[튜플, 딕셔너리, 집합]

- 튜플
- 딕셔너리
- 집합

튜플(tuple)

- 소괄호 () 안에 콤마(,)로 구분된 요소
- 수정이 불가능한 정보(주민등록번호, 학번 등)를 저장할 때 사용
- 튜플 만들기

```
>>> t1 = (1,2,3) # ( ) 속에 요소를 나열하여 만든다
>>> t1
(1, 2, 3)
>>> t2 = 1,2,3   # () 없이 요소를 나열해서 만들 수도 있다
>>> t2
(1, 2, 3)
>>> t3 = 1,      # 요소가 한 개인 경우 끝에 ,를 붙인다
>>> t3
(1, )
>>> t4 = tuple(); t5 = () # 빈 튜플 생성 방법
>>> t4
()
>>> t5
()
```

튜플의 특징

- 리스트와 비슷한 자료구조이지만 요소 값의 변경 불가(immutable)
- 바꿀 수 없는 객체들을 담을 때 사용. 요소값 삭제, 변경 불가

```
>>> t1=(1,2,3)
>>> del t1[0]        # 요소값 삭제 불가
File "<python-input-14>", line 1, in <module>
    del t1[0]
        ~~^^^
TypeError: 'tuple' object doesn't support item deletion
>>> t1[0]=0          # 요소값 변경 불가
File "<python-input-15>", line 1, in <module>
    t1[0]=0
    ~~^^^
TypeError: 'tuple' object does not support item assignment
```

- 튜플 자체를 삭제할 때는 del() 함수 사용 : `del(t1)`

튜플의 연산

```
>>> t1 = (1,2,3)
>>> t1[1]          # 인덱싱
2
>>> t1[1:3]        # 슬라이싱
(2,3)

>>> t2 = ('a','b')
>>> t1 + t2        # 두 개의 튜플을 하나로 합치기
(1, 2, 3,'a','b')

>>> t1*2           # 동일한 튜플을 하나 더 생성하여 합치기
(1,2,3,1,2,3)
```

튜플과 리스트의 상호 변환

```
>>> T1=(1,2,3)
>>> L1=list(T1) # 튜플을 리스트로 변환
>>> L1
[1, 2, 3]

>>> L1.append(4)
>>> L1
[1, 2, 3, 4]

>>> T1=tuple(L1) # 리스트를 튜플로 변환
>>> T1
(1, 2, 3, 4)
```

튜플의 패킹(packing)과 언패킹(unpacking)

- 패킹(packing) : 괄호 () 속에 요소를 나열해 튜플을 생성
- 언패킹(unpacking) : 튜플의 요소를 하나씩 변수에 할당

```
>>> T = (1, 2, 3)     # 패킹 : 괄호 () 속에 요소를 나열해 튜플을 생성

>>> a, b, c = T       # 언패킹 : 튜플의 요소를 하나씩 변수에 할당
>>> a
1
>>> b
2
>>> c
3
```

튜플의 메소드

- 튜플의 요소의 수를 세는 메소드는 count()

```
>>> t = (1, 2, 3, 4, 3)
>>> t.count(3) # 요소 3이 2번 나타남
2
```

- 위치를 알 수 있는 메소드는 index()인데, 같은 값이 2번 이상 있을 경우 처음 요소의 위치값을 반환

```
>>> t = (1, 2, 3, 2, 5)
>>> t.index(2)
1
```

딕셔너리(dictionary, 사전) 자료형

- 찾으려고 하는 단어(키, key)를 주면 대응되는 값(value)을 알 수 있는 자료구조
  (예: 사전, 전화번호부)
- 쉼표 기호(,)로 구분되는 이 키(key):값(value)의 쌍이 순서 없이 나열되어 중괄호{ }로 묶임
- 딕셔너리를 만들 때는 중괄호 {}를 사용하여 다음과 같이 키와 값의 쌍으로 구성

```
>>> H = {'Kim':180, 'Lee':175, 'Park':170}
>>> H
{'Kim': 180, 'Lee': 175, 'Park': 170}
```

딕셔너리 생성

- dict() 함수를 이용해 빈 딕셔너리를 만들 수도 있고

```
>>> test = dict() # 빈 딕셔너리의 생성
>>> test
{}
>>> type(test)
<class 'dict'>
```

딕셔너리로 변환

- 여러 가지 자료구조를 dict() 함수를 이용해 다음과 같이 딕셔너리로 변환할 수 있음

```
>>> ll = [['a',0],['b',1],['c',2]] # 리스트의 리스트를 딕셔너리로 변환
>>> dict(ll)
{'a': 0, 'b': 1, 'c': 2}
>>> lt = [('a',0),('b',1),('c',2)] # 요소가 튜플인 리스트를 딕셔너리로 변환
>>> dict(lt)
{'a': 0, 'b': 1, 'c': 2}
>>> tl = (['a',0],['b',1],['c',2]) # 요소가 리스트인 튜플을 딕셔너리로 변환
>>> dict(tl)
{'a': 0, 'b': 1, 'c': 2}
>>> sl = ['a0', 'b1', 'c2']        # 두 글자 문자열 리스트를 딕셔너리로 변환
>>> dict(sl)
{'a': '0', 'b': '1', 'c': '2'}
>>> st = ('a0', 'b1', 'c2')        # 두 글자 문자열 튜플을 딕셔너리로 변환
>>> dict(st)
{'a': '0', 'b': '1', 'c': '2'}
```

딕셔너리 사용

- 특정한 키 값이 딕셔너리에 존재하는 지 알고 싶으면 다음과 같이 in 키워드를 사용 확인

```
>>> H = {'Kim':180, 'Lee':175, 'Park':170}
>>> 'Kim' in H
True
```

- 특정한 키에 대응되는 값을 알고 싶으면 다음과 같이 대괄호 안에 '키' 값을 주어 확인

```
>>> H['Kim']
180
```

- 딕셔너리의 `키` 로 사용할 수 있는 것은 정수, 실수, 문자열, 튜플 등

딕셔너리 관련 함수와 메소드

- del(), get(), pop(), clear()

```
>>> H = {'Kim':180, 'Lee':175, 'Park':170}
>>> H.get('Kim')  # 특정 요소의 값에 접근하려면 get() 메소드
180
>>> del(H['Lee']) # 딕셔너리의 쌍을 삭제하려면 del(딕셔너리이름[키]) 함수 사용
>>> H
{'Kim': 180, 'Park': 170}
>>> H.pop('Kim')  # 또는 특정 요소를 삭제하려면 pop() 메소드에 키값을 주어 사용
180
>>> H
{'Park': 170}
>>> H.clear()     # 딕셔너리를 초기화 하려면 clear() 메소드를 사용. 빈 딕셔너리로 만듬
>>> H
{}
```

dict_ 데이터 리스트

```
>>> H = {'Kim':180, 'Lee':175, 'Park':170}
>>> H.keys()       #  딕셔너리 변수의 키를 출력하려면 keys( ) 함수
dict_keys(['Kim', 'Lee', 'Park'])
>>> H.values()     # 값을 출력하려면 values( ) 함수
dict_values([180, 175, 170])
>>> H.items()      # 키-값 쌍을 출력하려면 items( ) 함수
dict_items([('Kim', 180), ('Lee', 175), ('Park', 170)])

>>> type(H.keys())
<class 'dict_keys'>
>>> type(H.values())
<class 'dict_values'>
>>> type(H.items())
<class 'dict_items'>
```

- 앞의 dict_로 시작하는 데이터들을 list() 함수를 이용해 각각 리스트로 변환 가능

```
>>> L1 = list(H.keys())
>>> L2 = list(H.values())
>>> L3 = list(H.items())
>>> L1
['Kim', 'Lee', 'Park']
>>> L2
[180, 175, 170]
>>> L3
[('Kim', 180), ('Lee', 175), ('Park', 170)]
```

딕셔너리 함축

- 딕셔너리 함축은 앞의 리스트 함축과 유사
- 숫자 1부터 4 까지의 수의 제곱에 해당되는 수를 표시하는 딕셔너리를 생성

```
>>> dic = {str(n)+'의 제곱' : n*n for n in range(1,5)}
>>> dic
{'1의 제곱': 1, '2의 제곱': 4, '3의 제곱': 9, '4의 제곱': 16}
```

집합(set, 세트) 자료형

- 중괄호 안에 요소들이 중복되지 않고, 순서없이 모여 있는 것
- 키만 모아 놓은 딕셔너리의 특수한 형태로 생각
- 세트를 생성하려면 딕셔너리처럼 중괄호를 사용하지만 콜론(:)없이 값을 입력
- 집합은 포함된 요소들의 순서나 중복에 상관없이 묶음 자체를 필요로 할 때 주로 사용
- 이 자료형은 집합의 연산 등에 사용
- 요소의 중복이 허용되지 않는 특징을 이용해 리스트 또는 튜플의 중복 제거에 사용되기도 함

집합의 연산

- 교집합(공통집합), 합집합, 차집합 연산이 가능

```
>>> s1={1,2,3}
>>> s2={2,3,4}
>>> s1 & s2                 # 교집합
{2, 3}
>>> s1.intersection(s2) # 이것도 교집합. s2.intersection(s1)도 동일
{2, 3}
>>> s1 | s2                 # 합집합
{1, 2, 3, 4}
>>> s1.union(s2)            # 이것도 합집합. s2.union(s1)도 동일
{1, 2, 3, 4}
>>> s1 - s2                 # 차집합  s1.difference(s2)도 동일
{1}
>>> s2 - s1                 # 차집합  s2.difference(s1)과 동일
{4}
```

집합의 함수

- 빈 집합을 만들때 set() 함수를 이용

```
>>> S = set(); S
set()
```

- set() 함수와 리스트, 튜플, 딕셔너리를 이용해 집합을 만들 수 있음

```
>>> s1 = set([1,2,3]); s1
{1, 2, 3}
>>> s2 = set((1,2,3)); s2
{1, 2, 3}
>>> s3 = set({'a':1, 'b':2}); s3
{'b', 'a'}
```

- set의 항목의 갯수를 셀 때는 len() 함수 사용

```
>>> len(s2)
3
```

집합의 메소드

- 요소 하나를 추가할 때는 add(), 여러 요소를 추가하려면 update() 사용
- 요소 제거 시 discard() 또는 remove() 사용(없는 요소를 remove() 이용해 제거하려 하면 에러)
- 세트의 모든 요소를 제거할 때는 clear() 이용

```
>>> N = set()
>>> N.add(1) # 요소 하나를 추가
>>> N
{1}
>>> N.update([2,3]) # 여러 요소를 추가
>>> N
{1, 2, 3}
>>> N.remove(3)
>>> N
{1, 2}
>>> N.clear()
>>> N
set()
```

Set Comprehension(세트 함축)

- 리스트 함축과 딕셔너리 함축과 유사
- 다음과 같이 사용하면 숫자 1부터 8까지의 수 중 짝수의 집합을 표시

```
>>> even = { x for x in range(1, 9) if x%2 == 0 }
>>> even
{8, 2, 4, 6}
```