Chapter 7 – 자료 구조(Data Structure)

데이터 구조(Data Structure)

-메모리상에 **데이터를 효율적으로 관리**하는 방법
-파이썬 에서는 **리스트, 튜플, 집합, 사전** 등의 기본 데이터 구조를 제공함

Stact(스텍)
-나중에 넣은 데이터를 먼저 반환하도록 설계된 메모리 구조 **Last In First Out (LIFO)**
-Data의 입력을 **Push**, 출력을 **Pop**이라고 함

```
>>> a = [1,2,3,4,5]
>>> a.append(10)
>>> a.append(20)
>>> a.pop()          # 20 출력
20
>>> a.pop()          # 10 출력
10
```

append – 끝에 값을 추가, pop – 끝에서 값을 제거

Queue(큐)
-먼저 넣은 데이터를 먼저 반환하도록 설계된 메모리 구조 **First in First out (FIFO)**
-stact과 반대되는 개념
-데이터 입력 **put,** 데이터 추출 **get**

```
>>> a = [1,2,3,4,5]
>>> a.append(10)
>>> a.append(20)
>>> a.pop(0) # 1 출력
1
>>> a.pop(0) # 2 출력
2
```

append – 값을 추가, pop(0) 값을 추출

Tuple(튜플)
-값이 변경이 불가능한 리스트
-선언 시 "[]" 가 아닌 "()"를 사용
t = (1) 일반정수(int)로 인식, 따라서 t = (1,) tuple 선언 시 ','값을 붙여준다.

Set(집합)
-값을 **순서없이** 저장, 중복 불허 하는 자료형
Ex) s= set(1,2,3,1,2,3) -> s {1,2,3}

Dictionary(사전)

- 데이터를 저장 할 때는 구분 지을 수 있는 값을 함께 저장
- 구분을 위한 데이터 고유 값을 key라고 한다.
- key와 value를 매칭하여 key로 value를 검색
- {key1: Value1, key: Value2, key: Value3……}

Collections

- List, Tuple ,Dict에 대한 Python Built-in 확장 자료 구조(모듈)
- 편의성, 실행 효율 등을 사용자에게 제공함

Deque

- **Stack** 과 **Queue**를 둘다 지원하는 모듈, List에 비해 효율적인 자료 저장 방식을 지원함
- rotate(기존 데이터를 두칸식 이동), reverse(역순)등 Linked List의 특성을 지원함
- 기존 list 형태의 함수를 모두 지원함, 속도가 빠르다.

Ordered Dict

- Dict와 달리, 데이터를 입력한 순서대로 dict를 반환함

```
for k, v in OrderedDict(sorted(d.items(), key=lambda t: t[0])).items():
    print(k, v)
```

```
l 500
x 100
y 200
z 300
```

```
for k, v in OrderedDict(sorted(d.items(), key=lambda t: t[1])).items():
    print(k, v)
```

```
x 100
y 200
z 300
l 500
```

t[0]은 앞에 알파벳 오름차순 ,t[1]은 값의 오름차순,

```
for k, v in OrderedDict(sorted(d.items(), reverser = True, key= lambda t : t[1])).items():
        print(k, v)
```

reverse 명령어를 추가하면 **내림차순**으로 변경

Default Dict

- Dict type의 값에 기본 값을 지정, 신규값 생성시에 사용하는 방법

```
from collections import defaultdict
d = defaultdict(object)     # Default dictionary를 생성
d = defaultdict(lambda: 0)  # Default 값을 0으로 설정합
print(d["first"])
```

Counter

- Sequence type의 data element들의 개수를 dict 형태로 반환

```
from collections import Counter

c = Counter()                           # a new, empty counter
c = Counter('gallahad')                 # a new counter from an iterable
print(c) Counter({'a': 3, 'l': 2, 'g': 1, 'd': 1, 'h': 1})
```

- Dict type, keyword parameter 등도 모두 처리 가능

```
c = Counter({'red': 4, 'blue': 2})      # a new counter from a mapping
print(c)
print(list(c.elements()))
```

```
Counter({'red': 4, 'blue': 2})
['blue', 'blue', 'red', 'red', 'red', 'red']
```

```
c = Counter(cats=4, dogs=8)             # a new counter from keyword args
print(c)
print(list(c.elements()))
```

```
Counter({'dogs': 8, 'cats': 4})
['dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'cats', 'cats', 'cats', 'cats']
```

namedtuple

- Tuple 형태로 Data 구조체(이름을 지정해서 저장하는 방법)를 저장하는 방법

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p[0] + p[1])
```

```
x, y = p
print(x, y)
print(p.x + p.y)
print(Point(x=11, y=22))
```