03_Python(Sequence Type)

1. List

1.1 생성

```
a = list()  # 공백 리스트 생성
a = []      # 공백 리스트 생성
a = [1, 2, 3]   # 일반적인 리스트 생성
a = [1, 2, 3, "안녕", 3.14, False]    # 여러 타입 혼용 가능
a = [1, 2, [3, 4, 5], 6, 7]
```

1.2 연산

```
a = [1, 2, 3]
b = [4, 5, 6]
print(a + b)    # 연결[1, 2, 3, 4, 5, 6]
print(a * 3)    # [1, 2, 3, 1, 2, 3, 1, 2, 3]

a = [1, 2, 3]
a[0] = 5
print(a)        # [5, 2, 3]

a[0] = [9, 9, 9]
print(a)        # [[9, 9, 9], 2, 3]

a[0:1] = [9, 9, 9] # list안의 list가 아닌 list의 요소를 변경
print(a)        # [9, 9, 9, 2, 3]

a = [1, 2, 3, 4, 5, 6, 7]
# 위의 list를 1, 2, 6, 7 로 바꾸려면
a[2:5] = []
print(a)
```

1.3 함수

```
my_list = list([1, 2, 3])
```

- 요소 추가

  ```
  my_list.append(4)   # [1, 2, 3, 4]

  my_list.append([5, 6, 7])   # [1, 2, 3, 4, [5, 6, 7]]
  ```

- 리스트 확장(java의 addall과 비슷)

  ```
  my_list.extend([8, 9, 10])  # [1, 2, 3, 4, [5, 6, 7], 8, 9, 10]
  ```

- 정렬

```
my_list = [7, 3, 1, 8, 2]
my_list.sort()  # 기본적인 오름차순 정렬, 원본을 제어
my_list.reverse()   # 내림차순으로 하는 방법
```

- indexing

```
my_list.index(1)    # index()는 찾는 값의 위치 반환
```

2. Tuple

- Tuple은 List와 유사하지만, 표현 방법이 다르고, 수정, 삭제가 불가능하다.

2.1 생성

```
a = ()
a = (1, 2 ,3)       # [1, 2, 3]
a = (1,)            # [1]
                    # 요소가 1개 있을 때 tuple을 표현하려면
                    # a = (1,)
a = (1, 2, 3, 4)    # tuple
                    # tuple은 ()를 생략 가능
a = 1, 2, 3, 4
a, b, c = 10, 20, 30
print(a)            # 10
```

2.2 인덱싱

```
a = (1, 2, 3, 4)
print(a[1])     # 2
print(a[2:4])   # (3, 4)
```

2.3 연산

```
a = (1, 2, 3)
b = (5, 6, 7)
print(a + b)    # (1, 2, 3, 5, 6, 7)
```

2.4 List <-> Tuple

```
my_list = [1, 2, 3]
my_list = tuple(my_list)    # list를 tuple로 변환
print(my_list)

my_tuple = 10, 20, 30, 40
my_list = list(my_tuple)    # tuple을 list로 변환
print(my_list)
```

3. Range

- range는 **숫자 Sequence**로 주로 **for문에서 사용**

- 인자가 1개인 경우
  - 0부터 시작, 1씩 증가

```
my_range = range(10)
```

- 인자가 2개인 경우
  - 시작과 끝을 의미

```
my_range = range(10, 20)    # 10~19, 20은 제외
```

- 인자가 3개인 겨우
  - 시작, 끝, 증감을 의미

```
my_range(10, 20, 3) # 10, 13, 16, 19
```

4. Dict

- 표현법은 JSON표현과 유사 {"name" : "홍길동", "age" : 30}

```
my_dict = {"name" : "홍길동", "age" : 30}
print(type(my_dict))  #  <class 'dict'>
```

- 새로운 key : value를 추가할 경우

```
my_dict[100] = "홍길동"
print(my_dict)      # {'name': '홍길동', 'age': 30, 100: '홍길동'}
```

- 특정 key를 삭제할 경우

```
del my_dict["age"]
print(my_dict)      #  {'name': '홍길동', 100: '홍길동'}
```

- key값이 중복되는 경우

```
my_dict = {"name" : "홍길동", "age" : 30, "age" : 40}
print(my_dict)      # {'name': '홍길동', "age": 40}
```

- keys()

```
my_dict = {"name" : "홍길동", "age" : 30, "address" : "서울"}
print(my_dict.keys())
# 리턴값은 key값들의 리스트처럼 생긴 객체
# list와 유사하지만 list의 함수는 사용할 수 없다
# values() : dict의 value값들만 뽑음
# items() : (key, value)의 형태로 구성된 리스트처럼 생긴 객체를 리턴

my_dict = {"name" : "홍길동", "age" : 30, "address" : "서울"}
# for 문을 이용하여 모든 key와 그에 대한 value값을 출력
for key in y_dict.keys() :
    print("{0}, {1}".format(key, my_dict[key]))
```

5. Set

- set은 중복이 없고, 순서가 없다.

```
my_set = set([1, 2, 3]) # set 생성 => {1, 2, 3}
print(my_set)
my_set = set("Hello") # {'H', 'e', 'l', 'o'}
print(my_set)
```

- 기본적인 set연산(교집합, 합집합, 차집합)

```
s1 = {1, 2, 3, 4, 5}
s2 = {4, 5, 6, 7, 8}

print(s1 & s2)  # 교집합(intersection)
print(s1 | s2)  # 합집합(union)
print(s1 - s2)  # 차집합(differences)
```

- 기타 함수

```
my_set = {1, 2, 3, 4, 5}

# set에 새로운 요소를 추가하려면
my_set.add(10)

# set에 여러개를 추가하려면
my_set.update([7, 8, 9])

# set에서 삭제할 경우
my_set.remove(1)
```