# 05
# collections 모듈

# 05. collections 모듈

- collections 모듈은 이미 앞에서 배운 다양한 자료구조인 리스트, 튜플, 딕셔너리 등을 확장하여 제작된 파이썬의 내장 모듈이다.
- collections 모듈은 deque, OrderedDict, defaultdict, Counter, namedtuple 등을 제공하며, 각 자료구조를 호출하는 코드는 다음과 같다.

```
from collections import deque
from collections import OrderedDict
from collections import defaultdict
from collections import Counter
from collections import namedtuple
```

# *dequeue(Double Ended Queue, 데크)란?*

- 양방향에서 데이터를 넣거나(append, leftappend) 빼는 것(pop, popleft)이 가능한 queue
  - .append(item): 우측에 추가 하기
  - .appendleft(item): 좌측에 추가하기
  - .pop(): 우측에서 빼서 반환하기. 오른쪽의 끝 값을 가져오면서 deque에서 제거
  - .popleft(): 좌측에서 빼서 반환하기. 왼쪽의 끝 값을 가져오면서 deque에서 제거
- 그 외 다수의 메소드를 지원한다.
- deque는 stack & queue는 물론 여타의 자료 형을 대부분 지원한다.

# deque.method

예제 z_deque.py

- d.append(item) - 우측에 1개의 자료를 추가
- d.appendleft(item) - 좌측에 1개의 자료를 추가
- a=d.pop() - 우측에서 1개 자료를 인출하고 해당 자료를 deque에서 삭제
- a=d.popleft() - 좌측에서 1개 자료를 인출하고 해당 자료를 deque에서 삭제
- d.extend([items]) - 우측에 list, tuple로 묶인 여러 개의 자료를 추가
- d.extendleft([items]) - 좌측에 list, tuple로 묶인 여러 개의 자료를 추가
- d.rotate(n) - 우측으로 원소들을 n회 회전. -n은 좌측으로..
- d.reverse() - 원소의 배열 순서를 역순으로 바꿈.
- d.remove(item) - 지료에서 지정된 값(item)을 삭제

# 05. collections 모듈

## deque 모듈

- deque 모듈은 스택과 큐를 모두 지원하는 모듈이다.
- deque 모듈을 사용하기 위해서는 리스트와 비슷한 형식으로 데이터를 저장해야 한다. 먼저 append( ) 함수를 사용하면 기존 리스트처럼 데이터가 인덱스 번호를 늘리면서 쌓이기 시작한다. 다음 코드를 확인하자.

```
>>> from collections import deque
>>>
>>> deque_list = deque()
>>> for i in range(5):
...     deque_list.append(i)
...
>>> print(deque_list)
deque([0, 1, 2, 3, 4])
```

# 05. collections 모듈

## deque 모듈

- 여기서 다음 코드와 같이 deque_list.pop()을 작성하면, 오른쪽 요소부터 하나씩 추출된다. 즉, 스택처럼 나중에 넣은 값부터 하나씩 추출할 수 있다

```
>>> deque_list.pop()
4
>>> deque_list.pop()
3
>>> deque_list.pop()
2
>>> deque_list
deque([0, 1])
```

# 05. collections 모듈

## deque 모듈

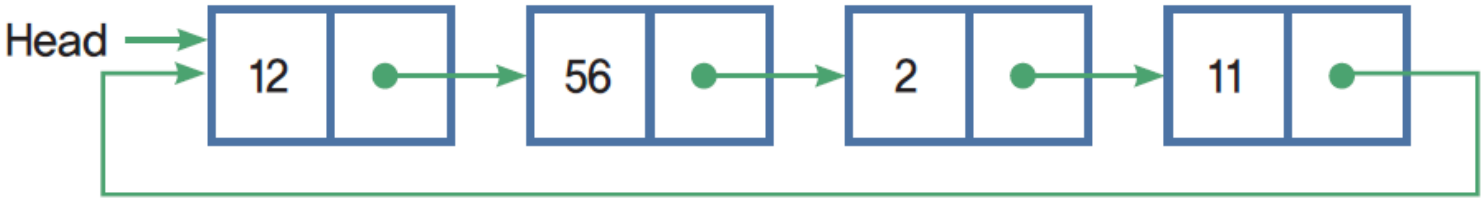
- 그렇다면 deque에서 큐는 어떻게 사용할 수 있을까? pop(0)을 입력하면 실행될 것 같지만, 이 함수는 deque에서 작동하지 않는다. 대신 deque는 appendleft( ) 함수로 새로운 값을 왼쪽부터 입력되게 하여 먼저 들어간 값부터 출력될 수 있도록 할 수 있다.

```
>>> from collections import deque
>>>
>>> deque_list = deque()
>>> for i in range(5):
...     deque_list.appendleft(i)
...
>>> print(deque_list)
deque([4, 3, 2, 1, 0])
```

# 05. collections 모듈

## deque 모듈

- **deque 모듈의 장점 :** deque는 연결 리스트의 특성을 지원한다. 연결 리스트는 데이터를 저장할 때 요소의 값을 한 쪽으로 연결한 후, 요소의 다음 값의 주소값을 저장하여 데이터를 연결하는 기법이다.

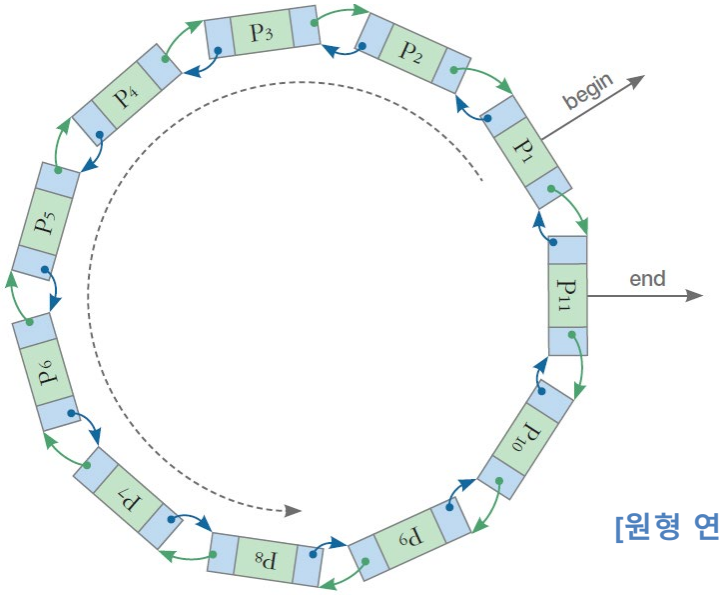



[연결 리스트의 형태]

# 05. collections 모듈

## deque 모듈

- 연결 리스트는 다음 요소의 주소값을 저장하므로 데이터를 원형으로 저장할 수 있다. 또한, 마지막 요소에 첫 번째 값의 주소를 저장한다면 해당 값을 찾아갈 수 있다. 이러한 특징 때문에 가능한 기능 중 하나가 rotate( ) 함수이다. rotate( )는 기존 deque에 저장된 요소들의 값 인덱스를 바꾸는 기법이다. 연결 리스트는 양쪽 끝의 요소들을 연결할 수 있으므로 원형의 데이터 구조를 가질 수 있다. 이러한 특징을 이용하여 각 요소의 인덱스 번호를 하나씩 옮긴다면, 실제로 요소를 옮기지 않더라도 인덱스 번호를 바꿀 수 있다.




[원형 연결 리스트의 형태]

# 05. collections 모듈

## deque 모듈

- 다음 코드를 살펴보면, 기존 데이터에 rotate(2) 함수를 입력하니 3과 4의 값이 두 칸씩 이동하여 0번째, 1번째 인덱스로 옮겨진 것을 확인할 수 있다. 다시 rotate(2)를 사용하면, 1과 2가 0번째, 1번째 인덱스로 이동한다.

```
>>> from collections import deque
>>>
>>> deque_list = deque()
>>> for i in range(5):
...     deque_list.appendleft(i)
...
>>> print(deque_list)
deque([0, 1, 2, 3, 4])
>>> deque_list.rotate(2)
>>> print(deque_list)
deque([3, 4, 0, 1, 2])
>>> deque_list.rotate(2)
>>> print(deque_list)
deque([1, 2, 3, 4, 0])
```

# 05. collections 모듈

## deque 모듈

- deque 모듈은 reversed( ) 함수를 사용하여 기존과 반대로 데이터를 저장할 수 있다.

```
>>> print(deque(reversed(deque_list)))
deque([0, 4, 3, 2, 1])
```

- deque 모듈은 기존의 리스트에서 지원하는 함수도 지원한다. extend( )나 extendleft( ) 함수를 사용하면, 리스트가 통째로 오른쪽이나 왼쪽으로 추가된다

```
>>> deque_list.extend([5, 6, 7])
>>> print(deque_list)
deque([1, 2, 3, 4, 0, 5, 6, 7])
>>> deque_list.extendleft([5, 6, 7])
>>> print(deque_list)
deque([7, 6, 5, 1, 2, 3, 4, 0, 5, 6, 7])
```

# 05. collections 모듈

## OrderedDict 모듈

- OrderedDict 모듈은 이름 그대로 순서를 가진 딕셔너리 객체이다. 딕셔너리 파일을 저장하면 키는 저장 순서와 상관없이 저장된다.

코드 7-2 ordereddict1.py

```
1 d = {}
2 d['x'] = 100
3 d['l'] = 500
4 d['y'] = 200
5 d['z'] = 300
6
7 for k, v in d.items():
8     print(k, v)
```

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

# 05. collections 모듈

## OrderedDict 모듈

코드 7-3 ordereddict2.py

```
1  from collections import OrderedDict      # OrderedDict 모듈 선언
2
3  d = OrderedDict()
4  d['x'] = 100
5  d['y'] = 200
6  d['z'] = 300
7  d['l'] = 500
8
9  for k, v in d.items():
10     print(k, v)
```

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

# 05. collections 모듈

## OrderedDict 모듈

코드 7-4 ordereddict3.py

```
def sort_by_key(t):
    return t[0]

from collections import OrderedDict        # OrderedDict 모듈 선언

d = dict()
d['x'] = 100
d['y'] = 200
d['z'] = 300
d['l'] = 500

for k, v in OrderedDict(sorted(d.items(), key=sort_by_key)).items():
    print(k, v)
```

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

# 05. collections 모듈

## OrderedDict 모듈 : [코드 7-4] 해석

- [코드 7-4]를 보면 딕셔너리의 값인 변수 d를 리스트 형태로 만든 다음, sorted( ) 함수를 사용하여 정렬한다. sorted(d.items(), key=sort_by_key)의 코드만 따로 실행하면 다음처럼 정렬되어 이차원 형태로 출력되는 값을 확인할 수 있다.

```
[('l', 500), ('x', 100), ('y', 200), ('z', 300)]
```

- 값을 기준으로 정렬한다면 [코드 7-4]의 1행과 2행을 다음처럼 바꾸면 된다. 참고로 t[0]과 t[1]은 위 리스트 안의 튜플 값 중 0번째 인덱스(l, x, y, z)와 1번째 인덱스(500, 100, 200, 300)를 뜻한다.

```
def sort_by_value(t):
    return t[1]
```

# 05. collections 모듈

## defaultdict 모듈

- defaultdict 모듈은 딕셔너리의 변수를 생성할 때 키에 기본 값을 지정하는 방법이다.

➡ 실제 딕셔너리에서는 [코드7 -5]처럼 키를 생성하지 않고 해당 키의 값을 호출하려고 할 때, 오류가 발생한다. 즉, 코드에서 first의 키 값을 별도로 생성하지 않은 채 바로 호출하여 오류가 발생하였다.

코드 7-5 defaultdict1.py

```
1 d = dict()
2 print(d["first"])
```

```
Traceback (most recent call last):
    File "defaultdict1.py", line 2, in <module>
        print(d["first"])
KeyError: 'first'
```

# 05. collections 모듈

## defaultdict 모듈

- 그렇다면 defaultdict 모듈은 어떻게 작동할까?

코드 7-6 defaultdict2.py

```
1 from collections import defaultdict
2
3 d = defaultdict(lambda: 0)                    # Default 값을 0으로 설정
4 print(d["first"])
```

```
0
```

➡ 핵심은 3행의 d = defaultdict(lambda: 0)이다. defaultdict 모듈을 선언하면서 초깃값을 0으로 설정한 것이다. 현재 lambda( ) 함수를 배우지 않아 코드를 정확히 이해하기 어렵겠지만, 'return 0'이라고 이해하면 된다. 어떤 키가 들어오더라도 처음 값은 전부 0으로 설정한다는 뜻이다.

# 05. collections 모듈

참고 z_dict2.py

## defaultdict 모듈

- defaultdict의 초깃값은 [코드 7-7]처럼 리스트 형태로도 설정할 수 있다.

코드 7-7 defaultdict3.py

```
1 from collections import defaultdict
2
3 s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
4 d = defaultdict(list)
5 for k, v in s:
6     d[k].append(v)
7
8 print(d.items())
9 [('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

```
dict_items([('yellow', [1, 3]), ('blue', [2, 4]), ('red', [1])])
```

# 05. collections 모듈

예제 z_Counter.py

## Counter 모듈

- Counter 모듈은 시퀀스 자료형의 데이터 요소 개수를 딕셔너리 형태로 반환하는 자료구조이다. 즉, 리스트나 문자열과 같은 시퀀스 자료형 안의 요소 중 값이 같은 것이 몇 개 있는지 반환해 준다.

```
>>> from collections import Counter
>>>
>>> text = list("gallahad")
>>> text
['g', 'a', 'l', 'l', 'a', 'h', 'a', 'd']
>>> c = Counter(text)
>>> c
Counter({'a': 3, 'l': 2, 'g': 1, 'h': 1, 'd': 1})
>>> c["a"]
3
```

# 05. collections 모듈

## Counter 모듈

- 기존 문자열값인 'gallahad'를 리스트형으로 변환한 후, text 변수에 저장하였다.
- c라는 Counter 객체를 생성하면서 text 변수를 초깃값으로 설정하고 이를 출력하면, 위 결과처럼 각 알파벳이 몇 개씩 있는지 쉽게 확인할 수 있다.
- c["a"]처럼 딕셔너리 형태의 문법을 그대로 이용해 특정 텍스트의 개수도 바로 출력할 수 있다.
- 앞서 defaultdict를 사용하여 각 문자의 개수를 셌는데, Counter를 이용하면 그런 작업을 매우 쉽게 할 수 있다.

# 05. collections 모듈

## Counter 모듈

- 다음과 같이 코드를 작성하면 정렬까지 끝낸 결과물을 확인할 수 있는데, 이전 Lab에서 수행한 작업을 단 한 줄의 코드로 작성한 것을 확인할 수 있다.

```
>>> text = """A press release is the quickest and easiest way to get free
publicity. If well written, a press release can result in multiple published
articles about your firm and its products. And that can mean new prospects
contacting you asking you to sell to them. ….""".lower().split()
>>> Counter(text)
Counter({'and': 3, 'to': 3, 'can': 2, 'press': 2, 'release': 2, 'you': 2, 'a': 2, 'sell': 1,
'about': 1, 'free': 1, 'firm': 1, 'quickest': 1, 'products.': 1, 'written,': 1, 'them.': 1,
'….': 1, 'articles': 1, 'published': 1, 'mean': 1, 'that': 1, 'prospects': 1, 'its': 1,
'multiple': 1, 'if': 1, 'easiest': 1, 'publicity.': 1, 'way': 1, 'new': 1, 'result': 1,
'the': 1, 'your': 1, 'well': 1, 'is': 1, 'asking': 1, 'in': 1, 'contacting': 1, 'get': 1})
```

# 05. collections 모듈

## Counter 모듈

- Counter 모듈은 단순히 시퀀스 자료형의 데이터를 세는 역할도 있지만, 딕셔너리 형태나 키워드형태의 매개변수를 사용하여 Counter를 생성할 수 있다.

➡ 먼저 딕셔너리 형태로 Counter 객체를 생성하는 방법이다. 다음 코드를 보면, {'red': 4, 'blue': 2}라는 초깃값을 사용하여 Counter를 생성한 것을 확인할 수 있다. 또한, elements( ) 함수를 사용하여, 각 요소의 개수만큼 리스트형의 결과를 출력하는 것을 확인할 수 있다.

```
>>> from collections import Counter
>>>
>>> c = Counter({'red': 4, 'blue': 2})
>>> print(c)
Counter({'red': 4, 'blue': 2})
>>> print(list(c.elements()))
['red', 'red', 'red', 'red', 'blue', 'blue']
```

# 05. collections 모듈

## Counter 모듈

- 키워드 형태의 매개변수를 사용하여 Counter를 생성하는 방법이다. 매개변수의 이름을 키(key)로, 실제 값을 값(value)으로 하여 Counter를 생성할 수 있다.

```
>>> from collections import Counter
>>>
>>> c = Counter(cats = 4, dogs = 8)
>>> print(c)
Counter({'dogs': 8, 'cats': 4})
>>> print(list(c.elements()))
['cats', 'cats', 'cats', 'cats', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs']
```

# 05. collections 모듈

## Counter 모듈

- Counter는 기본 사칙연산을 지원한다. 파이썬에서 지원하는 기본 연산인 덧셈, 뺄셈, 논리 연산 등이 가능하다.

```
>>> from collections import Counter
>>>
>>> c = Counter(a = 4, b = 2, c = 0, d = -2)
>>> d = Counter(a = 1, b = 2, c = 3, d = 4)
>>> c.subtract(d)                          # c - d
>>> c
Counter({'a': 3, 'b': 0, 'c': -3, 'd': -6})
```

# 05. collections 모듈

## Counter 모듈

- + 기호는 두 Counter 객체에 있는 각 요소를 더한 것이고, & 기호는 두 객체에 같은 값이 있을 때, 즉 교집합의 경우에만 출력하였다. 반대로 | 기호는 두 Counter 객체에서 하나가 포함되어 있다면, 그리고 좀 더 큰 값이 있다면 그 값으로 합집합을 적용하였다.

```
>>> from collections import Counter
>>>
>>> c = Counter(a = 4, b = 2, c = 0, d = -2)
>>> d = Counter(a = 1, b = 2, c = 3, d = 4)
>>> print(c + d)
Counter({'a': 5, 'b': 4, 'c': 3, 'd': 2})
>>> print(c & d)
Counter({'b': 2, 'a': 1})
>>> print(c | d)
Counter({'a': 4, 'd': 4, 'c': 3, 'b': 2})
```

# 05. collections 모듈

예제 z_namedtuple.py

## namedtuple 모듈

- namedtuple 모듈은 튜플의 형태로 데이터 구조체를 저장하는 방법이다.
- 튜플 원소를 인덱스가 아닌 이름으로 접근 가능한 장점이 있다.

```
>>> from collections import namedtuple
>>>
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)
>>> p
Point(x=11, y=22)
```

```
>>> Pinfo = namedtuple("biz_card", "name age phone_num")
>>> bcard_John = Pinfo("John", 30, "012-345-6789")
>>> bcard_John.name
'John'
>>> bcard_John.age
30
>>> bcard_John.phone_num
'012-345-6789'
>>> bcard_John[0]
'John'
>>> bcard_John[1]
30
>>> bcard_John[2]
'012-345-6789'
```