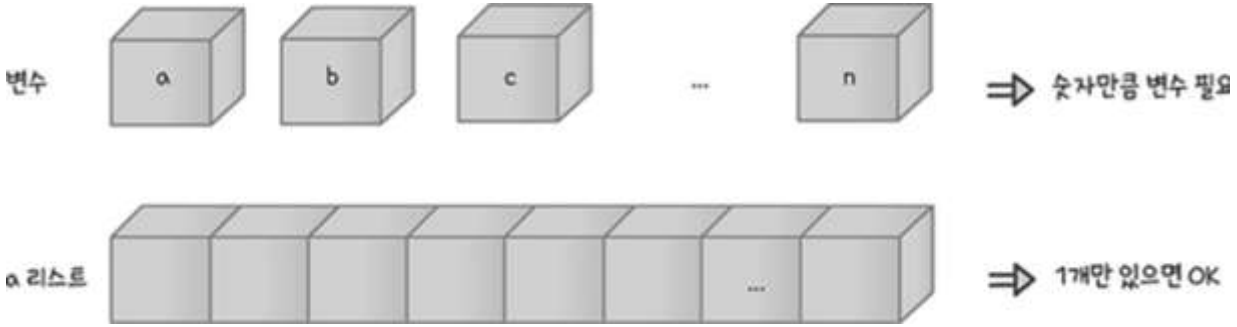
파이썬 자료형 - 리스트

김지성 강사

파이썬 자료형 - 리스트

✓ 리스트(list)는 자료들의 모임으로 **여러 변수를 사용하지 않아도 여러 개의 값을 담을 수 있다**.

✓ 만약 1~10까지의 값을 담으려고 한다면, 리스트가 없이는 총 10개의 변수를 담아야 한다. 이러한 불편함을 해결하기 위해서 사용되는게 바로 리스트(list)이다.



파이썬 자료형 - 리스트

✓ 리스트의 특징은 다음과 같다.

- ○ list는 자료들의 모임
- ○ 입력된 순서가 유지
- ○ list() 생성자 혹은 []로 리스트를 만듦
- ○ 1개의 데이터만으로도 리스트를 구성할 수 있음
- ○ 여러 개의 자료형도 저장이 가능
- ○ 가변 객체이므로 수정이 가능

```
list1 = [1, 2, 3, 4, 5]
print(type(list1), list1)
```

```
<class 'list'> [1, 2, 3, 4, 5]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 생성

○ 기본적으로 생성자 **list()**를 활용하는 방법과 **[]** 대괄호를 활용하여 리스트를 생성할 수 있음.

```
list1 = list()
print(type(list1), list1)

list2 = []
print(type(list2), list2)

list3 = [1, 2, 3, 4, 5]
print(type(list3), list3)

list4 = list("hello")
print(type(list4), list4)

list5 = [1, 2, 3, 1.1, 2.2, 'A', True,(1,2,3)]
print(type(list5), list5)
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 덧셈/곱셈

- ○ 리스트와 리스트를 더하면 값이 더해지는 것이 아닌 2개의 리스트가 1개로 합쳐진다.
- ○ 리스트를 곱하면 그 수만큼 리스트가 복사되어 늘어난다.

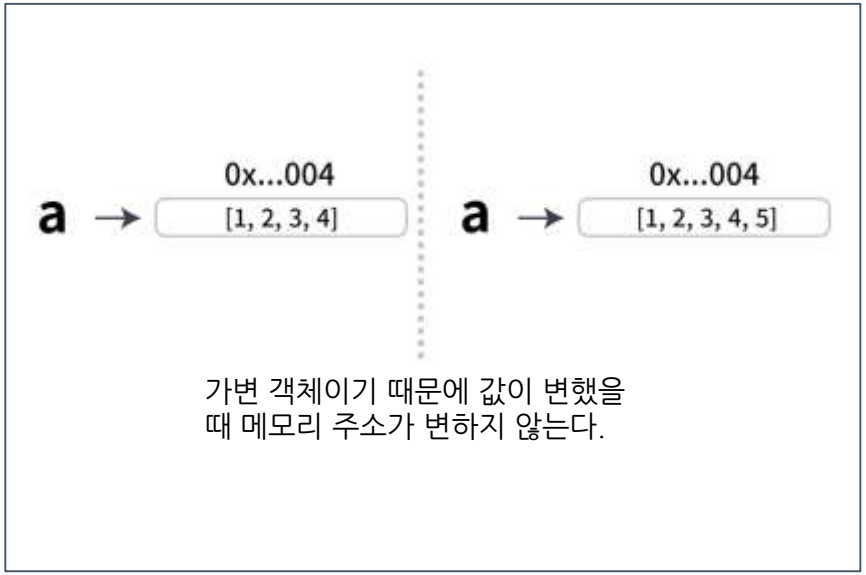
```
list_1 = [1, 2, 3, 4, 5]
list_2 = [10, 20, 30, 40, 50]
print(list_1+list_2)
print(list_1*2)
```

```
[1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 값 변경하기

- ○ 리스트는 가변 객체이기 때문에 값을 변경할 수 있다.
- ○ 이 때 값을 변경해도 메모리의 주소는 변함이 없음.



파이썬 자료형 - 리스트 메소드

✓ 리스트 값 변경하기

- ○ 리스트는 가변 객체이기 때문에 값을 변경할 수 있다.
- ○ 이 때 값을 변경해도 메모리의 주소는 변함이 없음.

```
list_1 = [1, 2, 3, 4, 5]
print(list_1, id(list1))
list_1[0]=10
print(list_1, id(list1))
```

```
[1, 2, 3, 4, 5] 2751466845952
[10, 2, 3, 4, 5] 2751466845952
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 크기 계산

○ len() 메소드를 사용하면 리스트안에 담긴 값의 개수를 알 수 있다.

```
list_1 = [1, 2, 3, 4, 5]
print(len(list_1))
```

```
5
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 추가

- append() 메소드를 사용하면 값을 추가할 수 있다.
- 이 때 추가된 값은 가장 뒤에 위치하게 된다.

```
list_1 = [1, 2, 3, 4, 5]
list_1.append(6)
print(list_1)
```

```
[1, 2, 3, 4, 5, 6]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 추가

○ extend() 메소드를 사용하면 원래 리스트의 형태를 유지한채로 값을 추가 할 수 있다.

○ 이 때 추가된 값은 가장 뒤에 위치하게 된다.

```
list_1 = [1, 2, 3, 4, 5]
list_2 = [10,20]
list_1.append(list_2)
print(list_1)

list_1 = [1, 2, 3, 4, 5]
list_2 = [10,20]
list_1.extend(list_2)
print(list_1)
```

```
[1, 2, 3, 4, 5, [10, 20]]
[1, 2, 3, 4, 5, 10, 20]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 삽입

○ insert('값을 넣을 인덱스', '값') 메소드를 사용하면 값을 원하는 위치에 추가할 수 있다.

```
list_1 = [1, 2, 3, 4, 5]
list_1.insert(3,100)
print(list_1)
```

```
[1, 2, 3, 100, 4, 5]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 삭제

- remove('삭제할 값') 메소드를 사용하면 해당되는 값을 제거할 수 있다.
- pop('인덱스') 메소드를 사용하면 인덱스에 해당하는 값을 삭제할 수 있다.
- clear() 메소드를 사용하면 리스트내에 있는 값을 전부 삭제할 수 있음.

```
list_1 = [1, 2, 3, 4, 5]
list_1.remove(3)
print(list_1)

list_1.pop()
print(list_1)
list_1.pop(1)
print(list_1)

list_1.clear()
print(list_1)
```

```
[1, 2, 4, 5]
[1, 2, 4]
[1, 4]
[]
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 정렬

- ○ sort() 메소드를 이용하여 정렬이 가능. 옵션으로 reverse=True 옵션을 주어 내림차순도 가능
- ○ sorted() 함수를 이용하여 정렬이 가능.

```
list_1 = [4, 3, 2, 1, 5]
#sort()
list_1.sort()
print("sort : ", list_1) # 원본이 변함

# reverse()
list_1 = [4, 3, 2, 1, 5]
list_1.sort(reverse=True)
print("reverse : ", list_1) # 원본이 변함

# sorted
list_1 = [4, 3, 2, 1, 5]
print(sorted(list_1))
print(list_1) # 원본은 변하지 않음
```

파이썬 자료형 - 리스트 메소드

✓ 리스트 값 찾기

- index('찾을 값') 메소드를 이용하여 찾고자 하는 값의 위치를 알 수 있다.
- in 연산자를 이용하여 값이 들어가있는지 확인할 수 있다.
- count('찾을 값') 메소드를 이용하면 해당 값이 리스트에 몇 번 나타나는지 알 수 있다.

```
list_1 = [10, 20, 30, 40, 50]
print(list_1.index(30))

print(10 in list_1)

list_1 = [10, 20, 30, 40, 50, 50, 50]
print(list_1.count(50))
```

파이썬 자료형 - 리스트

✓ 연습 문제1

○ 아래의 빈 리스트를 사용하여 다음 요구사항을 충족하는 코드를 작성하세요.

1. append()를 사용하여 'apple', 'banana', 'cherry'를 추가합니다.
2. 'banana'와 'cherry' 사이에 'orange'를 삽입합니다.
3. 리스트를 알파벳 순으로 정렬합니다.
4. 정렬된 리스트에서 'cherry'를 삭제합니다.
5. 최종 리스트를 출력하세요.

```
fruits = []
```

```
['apple', 'banana', 'orange']
```

파이썬 자료형 - 리스트

✓ 연습 문제2

○ 아래의 빈 리스트를 사용하여 다음 요구사항을 충족하는 코드를 작성하세요.

1. 리스트를 오름차순으로 정렬합니다.
2. 정렬된 리스트에 4를 올바른 위치에 삽입합니다.
3. 리스트의 마지막 요소를 삭제합니다.
4. 최종 리스트를 출력하세요.

```
student_numbers = [5, 3, 8, 1, 2]
```

파이썬 자료형 - 리스트

✓ 연습 문제3

○ 아래의 빈 리스트를 사용하여 다음 요구사항을 충족하는 코드를 작성하세요.

1. 리스트의 길이를 구하고 출력하세요.
2. 리스트의 첫 번째 값과 리스트 길이를 더한 값을 출력하세요.

```
numbers = [5, 10, 15, 20, 25]
```

```
5
10
```

파이썬 자료형 - 리스트

✓ 연습 문제4

○ 아래의 빈 리스트를 사용하여 다음 요구사항을 충족하는 코드를 작성하세요.

1. 'blue'가 처음 등장하는 위치(인덱스)를 출력하세요.
2. 'yellow'가 처음 등장하는 위치(인덱스)를 출력하세요.

```
colors = ['red', 'blue', 'green', 'yellow', 'blue']
```

1

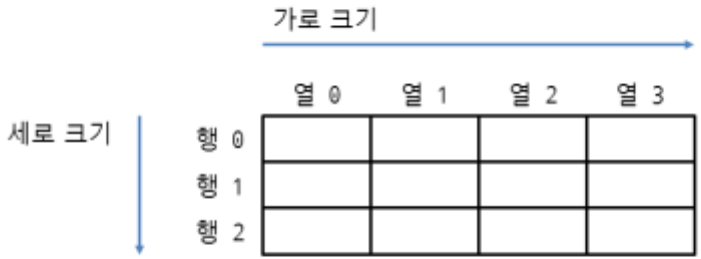
3

파이썬 자료형 - 다차원 리스트

김지성 강사

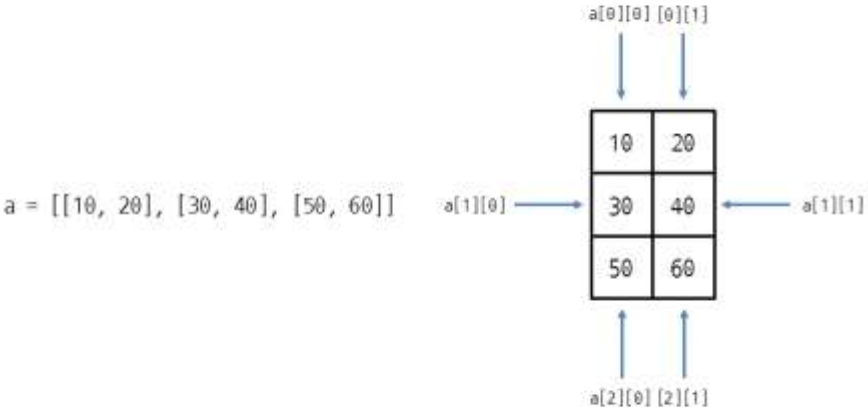
파이썬 자료형 - 리스트

- ✓ 다차원 리스트는 리스트안에 리스트가 있는 구조로 고차원 데이터를 다루는데 효과적이다.
- ✓ 2차원 리스트는 흔히 행렬로 표현이 되며 3차원, 4차원의 고차원 리스트 또한 구현이 가능하다.



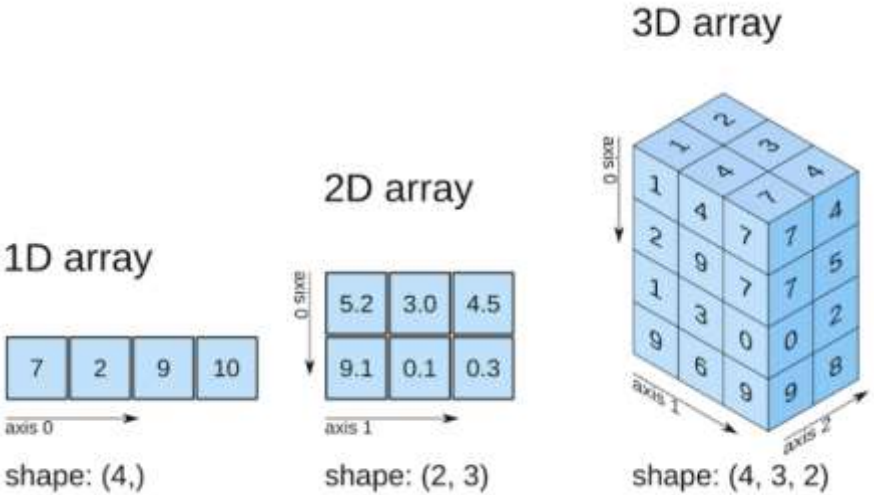
파이썬 자료형 - 리스트

```
a = [[10,20],[30,40],[50,60]]
print(a)
print(a[0][1])
```



파이썬 자료형 - 리스트

```
a = [[[10,20],[30,40],[50,60]],
     [[1,2],[3,4],[5,6]]]
print(a)
print(a[0][1][0])
```



파이썬 자료형 - 리스트 복사

김지성 강사

파이썬 자료형 - 리스트 할당과 복사

✓ 가변 객체는 값을 수정할 수 있으며, 값이 수정돼도 새로운 메모리가 할당되지 않는다.

✓ 불변 객체는 값을 수정할 수 없으며, 값이 수정되면 새로운 메모리에 그 값이 할당된다.

	데이터 타입
가변(mutable)	list, set, dict
불변(immutable)	int, float, bool, tuple, string, unicode

파이썬 자료형 - 리스트 할당과 복사

- ✓ a라는 변수에 리스트를 만들고 b=a로 리스트를 다른 변수에 할당하면 리스트는 두 개가 되는 것이 아니라 실제로 1개의 메모리에 존재한다. a와 b 모두 이 메모리를 가리키고 있을 뿐이다.
- ✓ 이러한 경우를 **할당**이라고 한다. 2개의 변수는 각각 메모리를 차지하고 있는 것이 아닌 동일한 메모리를 가리키고 있다.
- ✓ 즉 할당은 원본 그 자체이다.

```
a = [0, 0, 0, 0, 0]
b = a
print(id(a)==id(b))
```

파이썬 자료형 - 리스트 할당과 복사

✓ copy() 메소드를 사용하면 **얕은 복사(shallow copy)**가 된다.

✓ 얕은 복사(shallow copy)는 리스트 객체는 새롭게 생성하지만 그 내부 요소는 원본을 참조하고 있다.

```
a=[0,0,0,0,0]
b=a.copy()
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={a[0]==b[0]}")
```

```
a=[[0,0,0,0,0]]
b=a.copy()
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={a[0]==b[0]}")
```

파이썬 자료형 - 리스트 할당과 복사

✓ 하지만 얕은 복사된 b를 수정하면 **1차원 리스트에서는 원본이 변하지 않지만 2차원 리스트에서는 원본이 변경**된다. 메모리 주소 또한 1차원 리스트는 변경, 2차원 리스트는 동일하다.

```
a=[0,0,0,0,0]
b=a.copy()
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
b[0]=5
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
print(a)
print(b)
```

```
a==b=False / a[0]==b[0]=True
a==b=False / a[0]==b[0]=False
[0, 0, 0, 0, 0]
[5, 0, 0, 0, 0]
```

```
a=[[0,0,0,0,0]]
b=a.copy()
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
b[0][0]=5
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
print(a)
print(b)
```

```
a==b=False / a[0]==b[0]=True
a==b=False / a[0]==b[0]=True
[[5, 0, 0, 0, 0]]
[[5, 0, 0, 0, 0]]
```

파이썬 자료형 - 리스트 할당과 복사

- ✓ 이러한 현상이 일어나는 이유는 **1차원 리스트 내부에 있는 객체가 불변 객체**이기 때문이다. int type의 값이 들어가있는 내부 값은 **불변 객체이기 때문에 값의 수정이 일어났을 때 새로운 객체를 만들게된다**.
- ✓ 2차원 리스트에서의 수정은 **1차원 리스트 내부를 변경한 것이기 때문에 가변 객체인 list를 변경**했으므로 **새로운 객체를 만들지 않고 수정**한다.
- ✓ 그렇기 때문에 1차원 리스트를 조작할 때 우리는 뒤에서 배우게 될 깊은 복사(deep copy)처럼 보여질 수 있지만 얕은 복사로 진행되는 것이고 내부 값이 불변 객체의 특성 때문에 이러한 현상이 일어난다는 것을 알아두자.

파이썬 자료형 - 리스트 할당과 복사

✓ 얕은 복사로 복사된 변수 b를 수정하면 원본 a도 마찬가지로 변경된 것을 볼 수 있다.

```
a = [[0, 0, 0, 0, 0]]
b = a
print(id(a) == id(b))
print(id(a[0]) == id(b[0]))
b[0][0] = 99
print(a)
print(b)
```

```
True
True
[[99, 0, 0, 0, 0]]
[[99, 0, 0, 0, 0]]
```

```
a=[[0,0,0,0,0]]
b=a.copy()
print(id(a) == id(b))
print(id(a[0]) == id(b[0]))
b[0][0] = 99
print(a)
print(b)
```

```
False
True
[[99, 0, 0, 0, 0]]
[[99, 0, 0, 0, 0]]
```

파이썬 자료형 - 리스트 할당과 복사

✓ 진정한 복사를 하려면 copy 모듈안에 있는 deepcopy() 메소드를 활용해야 한다.

✓ 깊은 복사(deep copy)를 진행하면 복사된 b를 수정해도 원본인 a는 수정이 되지 않는다.

```
import copy
a=[[0,0,0,0,0]]
b=copy.deepcopy(a)
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
b[0][0] = 99
print(f"a==b={id(a)==id(b)} / a[0]==b[0]={id(a[0])==id(b[0])}")
print(a)
print(b)
```

파이썬 자료형 - 리스트 할당과 복사

✓ 즉 정리하자면, 할당/얕은 복사/깊은 복사가 존재한다.

- ○ 할당 : 원본 객체를 그대로 참조하는 것으로 원본 그 자체를 의미한다.
- ○ 얕은 복사(shallow copy) : 객체는 새로 만들지만 객체 내부 요소는 원본을 참조한다.
- ○ 깊은 복사(deep copy) : 객체와 내부 요소 자체를 새롭게 만들어낸다.

파이썬 자료형 - 튜플(Tuple)

김지성 강사

파이썬 자료형 - 튜플

- ✓ 튜플(tuple) 자료형은 리스트와 비슷하게 데이터의 목록이다.
- ✓ **콤마**(,) 혹은 **괄호()**로 데이터를 나열하면 튜플 자료형이 된다.
  - ○ ex) **a=1,2,3** or **a=(1,2,3)**
- ✓ 리스트와 동일하게 인덱스 슬라이싱을 통해 접근이 가능하고 + 연산을 통한 결합, * 연산을 통한 반복 또한 가능하다.
- ✓ **튜플은 불변 객체**로 값 변경 및 삭제가 불가능하다.

파이썬 자료형 - 튜플

✓ 리스트와 튜플은 서로 매우 비슷하다. 그럼 왜 튜플을 사용하는 것일까?

✓ 이유는 다음과 같다.

- **리스트에 비해 가볍다**.
  1. 동적 배열 할당으로 인해 리스트는 추가적인 메모리 사이즈를 할당하려면 더블링(doubling)을 자체적으로 진행한다.
- 불변 객체이기 때문에 **수정이 필요없는 데이터를 사용**해야 할 경우 **튜플이 훨씬 안정적**이다.

파이썬 자료형 - 튜플

✓ 튜플 실습

```
# tuple 만들기
tuple1 = 1, 2, 3, 4, 5
tuple2 = (6, 7, 8, 9, 10)
print(tuple1, type(tuple1))
print(tuple2, type(tuple2))

# 다음은 tuple이 아닙니다.
tuple3 = 1
tuple4 = (10)
print(tuple3, type(tuple3))
print(tuple4, type(tuple4))
```

파이썬 자료형 - 튜플

✓ 튜플 실습

```
# 다음은 tuple입니다.
tuple3 = 1,
tuple4 = (6,)
print(tuple3, type(tuple3))
print(tuple4, type(tuple4))

# 다른 자료형들의 모임
tuple5 = '하나', 123, True
print(tuple5, type(tuple5))

# tuple이 tuple을 가짐
tuple6 = 'id', ('kim','lee'),(33,25), (True, False)
print(tuple6, type(tuple6))
```

파이썬 자료형 - 튜플

✓ 튜플 실습

```
# tuple
tuple1 = 1, 2, 3
print("개수 : " + str(len(tuple1)))
print('*' * 50)

print(tuple1[0], tuple1[1],tuple1[2],tuple1[-1], type(tuple1[0]))
print(tuple1[0:1],type(tuple1[0:1]))
print(tuple1[1:3],type(tuple1[0:1]))
print('*' * 50)
```

파이썬 자료형 - 튜플

✓ 튜플 실습

```
tuple1 = 1, 2, 3
tuple2 = ('a', 'b', 'c', 'd')
tuple3 = tuple1 + tuple2
tuple4 = tuple2 * 3
print("tuple2",tuple2)
print("tuple3",tuple3)
print("tuple4",tuple4)
```

파이썬 자료형 - 튜플

✓ 튜플 실습

```
# Tuple은 불변 객체
tuple1 = 1, 2, 3, 4
print(tuple1, type(tuple1))

# 요소 값 변경
# tuple1[0] = 'A'

# 요소 값 삭제
# del tuple1[0]

# 첫번째 값을 변경하려면 : 새로운 객체를 만들어야 합니다.
tuple1 = ('A',) + tuple1[1:]
print(tuple1)

# 세번째 값을 변경하려면
tuple1 = tuple1[:2] + ('C',) + tuple1[3:]
print(tuple1)

# 두번째 값을 삭제하려면
tuple1 = (tuple1[0],) + tuple1[2:]
print(tuple1)
```

파이썬 자료형 - 집합(Set)

김지성 강사

파이썬 자료형 - 집합

- ✓ 집합(set) 자료형은 **중복을 허용하지 않는** 자료형이다.
- ✓ 입력된 순서 또한 집합(set) 자료형에서는 중요하지 않다. (**unordered**)
- ✓ **set() 생성자 함수**를 사용하거나 **{}**를 사용하여 만들 수 있다.
- ✓ 주로 **교집합**, **합집합**, **차집합을** 구할 때 유용하게 사용할 수 있다.

파이썬 자료형 - 집합

✓ 집합 실습

```
# set 자료형 생성
s1 = set([1, 2, 3])
s2 = {10,20,30}
print(s1, type(s1))
print(s2, type(s2))

# 집합 자료형 접근
s1 = set([1, 2, 3])
l1 = list(s1)
print(l1[:2])
```

파이썬 자료형 - 집합

```
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])
# 교집합
print("s1 & s2",s1 & s2)
print("s1.intersection(s2)",s1.intersection(s2))
print()

# 합집합
print("s1 | s2",s1 | s2)
print("s1.union(s2)",s1.union(s2))
print()

# 차집합
print("s1 - s2",s1 - s2)
print("s2 - s1",s2 - s1)
print()
print("s1.difference(s2)",s1.difference(s2))
print("s2.difference(s1)",s2.difference(s1))
```

파이썬 자료형 - 집합

```
# 값 추가
s1 = set([1, 2, 3])
s1.add(4)
print(s1)

# 값 수정
s1 = set([1, 2, 3])
s1.update([4, 5, 6])
print(s1)
{1, 2, 3, 4, 5, 6}

# 값 제거
s1 = set([1, 2, 3])
s1.remove(2)
print(s1)

s1 = set([1, 2, 3])
s1.discard(44)
print(s1)
```

파이썬 자료형 - 딕셔너리(Dictionary)

김지성 강사

파이썬 자료형 - 딕셔너리

- ✓ 딕셔너리(dictionary)는 “**키(key)/값(value)**”의 쌍을 요소로 갖는 자료형이다.
- ✓ 흔히 Map이라고도 불리우며 **키(key)를 가지고 값(value)을 찾아내는 해시테이블(Hash Table) 구조**임.
- ✓ **dict() 생성자**를 사용하거나 **{}**를 사용하여 딕셔너리를 만들 수 있음.
- ✓ **키(key)는 값을 변경할 수 없는 불변 타입**이어야 하고, **값(value)은 가변/불변 모두 가능**하다.
  - ○ 딕셔너리에서의 키는 튜플, 문자열은 가능하나 리스트는 가변이기 때문에 키로 사용 불가

파이썬 자료형 - 딕셔너리

✓ 딕셔너리 실습

```
# dictionary
dict1 = {}
print(dict1, type(dict1))

dict2 = {'name':'한사람','age':22, 'gender': True}
print(dict2, type(dict2))
print(dict2['name'])
print(dict2['age'])
print(dict2['gender'])

tuple1 = ('이름','나이','성별')
dict3 = {tuple1 : ('홍길동', 22, "남자")}
print(dict3, type(dict3))

print(dict3[tuple1])
```

파이썬 자료형 - 딕셔너리

✓ 딕셔너리(dictionary) 사용법

- keys() 메소드는 키(key)값을 얻을 수 있음.
- values() 메소드는 값(value)값을 얻을 수 있음.
- dict[key]=value 의 방식으로 새로운 아이템을 추가할 수 있음.
- update() 메소드를 사용하면 한 번에 여러 개의 아이템을 추가할 수 있음.
- 같은 키(key)에 다른 값을 넣으면 값(value)가 수정됨.
- del dict[key]로 아이템 삭제가 가능.
- items() 메소드로 (key,value)쌍을 얻을 수 있음.

파이썬 자료형 - 딕셔너리

✓ 딕셔너리 실습

```
# Dictionary(사전)란?
dict1 = {'name': '홍길동', 'age': 22, 'gender': '남자'}
print(dict1, type(dict1))

# 키
print('keys() 메소드 : ', dict1.keys())

# 값 반복
print('values() 메소드 : ', dict1.values())

# 추가
dict1['hobby'] = ['독서', '러닝', '취침']
dict1['weight'] = 85.123
print('추가된 딕셔너리 확인 : ', dict1)
```

파이썬 자료형 - 딕셔너리

✓ 딕셔너리 실습

```
# 동시에 여러요소 추가
dict1.update({'weight':66.66,'height': 183.3})
print('update() 메소드 : ', dict1)

# 수정
dict1['hobby'] = ['모임','게임','등산']
print('수정된 딕셔너리 확인 : ', dict1)

# 요소 삭제
del dict1['weight']
print('weight 요소 삭제 : ', dict1)

# (key,value)쌍 얻기
print('딕셔너리 아이템 확인 : \n', dict1.items())
```