Python Data Structure

ESM2014-41

객체지향프로그래밍 및 실습

SKKU 시스템경영공학과

조영일

Python Data Structure

- 문자열, 숫자와 같은 단순 자료형이 아닌 다양한 변수의 집합을 효과적으로 관리하기 위한 구조

Python Data Structure

- Python Built-in Data Structure
  - Lists
    - (Iterable vs Iterator)
    - (Mutable vs Immutable)
  - Dictionaries
  - Tuples
  - Sets

Python Data Structure - Lists

- Lists : 순서가 존재하는 유한한 여러 아이템들의 모음
  - '[ ]' 사이에 ','로 구분하여 값을 나열함
  - 각각의 값들은 서로 다른 data type 가능

```
empty_list = [] # 비어있는 리스트 생성

number_list = [1, 2, 3, 4]  # 숫자를 원소로 가지는 리스트
string_list = ["a", "b", "c"]   # 문자열을 원소로 가지는 리스트
mixed_list = [1, 2, "c"] # 혼합도 가능
```

Python Data Structure – Lists Indexing & Slicing

- 문자열과 같은 방식으로 Indexing & Slicing 탐색 가능
  (즉, Python은 문자열을 List의 일종으로 구현함)


```
number_list = [1, 2, 3, 4, 5]

number_list[0] == 1 # True
number_list[0:2] == [1, 2] # True
number_list[2:] == [3, 4, 5] # True
number_list[:2] == [1, 2] # True
```

Python Data Structure – Lists Item Deletion

del list[i]

- List 내 특정 위치의 항목 삭제

```
>>> menus = ['espresso', 'americano', 'flat white', 'cafe mocha']
>>> del menus[0]
>>> menus
['americano', 'flat white', 'cafe mocha']
>>> del menus[2]
>>> menus
['americano', 'flat white']
```

Python Data Structure – Nested Lists

- 중첩된 리스트 : List 역시 다른 List 의 아이템이 될 수 있음

```
string_list = ["a", "b" "c"]
number_list = [1, 2, 3]

nested_list = [string_list, number_list]
nested_list # => [["a", "b", "c"], [1, 2, 3]]

nested_list[0][0] == "a" # True
nested_list[1][1] == 2
```

Python Data Structure – Lists Operators

- List Concatenation : 두개의 리스트를 결합

list1 + list2

```
>>> list1 = ['a', 'b', 'c']
>>> list2 = ['d', 'e', 'f']
>>> list3 = list1 + list2
>>> list3
['a', 'b', 'c', 'd', 'e', 'f']
```

Python Data Structure – Lists Operators

- List Repetition : 리스트의 원소들을 반복

list * n

```
>>> list1 = ['a', 'b', 'c']
>>> list2 = list1 * 3
>>> list2
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

Python Data Structure – Lists Methods

- append() : 맨 뒤에 아이템 추가
- extend() : 맨 뒤에 다른 list 붙이기

```
>>> lst = [1, 2, 3, 4]
>>> lst.append(5)
>>> lst
[1, 2, 3, 4, 5]
>>> lst.extend([6, 7]) # list를 extend
>>> lst
[1, 2, 3, 4, 5, 6, 7]
>>> lst.append([8, 9]) # list를 append
>>> lst
[1, 2, 3, 4, 5, 6, 7, [8, 9]]
```

Python Data Structure – Lists Methods

- count() : 특정 값 아이템의 등장회수 리턴
- index() : 특정 값 아이템의 첫 index 리턴

```
>>> lst = [1, 2, 3, 2, 2, 4, 5, 6, 7]
>>> lst.count(2)
3
>>> lst.count(7)
1
>>> lst.index(2)
1
>>> lst.index(7)
8
```

Python Data Structure – Lists Methods

- insert() : 특정 위치에 아이템추가

```
>>> list_1 = ['one', 'two', 'four', 'five']
>>> list_1.insert(0, 'zero')
>>> list_1
['zero', 'one', 'two', 'four', 'five']
>>> list_1.insert(2, 'three')
>>> list_1
['zero', 'one', 'three', 'two', 'four', 'five']
```

Python Data Structure – Lists Methods

- reverse() : 아이템들의 순서 뒤집음

```
>>> list_1 = [1, 2, 3, 4, 5]
>>> list_1.reverse()
>>> list_1
[5, 4, 3, 2, 1]
```

Python Data Structure – Lists Methods

- clear() : 모든 아이템 삭제(리스트를 비움)
- remove() : 첫번째로 등장하는 해당 아이템을 삭제

```
>>> list_1 = [1, 2, 3, 4, 5]
>>> list_1.clear()
>>> list_1
[]

>>> list_1 = [1, 2, 3, 2, 5, 7, 2]
>>> list_1.remove(2)
>>> list_1
[1, 3, 2, 5, 7, 2]
```

Python Data Structure – Lists Methods

- pop() : 맨 뒤 아이템을 반환하고 리스트에서 삭제함

```
>>> list_1 = [1, 2, 3, 4, 5]
>>> list_1.pop() # 기본 동작은 맨뒤의 값을 반환하고 삭제
5
>>> list_1
[1, 2, 3, 4]
>>> list_1.pop(2) # 특정 위치의 값을 반환하고 삭제 할 수 있음
3
>>> list_1
[1, 2, 4]
```

Python Data Structure – Lists Methods

- sort() : List의 정렬(오름차순 / 내림차순)

```
>>> list_1 = [3, 5, 1, 9, 7]
>>> list_1.sort()
>>> list_1
[1, 3, 5, 7, 9]

>>> list_1 = [3, 5, 1, 9, 7]
>>> list_1.sort(reverse = True)
>>> list_1
[9, 7, 5, 3, 1]
```

Python Data Structure – Lists Methods

- list.append()
- list.extend()
- list.count()
- list.index()
- list.insert()
- list.clear()
- list.remove()
- list.reverse()
- list.sort()
- list.pop()
- list.copy()

More Details :

https://docs.python.org/3.7/tutorial/datastructures.html

Python Data Structure – Iterable

- Iterable : Iteration이 가능한 객체, 자기자신의 원소(member)를 순서대로 반환 할 수 있는 객체
- Python에서 효율적인 코드를 작성하기 위해 특징적으로 사용하는 개념

- Strings, Lists, Tuples 등이 대표적

Python Data Structure – Iterable

```
string_list = ["a", "b" "c"]
abc_string = "abcdef"

# Iterable 의 길이
len(string_list) == 3
len(abc_string) == 6

# 'in' operator
"a" in string_list == True
"y" in abc_string == False
"def" in abc_string == True
```

Python Data Structure – Iterator

- Iterator : 값을 차례대로 꺼낼 수 있는 객체
- Iterable과 Iterator는 같지 않다.
- Iterable 객체의 경우 Python 내장 함수인 iter()를 통해 Iterator로 변환 가능

Python Data Structure – Iterator

In [1]:
```
# List => Iterable
test_list = [1, 2, 3]
```

In [2]:
```
test_list
```

Out[2]:
```
[1, 2, 3]
```

In [3]:
```
# iter() method
test_iter = iter(test_list)
```

In [4]:
```
test_iter
```

Out[4]:
```
<list_iterator at 0x7fc7f04d4358>
```

In [6]:
```
# Iterator 객체에서는 .__next__() 메서드를 통해서 순서대로 값을 꺼낼 수 있음
test_iter.__next__()
```

Out[6]:
```
1
```

In [7]:
```
test_iter.__next__()
```

Out[7]:
```
2
```

In [8]:
```
test_iter.__next__()
```

Out[8]:
```
3
```

Python Data Structure – Mutable vs Immutable

- Tricky example – Why differences occur?

```
>>> a = 1
>>> b = a
>>> b += 1 # Change "b" value
>>> print(a) # Variable "a" will not change
>>> 1

>>> list_1 = ["a", "b", "c"]
>>> list_2 = list_1
>>> del(list_2[0]) # Change "list_2" value
>>> print(list_1) # Also changes "list_1"
["b", "c"]
```

Python Data Structure – Mutable vs Immutable

- Python에서 모든 변수는 객체(Objects)이다.(객체의 정의는 추후 다룸)

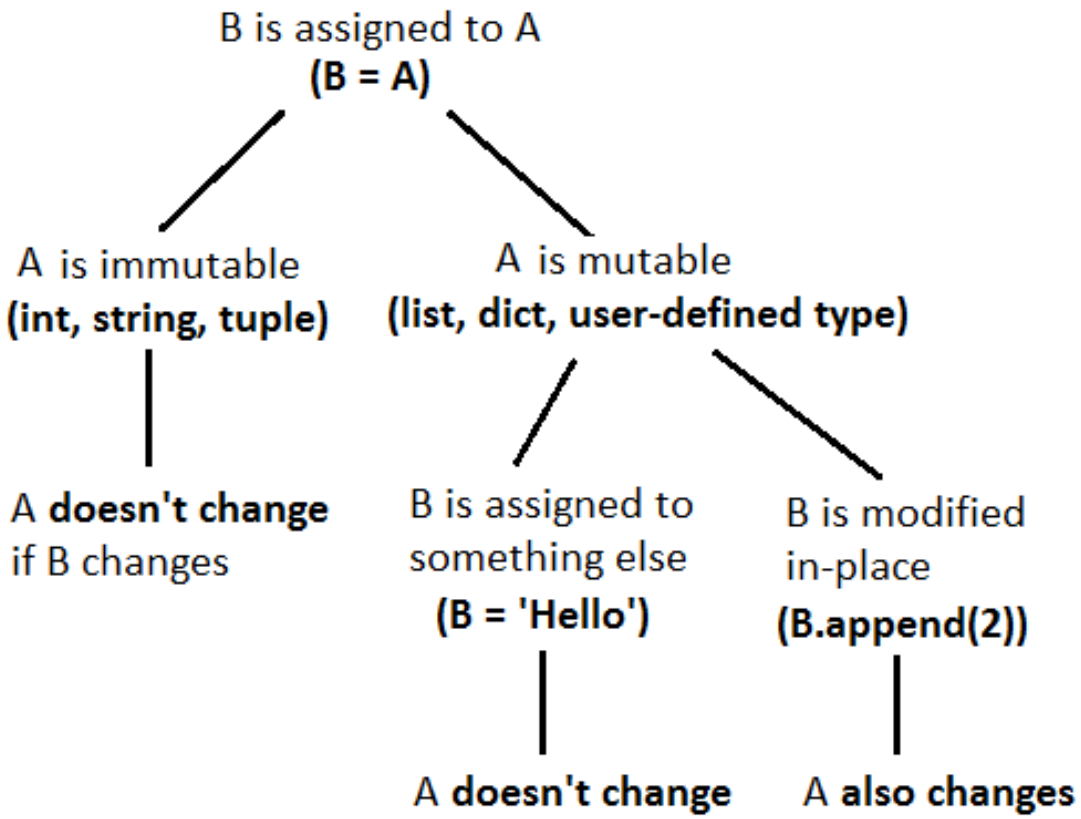
- 모든 객체는 두가지 유형으로 나뉜다.
  - Mutable : 객체를 선언한 후, 객체의 값을 수정 가능, 변수는 값이 수정된 같은 객체를 가리키게 됨 (list, set, dict)
  - Immutable : 객체를 생성한 후, 객체의 값을 수정 불가능, 값을 변경할 경우 해당 값을 가진 다른 객체를 가리키게 됨, (int, float, complex, bool, string, tuple)

Python Data Structure – Mutable vs Immutable

- 아래 예제에서 a와 b는 서로 같은 값을 가지지만, Int는 선언 이후 변경이 불가능(Immutable) 하므로 b와 a는 다른 객체이다. 따라서 b의 값을 변경하여도 a에 영향을 미치지 않는다.
- 반면 List는 Mutable 하므로 list_2와 list_1은 같은 객체를 가르킨다.(변수의 이름이 다를뿐 사실 같은 객체를 공유) 따라서 list_2의 값을 변경하면 list_1의 값도 변한다.

```
>>> a = 1
>>> b = a
>>> b += 1 # Int Data type is Immutable.
>>> print(a) # Variable "a" will not change
>>> 1

>>> list_1 = ["a", "b", "c"]
>>> list_2 = list_1
>>> del(list_2[0]) # List is Mutable.
>>> print(list_1) # Also changes "list_1"
["b", "c"]
```

Python Data Structure – Mutable vs Immutable



https://medium.com/@299_42481/mutable-and-immutable-objects-in-python-c75cd3216461

Python Data Structure - Dictionaries

- Dictionary : 키(Key)와 값(Value)가 서로 연결되어 있는 순서가 없는 집합

```
# Key => "a" / Value => 1
# Key => "b" / Value => 2
dict_example_1 = {"a": 1, "b": 2}

# Key => 1 / Value => "a"
# Key => 2 / Value => "b"
dict_example_2 = {1: "a", 2: "b"}

# Get value by key
dict_example_1["a"] == 1 # True

# Change value by key
dict_example_1["a"] = 3
dict_example_1 == {"a": 3, "b": 2} # True
```

Python Data Structure – Dictionaries

- Dictionary 선언 : 중괄호({})를 사용하거나 명시적 선언 방법 사용 dict()

```
# 중괄호
dict_example = {}

# or 명시적 선언 dict()
dict_example = dict()
```

Python Data Structure – Nested Dictionaries

- List와 마찬가지로 Dictionary 안에 Dictionary가 들어 갈 수 있으며, List역시 Dictionary에 들어갈 수 있음

```
# Dict in Dict
nested_dict_1 = {"a": {"a-1": 1}, "b": {"b-1": 2}}
nested_dict_1["a"]["a-1"] == 1

# List in Dict
nested_dict_2 = {"a": [1, 2, 3], "b": [4, 6, 6]}
nested_dict_2["a"][0] == 1
```

Python Data Structure – Dictionaries Methods

- Dictionary Update : Dictionary의 값(Value)를 변경하는 방법
- 단일 수정 vs 다중 수정

```
dict_example = {"a": 1, "b": 2, "c": 3}

# 단일 수정은 키로 접근하여 값을 할당
dict_example["a"] = 4

# 다중 수정은 update 메서드 사용
dict_example.update({
    "a": 1,
    "b": 4
})

# 없는 Key를 update 할 경우 추가됨
dict_example.update({
    "d": 4
})
```

Python Data Structure – Dictionaries Methods

- .keys() : dictionary의 key들을 list 형태로 반환
- .values() : dictionary의 value들을 list 형태로 반환
- .items() : dictionary의 key-value 조합을 tuple의 list 형태로 반환

```
In [1]: dict_example = {"a": 1, "b": 2, "c": 3}

In [2]: dict_example.keys()
Out[2]: dict_keys(['a', 'b', 'c'])

In [3]: dict_example.values()
Out[3]: dict_values([1, 2, 3])

In [4]: dict_example.items()
Out[4]: dict_items([('a', 1), ('b', 2), ('c', 3)])
```

Python Data Structure - Tuples

- Tuple : List와 같이 순서가 있는 값들의 집합
- List와의 차이점
  - List는 [ ] 로 선언 하지만, Tuple은 ( )로 선언
  - List는 값을 수정, 삭제 가능 하지만 Tuple은 값을 변경 할 수 없다.

```
tuple_example = (1, 2, 3)   # Tuple 생성

tuple_example_2 = (1, "a", 3)   # List와 마찬가지로 다양한 자료형을 원소로 가질 수 있음
```

Python Data Structure - Tuples

- Tuple은 List와 유사하게 Index로 탐색 가능함

```
t1 = (1, 2, "a", "b")

t1[0] == 1
t1[3] == "b"
t1[0:2] == (1, 2)
```

Python Data Structure - Tuples

- Tuple : List와 같이 순서가 있는 값들의 집합
- List와의 차이점
  - List는 [ ] 로 선언 하지만, Tuple은 ( )로 선언
  - List는 값을 수정, 삭제 가능 하지만 Tuple은 값을 변경 할 수 없다.

```
tuple_example = (1, 2, 3)   # Tuple 생성

tuple_example_2 = (1, "a", 3)   # List와 마찬가지로 다양한 자료형을 원소로 가질 수 있음
```

Python Data Structure - Sets

- Set : Unique한 값들의 집합(수학의 집합 개념과 유사)
  - 순서가 없음(Unordered)
  - 중복된 값들을 허용하는 List, Tuple과 달리 유일한 값을 가져야함
  - Dictionary와
- Set 선언 : Dictionary와 동일하게 중괄호({})를 사용하거나 명시적 선언 방법 사용 set(**iterable)

```
# 중괄호
set_example = {1, 2, 3}

# or 명시적 선언 set(**iterable)
set_example = set([1, 2, 3])
```

Python Data Structure - Sets

- Set의 원소로 중복된 값을 넣을 경우 자동으로 제거됨

```
# 중복된 원소로 Sets 선언
set_example = {1, 2, 3, 3, 4, 4, 5, 5}

set_example == {1 ,2, 3, 4, 5}  # True
```

Python Data Structure – Sets Methods

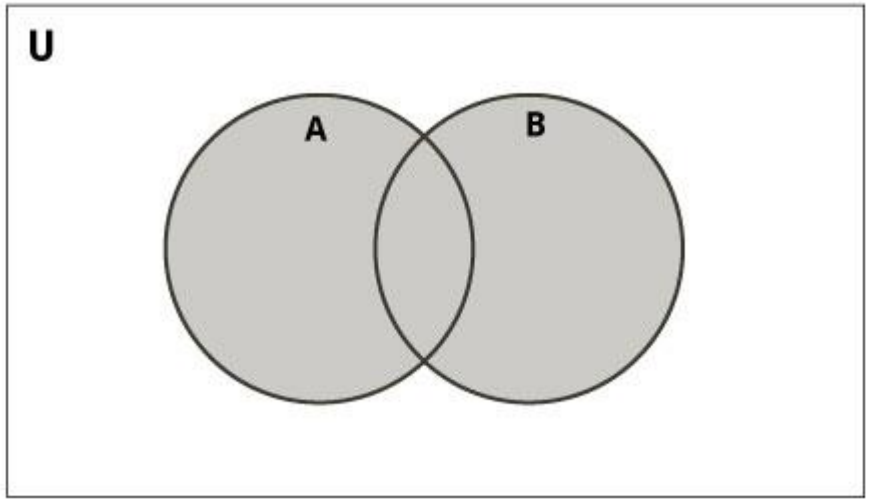
```
set_example = {1, 2}

# 원소 추가 : .add()
set_example.add(3)

# 여러개의 원소를 추가 : .update() // Dictionary의 Update와 다름
set_example.update({4, 5, 6})

# 원소 삭제 : .remove() // 해당 원소가 없을 경우 에러 발생
set_example.remove(4)

# 원소 삭제 : .discard() // 해당 원소가 없어도 에러가 발생하지 않음
set_example.remove(4)
```

Python Data Structure – Sets Operator

- Sets Operator : Sets 간의 집합 연산
  - ¦ : 합집합(Union) 연산자
  - & : 교집합(Intersection) 연산자
  - - : 차집합(Difference) 연산자
  - ^ : 대칭차집합(합집합 – 교집합)(Symmetric Difference) 연산자

Python Data Structure – Sets Operator

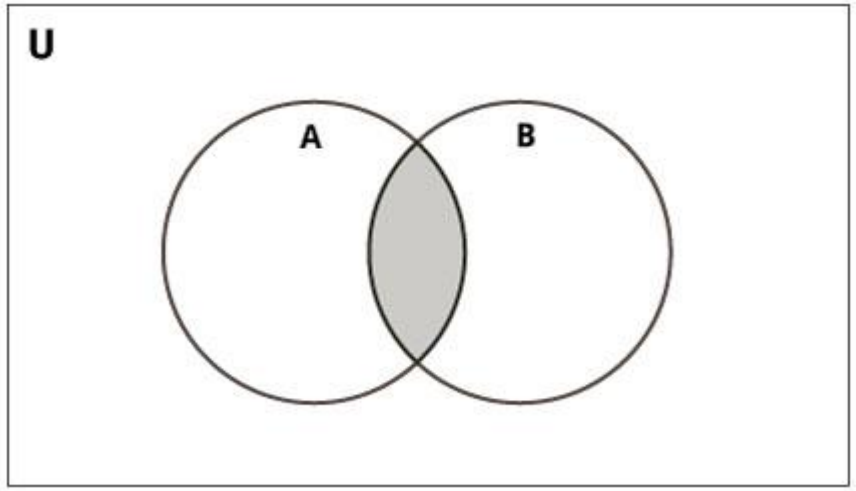
- Sets Operator : Sets 간의 집합 연산
  - ¦ : 합집합(Union) 연산자




```python
# Sets A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# "¦" operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A ¦ B)
```

Python Data Structure – Sets Operator

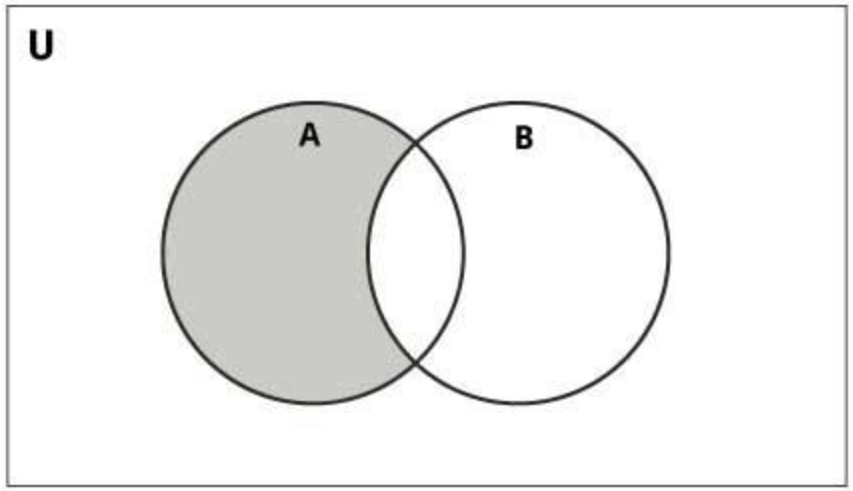
- Sets Operator : Sets 간의 집합 연산
  - & : 교집합(Intersection) 연산자




```python
# Sets A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# "&" operator
# Output: {4, 5}
print(A & B)
```

Python Data Structure – Sets Operator

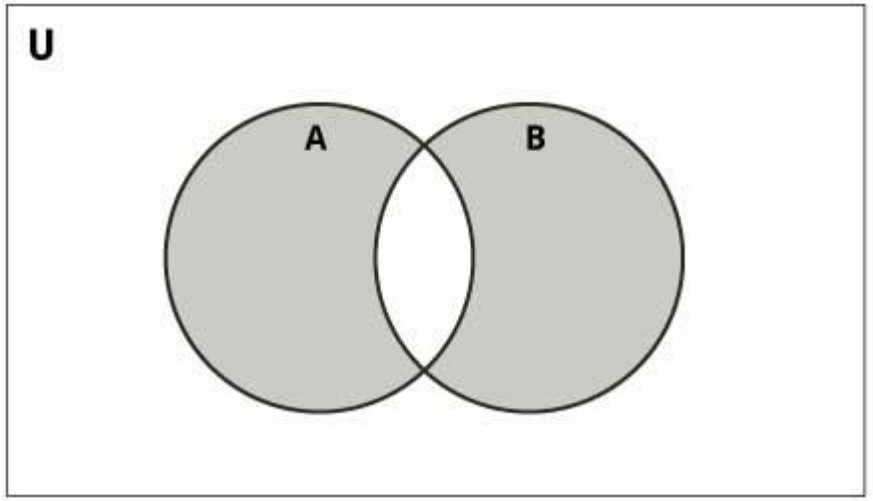
- Sets Operator : Sets 간의 집합 연산
  - - : 차집합(Difference) 연산자




```python
# Sets A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# "-" operator on A
# Output: {1, 2, 3}
print(A - B)
```

Python Data Structure – Sets Operator

- Sets Operator : Sets 간의 집합 연산
  - ^ : 대칭차집합(합집합 – 교집합)(Symmetric Difference) 연산자




```python
# Sets A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# "^" operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

Summary

- Definition of Python Lists, Dictionaries, Tuples, Sets
- Methods & Operators of each data type
- Iterator vs Iterable
- Mutable vs Immutable

Self Study Topic

- Mutable vs Immutable(Googling!)
- Deep Copy vs Shallow Copy(Googling!)
- Mutable/Immutable 과 Deep Copy/Shallow Copy 의 관계 (Googling!)