2, 2.4, 3.4], 'c': [5, 6, 5.6, 7.6], 'b': [3, 4, 3.5, 4.5]}
```

In [71]:
```
states = {}
for key, value in capitals.items():
    states[key] = {value}
    if key in nicknames:
        states[key].update({ nicknames[key] })

for key, value in nicknames.items():
    if key not in capitals:
        states[key] = { nicknames[key] }

states
```

```
{'Alabama': {'Cotton State', 'Montgomery'},
```

Out[71]:
```
'Arizona': {'Grand Canyon State', 'Phoenix'},
'California': {'Golden State', 'Sacramento'},
'New York': {'Albany'},
'Texas': {'Lone Star State'}}
```

In [72]:
```
states = {}
for key, value in capitals.items():
    states[key] = { 'capital': value }
    if key in nicknames:
        states[key].update({ 'nickname': nicknames[key] })

for key, value in nicknames.items():
    if key not in capitals:
        states[key] = {'nickname': nicknames[key] }

states
```

Out[72]:
```
{'Alabama': {'capital': 'Montgomery', 'nickname': 'Cotton State'},
 'Arizona': {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'},
 'California': {'capital': 'Sacramento', 'nickname': 'Golden State'},
 'New York': {'capital': 'Albany'},
 'Texas': {'nickname': 'Lone Star State'}}
```

중첩 딕셔너리 생성하기 (4)

다음과 같이 각각 State 이름을 키로 사용하는 딕셔너리 두 개가 있다고 가정합시다. 하나는 State의 `capital` 키에, 주도를 value로 저장하고 있고, 다른 하나는 State의 별명(nickname)을 value로 가지고 있습니다. 두 딕셔너리에 있는 주들이 갯수나 이름들이 일치하지 않을 수도 있습니다. 예를 들면, 'New York'주는 `capitals` 딕셔너리에는 존재하지만, `nicknames` 딕셔너리에는 존재하지 않습니다. `Texas` 주는 그 반대의 경우입니다.

이러한 두 딕셔너리(`capitals, nicknames`)를 병합하여 새로운 중첩 딕셔너리 `states` 를 생성하십시오.

Sample Run:

```
capitals = {
    "California": "Sacramento",
    "New York": "Albany",
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix"
}

nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "Texas": "Lone Star State"
}

flowers = {
    "New York": "Rose",
    "Florida": "Orange Blossom",
    "Georgia": "Native Azalea",
    "Alabama": "Camellia",
}
```

Expected Outputut: (after sorted)

```
states = {
    Alabama : {'capital': 'Montgomery', 'nickname': 'Cotton State',
'flower': 'Camellia'},
    Arizona : {'capital': 'Phoenix', 'nickname': 'Grand Canyon
State'},
    California : {'capital': 'Sacramento', 'nickname': 'Golden
State'},
    Florida : {'flower': 'Orange Blossom'},
    Georgia : {'flower': 'Native Azalea'},
    New York : {'capital': 'Albany', 'flower': 'Rose'},
    Texas : {'nickname': 'Lone Star State'}
 }
```

In [73]:
```
capitals = {
    "Alabama":  "Montgomery",
    "Arizona": "Phoenix",
    "California": "Sacramento",
    "New York": "Albany",
}
```

In [74]:
```
nicknames = {
    "Alabama": "Cotton State",
    "Arizona": "Grand Canyon State",
    "California": "Golden State",
    "Texas": "Lone Star State"
}
```

In [75]:
```
flowers = {
    "Florida": "Orange Blossom",
    "Georgia": "Native Azalea",
    "Alabama": "Camellia",
    "New York": "Rose"         # ["Rose", "Lilac"]
}
```

Step 1:

In [76]:
```
states = dict()
for key, value in capitals.items():
    states[key] = { value }

    if key in nicknames:
        states[key].update({ nicknames[key] })
    if key in flowers:
        states[key].update({ flowers[key] })

for key, value in nicknames.items():
    if key not in capitals:
        states[key] = { nicknames[key] }

for key, value in flowers.items():
    if key not in capitals:
        states[key] = {flowers[key]}

for k, v in sorted(states.items()):
    print(k, ':', v)
```

```
Alabama : {'Camellia', 'Montgomery', 'Cotton State'}
Arizona : {'Grand Canyon State', 'Phoenix'}
California : {'Sacramento', 'Golden State'}
Florida : {'Orange Blossom'}
Georgia : {'Native Azalea'}
New York : {'Albany', 'Rose'}
Texas : {'Lone Star State'}
```

Step 2:

In [77]:

```
states = {}
for key, value in capitals.items():
    states[key] = { 'capital': value }
    if key in nicknames:
        states[key].update({ 'nickname': nicknames[key] })
    if key in flowers:
        states[key].update({ 'flower': flowers[key] })

for key, value in nicknames.items():
    if key not in capitals:
        states[key] = { 'nickname': nicknames[key] }

for key, value in flowers.items():
    if key not in capitals:
        states[key] = { 'flower': flowers[key] }

for k, v in sorted(states.items()):
    print(k, ':', v)
```

```
Alabama : {'capital': 'Montgomery', 'nickname': 'Cotton State', 'flower': 'Camellia'}
Arizona : {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'}
California : {'capital': 'Sacramento', 'nickname': 'Golden State'}
Florida : {'flower': 'Orange Blossom'}
Georgia : {'flower': 'Native Azalea'}
New York : {'capital': 'Albany', 'flower': 'Rose'}
Texas : {'nickname': 'Lone Star State'}
```

In [78]:

```
states = {}
for key in {*capitals.keys(), *nicknames.keys(), *flowers.keys()}:
    states[key] = {}

for key in states:
    if key in capitals:
        states[key].update( {'capital': capitals[key] } )

    if key in nicknames:
        states[key].update(  { 'nickname': nicknames[key] } )

    if key in flowers:
        states[key].update( { 'flower': flowers[key] } )

for k, v in sorted(states.items()):
    print(k, ':', v)
```

```
Alabama : {'capital': 'Montgomery', 'nickname': 'Cotton State', 'flower': 'Camellia'}
Arizona : {'capital': 'Phoenix', 'nickname': 'Grand Canyon State'}
California : {'capital': 'Sacramento', 'nickname': 'Golden State'}
Florida : {'flower': 'Orange Blossom'}
Georgia : {'flower': 'Native Azalea'}
New York : {'capital': 'Albany', 'flower': 'Rose'}
Texas : {'nickname': 'Lone Star State'}
```

참고 사항: using chain and defaultdict

In [79]:
```
from itertools import chain
from collections import defaultdict

states = defaultdict(list)

for k, v in chain(capitals.items(), nicknames.items(), flowers.items()):
    states[k].append( v )

for k, v in sorted(states.items()):
    print(k, ':', v)
```

```
Alabama : ['Montgomery', 'Cotton State', 'Camellia']
Arizona : ['Phoenix', 'Grand Canyon State']
California : ['Sacramento', 'Golden State']
Florida : ['Orange Blossom']
Georgia : ['Native Azalea']
New York : ['Albany', 'Rose']
Texas : ['Lone Star State']
```

학습정리

1. 딕셔너리 데이터 구조와 그 특성을 이해하고,
2. 딕셔너리 어떻게 정의하는지,
3. 딕셔너리 자료 접근과 조작하는 다양한 기법을 배웠고,
4. 딕셔너리 연산자와 함수를 사용해 보며,
5. 딕셔너리 다루는 예제들도 많이 연습해 보았습니다. 또한
6. 딕셔너리 컴프리헨션과 람다(익명)함수도 다루었습니다.
7. 마지막으로 딕셔너리들을 병합(merge)하는 것도 시도하였습니다.

참고자료

-

In [ ]:
```
```

슬기로운 자는 재앙을 보면 숨어 피하여도 어리석은 자들은 나가다가 해를 받느니라. 잠언27:12