

데이터 과학을 위한
**파이썬
프로그래밍**



07. 자료구조

목차

1. 자료구조의 이해
2. 스택과 큐
3. 튜플과 세트
4. 딕셔너리
5. collections 모듈
6. Lab: 텍스트 마이닝 프로그램

01

자료구조의 이해

01. 자료구조의 이해

■ 자료구조의 개념

- 자료구조(data structure) : 특징이 있는 정보를 메모리에 효율적으로 저장 및 반환하는 방법으로, 데이터를 관리하는 방식이다. 특히 대용량일수록 메모리에 빨리 저장하고 빠르게 검색하여, 메모리를 효율적으로 사용하고 실행 시간을 줄일 수 있게 해 준다.

20 YELLOW PAGE 디라이프 전화번호부

대한민국 국영기관 국립중앙도서관 010-6532-6774 성도출판사 028-8616-5800 홍익출판사 00852-2529-4141 상해출판사 021-6295-5000 현대출판사 0532-8897-0001 신광출판사 024-2385-3388 광주출판사 020-3887-0555 사단출판사 029-8835-1001	가 KOTRA 중국무역관 6029-1005 충소기업진흥공단(충청) 023-6705-2399 충청일보(충청) 6382-0753 충청신문(충청) 185-8006-9033	건강 생활 정보 광안/사건사고 110 전국고급인력 112 국세청/가정안전관리 113 국세청/연세 115 국세청/연세 116 국세청/연세 117 국세청/연세 119 국세청/연세 120 국세청/연세 121 국세청/연세 122 국세청/연세 184	기업/건설 정보 건설기계 186-1040-3282 대우건설 186-1162-3396 대우건설 6734-3488 대우건설 6796-2600 / 2033 아울러스 컴퍼니 177-8319-6880	부동산 상해출판사 133-2195-2346 충청일보(충청) 187-2336-0523 충청일보(충청) 186-0218-9115 한신부동산 136-2165-5550 한신부동산 183-5801-1818	음식점 강남/가정 186-9050-0675 고향집(충청) 137-7318-1808 내고향 공방집 023-6721-0099 내고향 한식방집 023-6710-7519 내고향 치킨집 136-1822-4115 내고향 통닭집 023-6739-4999 매봉가 183-7564-8006 매봉가(충청) 139-8338-2107 매봉가(충청) 136-2834-0525 매봉가(충청) 023-6305-9288 복합식 136-4059-9991 세종권 186-9886-3758 스카이 피자국밥 185-8465-1153 매장양꼬치 186-1111-3050 일가 소파라/매장 135-0127-6665 충청 156-9210-5002 취향 집(충청) 131-4021-0035 한국가(충청) 157-3046-9099 한국강남스타일 159-9891-3032 한국강 159-2277-5826 한국 제주집 023-6703-4432	미용 / 식품 / 판매업 농산물 186-0234-0145 꽃집 186-8888-2723 꽃집 153-1028-8885 꽃집 182-2508-7772 꽃집 185-2382-7979	美好페이스아트 안면/미용/피부/화장/헤어/네일 한국화/한국화/한국화/한국화 185-8019-4590
---	---	--	--	---	--	---	---

(a) 전화번호부

9:58

그룹 연락처

Q 검색

L

나성업

C

도민준

리

라인업

마성지

(b) 휴대전화의 연락처

01. 자료구조의 이해

■ 파이썬에서의 자료구조

자료구조명	특징
스택(stack)	나중에 들어온 값을 먼저 나갈 수 있도록 해 주는 자료구조(last in first out)
큐(queue)	먼저 들어온 값을 먼저 나갈 수 있도록 해 주는 자료구조(first in first out)
튜플(tuple)	리스트와 같지만, 데이터의 변경을 허용하지 않는 자료구조
세트(set)	데이터의 중복을 허용하지 않고, 수학의 집합 연산을 지원하는 자료구조
딕셔너리 (dictionary)	전화번호부와 같이 키(key)와 값(value) 형태의 데이터를 저장하는 자료구조, 여기서 키값은 다른 데이터와 중복을 허용하지 않음
collections 모듈	위에 열거된 여러 자료구조를 효율적으로 사용할 수 있도록 지원하는 파이썬 내장(built-in) 모듈

[파이썬에서 제공하는 자료구조]

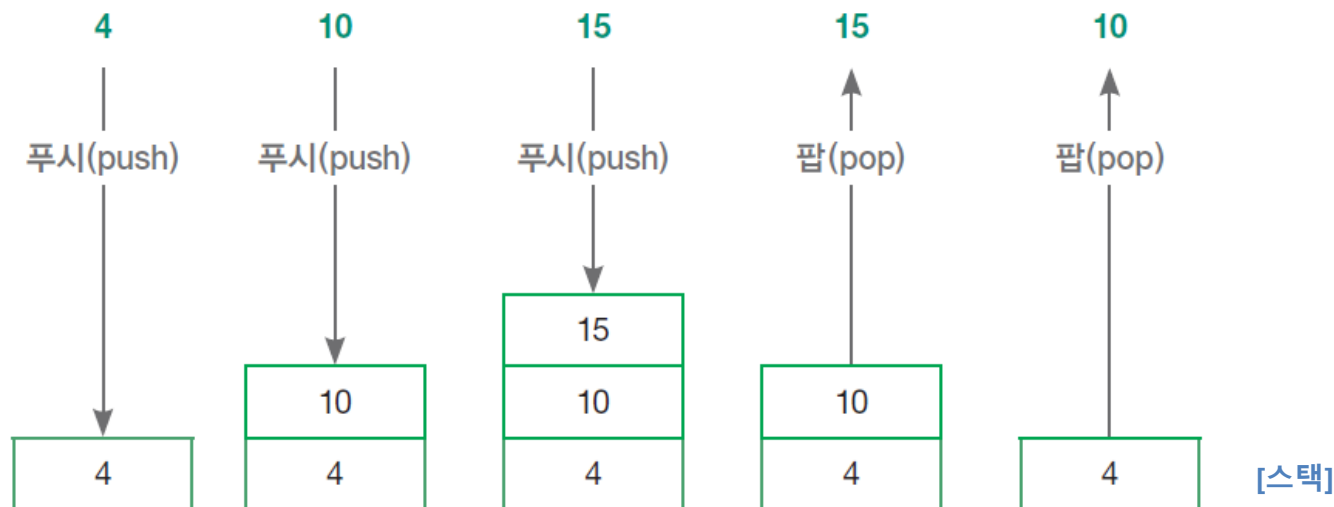
02

스택과 큐

02. 스택과 큐

■ 스택

- **스택(stack)** : 자료구조의 핵심 개념 중 하나로, 간단히 표현하면 'Last In First Out: LIFO'으로 정의할 수 있다. 즉, 마지막에 들어간 데이터가 가장 먼저 나오는 형태로, 데이터의 저장 공간을 구현하는 것이다.
- 아래 그림에서 4, 10과 같은 데이터를 저장하는 공간으로, 리스트와 비슷하지만 저장 순서가 바뀌는 형태를 스택 자료구조(stack data structure)라고 한다. 스택에서 데이터를 저장하는 것을 푸시(push), 데이터를 추출하는 것을 팝(pop)이라고 한다.



02. 스택과 큐

■ 스택

스택(stack)이라는 자료형이 따로 존재하는 것이 아니다.

파이썬에서는 list 자료를 스택인 것처럼 활용할 수 있다.

PUSH 맨 뒤에 밀어 넣기 `.append()` \leftrightarrow `.pop()` 맨 뒤에서 빼내기 POP

- 파이썬에서는 리스트를 사용하여 스택을 구현할 수 있다. 리스트라는 저장 공간을 만든 후, `append()` 함수로 데이터를 저장(push)하고 추출(pop)한다. 다음 코드를 확인해 보자.

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

- 먼저 변수 a에는 [1, 2, 3, 4, 5]가 할당된다.
- 변수 a에 10과 20을 추가하면, 변수 a에는 [1, 2, 3, 4, 5, 10, 20]이 할당된다.
- `pop()` 함수를 처음 실행하면, 가장 마지막에 저장된 20이 추출되면서 화면에 출력되고, 동시에 변수 a의 값은 [1, 2, 3, 4, 5, 10]으로 변한다.
- 다시 `pop()` 함수를 실행하면, 마지막에 저장된 10이 추출되면서 화면에 출력되고, 동시에 변수 a의 값은 [1, 2, 3, 4, 5]로 변한다.

02. 스택과 큐

■ 스택

- 스택으로 만들 수 있는 프로그램 중 하나는 입력한 텍스트의 역순을 추출하는 프로그램을 작성하는 것이다.

코드 7-1 stack.py

```
1 word = input("Input a word: ")
2 world_list = list(word)
3 print(world_list)
4
5 result = []
6 for _ in range(len(world_list)):
7     result.append(world_list.pop())
8
9 print(result)
10 print(word[::-1])
```

Input a word: PYTHON

← 사용자 입력(PYTHON)

['P', 'Y', 'T', 'H', 'O', 'N']

['N', 'O', 'H', 'T', 'Y', 'P']

NOHTYP

사용하지 않는 dummy 변수의 이름에 많이 활용되는 '_'

02. 스택과 큐

■ 스택 : [코드 7-1] 해석

- 먼저 입력한 텍스트는 변수 `word`에 저장되고, 그 값을 리스트 형으로 변환한다. 그 후 값을 차례대로 추출하면, 입력한 텍스트의 역순 값이 출력된다.
- [코드 7-1]에서 확인할 코드가 있다. 바로 `_` 기호이다. 일반적으로 `for`문에서 많이 쓰이는데, `for`문에 `_` 기호가 있으면 해당 반복문에서 생성되는 값은 코드에서 사용하지 않는다는 뜻이다. [코드 7-1]에서는 6행의 `range(len(world_list))`에서 생성되는 값이 반복문 내에서 사용되지 않으므로 `_`로 할당받은 것이다.

02. 스택과 큐

실습 1: z_queue.py

■ 큐

- **큐(queue)** : 스택과 다르게 먼저 들어간 데이터가 먼저 나오는 'Fist in First Out: FIFO'의 메모리 구조를 가지는 저장 체계이다.

PUT(A)	PUT(B)	PUT(C)	GET()	PUT(D)	GET()
				D	D
		C	C	C	C
	B	B	B	B	
A	A	A			

[큐]

큐(queue)라는 자료형이 따로 존재하는 것이 아니다.

파이썬에서는 list 자료를 큐인 것처럼 활용할 수 있다.

맨 뒤에 밀어 넣기 `.append()` \leftrightarrow `.pop(0)` 맨 앞에서 빼내기

오른 쪽에서 밀어 넣고, 왼쪽에서 빼어간다.

02. 스택과 큐

■ 큐

- 큐는 어떤 상황에서 사용할 수 있을까? 대표적인 예로, 앞서 언급한 사례 중 은행에서 대기 번호표를 뽑을 때 번호를 저장하는 방식이다. 먼저 온 사람이 앞의 번호표를 뽑고, 번호가 빠른 사람이 먼저 서비스를 받는 구조이다.
- 파이썬에서 큐를 구현하는 것은 기본적으로 스택의 구현과 같은데, `pop()` 함수를 사용할 때 인덱스가 0번째인 값을 쓴다는 의미로 `pop(0)`을 사용하면 된다. 즉, `pop()` 함수가 리스트의 마지막 값을 가져온다고 하면, `pop(0)`은 맨 처음 값을 가져온다는 뜻이다.

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> a.append(10)
```

```
# a = [1, 2, 3, 4, 5, 10]
```

```
>>> a.append(20)
```

```
# a = [1, 2, 3, 4, 5, 10, 20]
```

```
>>> a.pop(0)
```

```
1
```

```
>>> a.pop(0)
```

```
2
```

```
In[12]: a = [1, 2, 3, 4, 5]
```

```
In[13]: a.pop(1)
```

```
Out[13]: 2
```

```
In[14]: a
```

```
Out[14]: [1, 3, 4, 5]
```

`pop(index)` 인덱스 번호를 지정해서 데이터를 빼 낼 수도 있다.

append(thing) <-> pop(0)

실습 2: z_queue.py

```
8 a = [1, 2, 3, 4, 5]
9 print(a.pop(0), a)      # 1 [2, 3, 4, 5]
10 print(a.pop(0), a)     # 2 [3, 4, 5]
11 print(a.pop(0), a)     # 3 [4, 5]
12 print(a.pop(0), a)     # 4 [5]
13 print(a.pop(0), a)     # 5 []
14 #print(a.pop(0), a)    # 오류 발생!! IndexError:
15 a.append(10)
16 a.append(20)
17 a.append(30)
18 print(a.pop(0), a)     # 10 [20, 30]
19 print(a.pop(0), a)     # 20 [30]
20 print(a.pop(0), a)     # 30 []
```

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

03

튜플과 세트

03. 튜플과 세트

■ 튜플

Immutable data

이미 형성된 데이터의 내부 원소를 바꿀 수 없다는 뜻.
같은 이름으로 다시 선언하는 것은 가능하다.

- 튜플(tuple)은 리스트와 같은 개념이지만, **데이터를 변경할 수 없는 자료구조**이다

```
>>> t = (1, 2, 3)
>>> print(t + t , t * 2)
(1, 2, 3, 1, 2, 3) (1, 2, 3, 1, 2, 3)
>>> len(t)
3
```

`t = t * 2` # 가능할까?

⇒ 가능. `t`의 내용을 바꾼다기 보다는 변수 `t`를 선언하는 동작이다.

⇒ `t[0] = 10` # 이 동작이 수행 불가능하다.

- ➔ 첫 번째 줄에서 튜플을 선언하는데, 튜플은 괄호를 이용하여 `t=(1, 2, 3)`과 같은 형태로 선언한다. 대괄호 `[]`를 이용하는 리스트와는 차이가 있다. 하지만 선언 외에 여러 가지 연산은 리스트와 같아, 리스트에서 사용하는 연산, 인덱싱, 슬라이싱이 모두 동일하게 적용된다. 위의 코드처럼 튜플 간의 덧셈 `t + t`이나 곱셈 `t * 2`, 그리고 `len()`과 같은 리스트형 데이터에 사용하는 함수 모두 사용할 수 있다.

03. 튜플과 세트

■ 튜플

- 튜플과 리스트의 유일하면서도 큰 차이점이 있다면, 튜플의 값은 마음대로 변경할 수 없다는 것이다. 만약 튜플의 값을 변경하고 싶다면 다음과 같이 오류가 발생한다.

```
>>> t[1] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
In[2]: a=(1)
In[3]: type(a)
Out[3]: int
In[4]: a=(1,)
In[5]: type(a)
Out[5]: tuple
In[6]: a=[1]
In[7]: type(a)
Out[7]: list
```

← (괄호)는 꼭 튜플자료만을 의미하는 것이 아니다.
없었을 때 의미를 가지면 그 의미로 해석한다.

← 튜플로 해석할 수 밖에 없다. a=(2, 3)도 마찬가지이다.

03. 튜플과 세트

z_return_value.py

■ 튜플

- 튜플은 언제 사용할까? 함수의 return value

사실 프로그래밍을 하다 보면 자신이 하나의 함수만 만들고, 다른 사람이 그 함수의 결과값을 사용해야 하는 경우가 발생할 수 있다. 이때 반환해 주는 타입을 튜플로 선언하여 받아서 사용하는 사람이 마음대로 데이터를 바꾸지 못하게 할 수 있다.

- 그렇다면 잘 바뀌지 않는 데이터는 어떤 것이 있을까?

학번이나 이름, 주민등록번호와 같이 변경되지 않아야 하는 정보 등이다. 프로그래머가 이러한 이해 없이 마음대로 값을 변경하려고 할 때, 튜플은 이를 방지하는 기능을 한다.

```
7 def f1(a): return a, a * 2, a / 2
8 def f2(a): return [a, a * 2, a / 2]
9 def f3(a): return list((a, a * 2, a / 2))
10 def f4(a): return (a, )      # =return tuple((a, ))
11
12
13 ret = f1(10); print(1, type(ret), ret)
14 ret = f2(10); print(2, type(ret), ret)
15 ret = f3(10); print(3, type(ret), ret)
16 ret = f4(10); print(4, type(ret), ret)
17 a, b, c = f2(10); print(a, b, c)
```

```
1 <class 'tuple'> (10, 20, 5.0)
2 <class 'list'> [10, 20, 5.0]
3 <class 'list'> [10, 20, 5.0]
4 <class 'tuple'> (10,)
10 20 5.0
```

03. 튜플과 세트

■ 세트

- 세트(set) : 값을 순서 없이 저장하면서 중복을 불허하는 자료형이다. 세트는 튜플과 다르게 삭제나 변경이 가능하며, 다양한 집합 연산을 제공한다.

```
>>> s = set([1, 2, 3, 1, 2, 3])      # set() 함수를 사용하여 1, 2, 3을 세트 객체로 생성
>>> s
{1, 2, 3}
```



- 세트를 사용하기 위해 set() 함수를 사용하여 리스트나 튜플의 데이터를 넣으면, 해당 값이 세트 형태로 변환된다. 위 코드처럼 [1, 2, 3, 1, 2, 3]이라는 리스트형의 값을 세트로 변환하면, 중복을 제거한 후 {1, 2, 3}으로 변환되어 출력된다.

```
In[9]: s = {1, 2, 3, 2, 3, 4}
In[10]: s
Out[10]: {1, 2, 3, 4}
In[11]: l = [1, 2, 3, 2, 3, 4]
In[12]: l
Out[12]: [1, 2, 3, 2, 3, 4]
In[13]: set(l)
Out[13]: {1, 2, 3, 4}
In[14]: {[1, 2, 3, 2, 3, 4]}
Traceback (most recent call last):
  File "C:\Python\Python3.7.0\lib\s
    exec(code_obj, self.user_global
  File "<ipython-input-14-3a6d6b92f
    {[1, 2, 3, 2, 3, 4]}
TypeError: unhashable type: 'list'
```

03. 튜플과 세트

set methods

https://www.w3schools.com/python/python_ref_set.asp

■ 세트

- 세트는 튜플과 다르게 삭제나 변경이 가능하다. 이 기능을 위해 다양한 함수를 다음과 같이 지원한다.

```
>>> s
{1, 2, 3}
>>> s.add(1)           # 1을 추가하는 명령이지만, 중복 불허로 추가되지 않음
>>> s
{1, 2, 3}
>>> s.remove(1)        # 1 삭제
>>> s
{2, 3}
>>> s.update([1, 4, 5, 6, 7]) # [1, 4, 5, 6, 7] 추가
>>> s
{1, 2, 3, 4, 5, 6, 7}
>>> s.discard(3)       # 3 삭제
>>> s
{1, 2, 4, 5, 6, 7}
>>> s.clear()          # 모든 원소 삭제
>>> s
set()
```

add() : 원소 하나를 추가

remove() 또는 **discard()** : 원소 하나를 제거

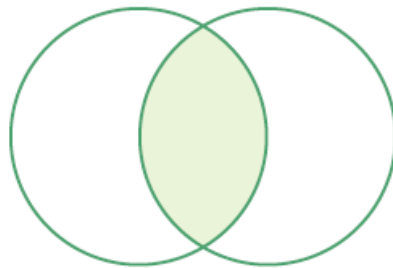
update() : 새로운 원소(여러 개)를 추가

clear() : 모든 원소를 지우기

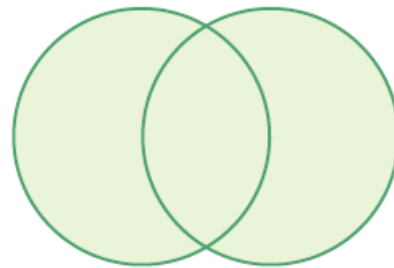
03. 튜플과 세트

■ 세트

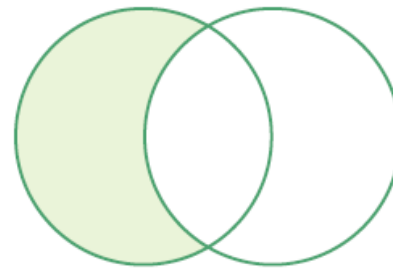
- 파이썬의 세트는 수학의 집합과 마찬가지로 다양한 집합 연산을 제공한다.



집합 1 집합 2
(a) 교집합



집합 1 집합 2
(b) 합집합



집합 1 집합 2
(c) 차집합

[집합 연산]

03. 튜플과 세트

■ 세트

```
>>> s1 = set([1, 2, 3, 4, 5])
>>> s2 = set([3, 4, 5, 6, 7])
>>>
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7}
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7}
>>> s1.intersection(s2)
{3, 4, 5}
>>> s1 & s2
{3, 4, 5}
>>> s1.difference(s2)
{1, 2}
>>> s1 - s2
{1, 2}
```

s1과 s2의 합집합

set([1, 2, 3, 4, 5, 6, 7])

s1과 s2의 교집합

set([3, 4, 5])

s1과 s2의 차집합

set([1, 2])

```
In[17]: s1 = {1, 2, 3, 4}
```

```
In[18]: s2 = {3, 4, 5, 6}
```

```
In[19]: a = s1.union(s2)
```

```
In[20]: a
```

```
Out[20]: {1, 2, 3, 4, 5, 6}
```

```
In[21]: s1
```

```
Out[21]: {1, 2, 3, 4}
```

```
In[22]: b = s1 | s2
```

```
In[23]: b
```

```
Out[23]: {1, 2, 3, 4, 5, 6}
```

03. 튜플과 세트

■ 세트

- 합집합은 두 집합의 중복값을 제거하고 합치는 연산이다. 위 코드에서는 `s1.union(s2)` 를 통해 `s1`과 `s2`의 합집합이 출력되었다. 합집합은 `union`과 같은 함수로도 표현할 수 있지만, `|` 기호로도 추출할 수 있다. 위 코드에서 `s1 | s2`의 결과가 `s1.union(s2)`와 동일한 것을 확인할 수 있다.
- 교집합은 두 집합 양쪽에 모두 포함된 값만 추출하는 연산이다. 위 코드에서 `s1`과 `s2`는 모두 3, 4, 5를 원소로 가지고 있다. 이 경우, `s1.intersection(s2)`나 `s1 & s2`로 교집합을 추출 할 수 있다.
- 차집합은 앞에 있는 집합 `s1`의 원소 중 `s2`에 포함된 원소를 제거하는 연산이다. 즉, `s1`에서 `s1`과 `s2`의 교집합 원소를 삭제하면 된다. 앞선 코드에서 `s1`은 [1, 2, 3, 4, 5]를 가지고 있으므로 [3, 4, 5]를 제거하면 [1, 2]만 남는다. 코드로 표현하면 `s1.difference(s2)` 또는 `s1 - s2`로 차집합을 추출할 수 있다.

03. 튜플과 세트

■ 세트

연산	함수	기호	예시
합집합	union		s1.union(s2), s1 s2
교집합	intersection	&	s1.intersection(s2), s1 & s2
차집합	difference	-	s1.difference(s2), s1 - s2

[세트의 집합 연산]

집합 자료의 로직 연산과 메소드

z_set_methods.py

```
13 # 실습 1: set 로직 연산
14 a = {5, 2, 8}
15 print('1, intersection:', a.intersection({1, 3, 5}), a & {1, 3, 5})    # {5}
16 print('2, union:', a.union({1, 3, 5}), a | {1, 3, 5})                # {1, 2, 3, 5, 8}
17 print('3, difference:', a.difference({1, 3, 5}), a - {1, 3, 5})        # {8, 2}
18
19 # 실습 2: set methods
20 b = a.add('9')    # 1개의 원소만 추가. 주의!!! 반환값 없음.
21 print('4, add:', a, b)    # {8, 2, '9', 5} None
22 b = a.update('B', 'A')    # 1개 이상의 원소를 추가. 주의!!! 반환값 없음.
23 print('5, update:', a, b)    # {2, 5, '8', 8, 'A', '9'} None
24 b = a.remove('9')    # 1개만 삭제 가능. 주의!!! 반환값 없음.
25 print('6, remove:', a, b)    # {2, 5, 8, 'A', 'B'} None
26 b = a.discard('B')    # 역시 1개만 삭제 가능. 주의!!! 반환값 없음.
27 print('7, discard:', a, b)    # {2, 5, 8, 'A'} None
28 b = a.clear()    # 모든 원소 삭제. 주의!!! 반환값 없음.
29 print('8, clear:', a, b)    # set{} None
```

```
1, intersection: {5} {5}
2, union: {1, 2, 3, 5, 8} {1, 2, 3, 5, 8}
3, difference: {8, 2} {8, 2}
4, add: {8, 2, 5, '9'} None
5, update: {2, 5, 8, 'A', '9', 'B'} None
6, remove: {2, 5, 8, 'A', 'B'} None
7, discard: {2, 5, 8, 'A'} None
8, clear: set() None
```


04

딕셔너리

04. 딕셔너리

■ 딕셔너리의 개념

- 딕셔너리(dictionary) : 전화번호부와 같이 키(key)와 값(value) 형태로 데이터를 저장하는 자료구조이다.
- 리스트나 튜플과는 달리 **인덱스를 이용하여 원소에 접근하지 않고, 키를 이용해 원소값(value)에 접근한다.**

학번	이름	생년월일	주소
20150230	홍길동	1995-04-03	서울시 동대문구
20150233	김영철	1995-04-20	성남시 분당구
20150234	오영심	1996-01-03	성남시 중원구
20150236	최성철	1995-12-27	인천시 계양구

[대학생 인적사항]

04. 딕셔너리

실습 0: z_dic1.py

■ 파이썬에서의 딕셔너리

- 파이썬에서 딕셔너리의 선언은 중괄호 {}를 사용하여 키와 값의 쌍으로 구성하면 된다.

딕셔너리 변수 = {키 1:값 1, 키 2:값 2, 키 3:값 3, ...}

딕셔너리 = {키1: 값1, 키2: 값2}

딕셔너리 = dict(키1=값1, 키2=값2)

딕셔너리 = dict(zip([키1, 키2], [값1, 값2]))

딕셔너리 = dict([(키1, 값1), (키2, 값2)])

딕셔너리 사용하기

```
1 <class 'dict'> {1: 'A', 2: 'C'}  
2 <class 'dict'> {'a1': 'A', 'a2': 'C'}  
3 <class 'dict'> {1: 6, 3: 9}  
4 <class 'dict'> {1: 'A', 'B': 7}
```

```
10 a = {1:'A', 2:'C'}; print(1, type(a), a)  
11 a = dict(a1='A', a2='C'); print(2, type(a), a) # 1='A'. 2='B'은 안됨.  
12 a = dict(zip([1, 3], [6, 9])); print(3, type(a), a)  
13 a = dict([(1, 'A'), ('B', 7)]); print(4, type(a), a)
```

zip 함수의
활용사례

```
for i, j in zip([1, 3], [6, 9]):  
    print(i, j)
```

출력
1 6
3 9

04. 딕셔너리

■ 파이썬에서의 딕셔너리

학번(키)	이름(값)
20140012	Janhyeok
20140059	Jiyong
20150234	JaeHong
20140058	Wonchul

[키와 값의 샘플]

- 표의 정보를 간단히 파이썬으로 표현해보자. student_info라는 변수를 먼저 선언한 후, 해당 변수에 {키:값} 형태로 값을 입력한다. 그럼 해당 변수는 간단히 저장된다.

```
>>> student_info = {20140012:'Sungchul', 20140059:'Jiyong', 20140058:'JaeHong'}
```

04. 딕셔너리

■ 파이썬에서의 딕셔너리

- 해당 변수에서 특정 값을 호출하는 방법이다. 해당 값의 키를 대괄호 [] 안에 넣어 호출할 수 있다. 변수의 자료형을 정확히 모르고 호출한다면, 리스트로 오해할 수도 있다.

```
>>> student_info[20140012]
'Sungchul'
```

- 재할당과 데이터 추가이다

```
>>> student_info[20140012] = 'Janhyeok'
>>> student_info[20140012]
'Janhyeok'
>>> student_info[20140039] = 'Wonchul'
>>> student_info
{20140012: 'Janhyeok', 20140059: 'Jiyong', 20140058: 'JaeHong', 20140039: 'Wonchul'}
```

재할당

추가

04. 딕셔너리

실습 1: z_dic1.py

```
53 # 사전형 자료 code를 생성한다.
54 code = {100: 'KT', 101: 'LG', 106: 'SK', 112: 'Crime', 119: 'Emergency'}
55 print('1) code: ', type(code), '|', code)
56
57 # keys() 메소드로 사전형 자료에서 key 원소를 리스트 형태를 반환받는다.
58 d_keys = code.keys()
59 print('2) code.keys(): ', type(d_keys), d_keys)
60
61 # 리스트 자료로 변환한다.
62 a = list(d_keys)
63
64 # values() 메소드로 사전형 자료에서 value 원소를 리스트 형태를 반환받는다.
65 d_values = code.values()
66 print('3) code.values: ', type(d_values), d_values)
67
68 # 리스트 자료로 변환한다.
69 b = list(d_values)
```

```
1) code: <class 'dict'> | {100: 'KT', 101: 'LG', 106: 'SK', 112: 'Crime', 119: 'Emergency'}
2) code.keys(): <class 'dict_keys'> dict_keys([100, 101, 106, 112, 119])
3) code.values: <class 'dict_values'> dict_values(['KT', 'LG', 'SK', 'Crime', 'Emergency'])
```

04. 딕셔너리

실습 1: z_dic1.py

```
71 # key와 value로 이루어진 2개의 list로 부터 사전형 자료를 복원한다.  
72 code_2 = dict(zip(a, b))  
73 print('4) dict(zip(a, b)): ', type(code_2), '|', code_2)  
74  
75 # items() 메소드로 사전형 자료로 부터 dict_items 자료를 반환받는다.  
76 d_items = code.items()  
77 print('5) code.items: ', type(d_items), '|', d_items)  
78  
79 # dict_items 자료를 list 자료로 변환한다.  
80 print('6) list(code.items): ', type(list(code.items())), '|', list(code.items()))
```

```
4) dict(zip(a, b)): <class 'dict'> | {100: 'KT', 101: 'LG', 106: 'SK', 112: 'Crime', 119: 'Emergency'}  
5) code.items: <class 'dict_items'> | dict_items([(100, 'KT'), (101, 'LG'), (106, 'SK'), (112, 'Crime'), (119, 'Emergency')])  
6) list(code.items): <class 'list'> | [(100, 'KT'), (101, 'LG'), (106, 'SK'), (112, 'Crime'), (119, 'Emergency')]
```

```
82 # 변외: 사전형 자료를 list로 변환하면 key만 반환한다.  
83 # value를 리스트로 반환 받으려면? : list(code.values())  
84 print(list(code))  
85 # [100, 101, 106, 112, 119]  
86  
87 print(list(code.values()))  
88 # ['KT', 'LG', 'SK', 'Crime', 'Emergency']
```

■ 딕셔너리의 생성

- 국가명과 국가 전화번호를 묶어 보여 주는 코드를 작성하면 다음과 같다.

```
>>> country_code = {}                                     # 딕셔너리 생성
>>> country_code = {"America": 1, "Korea": 82, "China": 86, "Japan": 81}
>>> country_code
{'America': 1, 'Korea': 82, 'China': 86, 'Japan': 81}
```


■ 딕셔너리의 메소드

- `keys()` 메소드는 자료형 구조 내의 `key`를 `dict_keys` 형으로 반환한다. 이 자료는 `list()` 함수로 리스트화 할 수 있다.

```
>>> country_code.keys()                # 딕셔너리의 키만 출력
dict_keys(['America', 'Korea', 'China', 'Japan'])
```

- `values()` 메소드는 사전형 자료 내의 `value`를 `dict_values` 형으로 반환한다. 이 자료는 `list()` 함수로 리스트화 할 수 있다.

```
>>> country_code["German"] = 49        # 딕셔너리 추가
>>> country_code
{'America': 1, 'Korea': 82, 'China': 86, 'Japan': 81, 'German': 49}
>>> country_code.values()              # 딕셔너리의 값만 출력
dict_values([1, 82, 86, 81, 49])
```

■ 딕셔너리의 메소드

- 키-값 쌍을 모두 보여 주기 위해서는 items() 메소드를 사용한다.

```
>>> country_code.items()                # 딕셔너리 데이터 출력
dict_items([('America', 1), ('Korea', 82), ('China', 86), ('Japan', 81), ('German', 49)])
```

■ 딕셔너리의 메소드

- 실제로 딕셔너리를 사용할 때는 for문과 함께 사용한다. 다음 코드와 같이 키-값 쌍을 화면에 출력할 수 있다.

```
>>> for k, v in country_code.items():  
...     print("Key:", k)  
...     print("Value:", v)  
...  
Key: America  
Value: 1  
Key: Korea  
Value: 82  
Key: China  
Value: 86  
Key: Japan  
Value: 81  
Key: Gernman  
Value: 49
```

■ 딕셔너리의 메소드

- 딕셔너리를 많이 사용하는 방법 중 하나는 if문을 사용하여 특정 키나 값이 해당 변수에 포함되어 있는지 확인하는 것이다.

```
>>> "Korea" in country_code.keys()
```

```
True
```

```
# 키에 "Korea"가 있는지 확인
```

```
>>> 82 in country_code.values()
```

```
True
```

```
# 값에 82가 있는지 확인
```

Dictionary method

<https://realpython.com/python-dicts/>

https://www.w3schools.com/python/python_ref_dictionary.asp

Method	Description
<code>clear()</code>	Removes all the elements from the dictionary
<code>copy()</code>	Returns a copy of the dictionary
<code>fromkeys()</code>	Returns a dictionary with the specified keys and values
<code>get()</code>	Returns the value of the specified key
<code>items()</code>	Returns a list containing a tuple for each key value pair
<code>keys()</code>	Returns a list containing the dictionary's keys
<code>pop()</code>	Removes the element with the specified key
<code>popitem()</code>	Removes the last inserted key-value pair
<code>setdefault()</code>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<code>update()</code>	Updates the dictionary with the specified key-value pairs
<code>values()</code>	Returns a list of all the values in the dictionary

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, 덱)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

deque.methods

- 그 외 다음과 같은 다양한 메소드를 지원한다.
 - deque는 stack & queue는 물론 여타의 자료 형을 대부분 지원한다.
 - .extend([items]) - 우측에 list, tuple로 묶인 여러 개의 자료를 추가
 - .extendleft([items]) - 좌측에 list, tuple로 묶인 여러 개의 자료를 추가
 - .rotate(n) - 우측으로 원소들을 n회 회전. -n은 좌측으로..
 - .reverse() - 원소의 배열 순서를 역순으로 바꿈.
 - .remove(item) - 자료에서 지정된 값(item)을 삭제

■ 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])
```

■ 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])
```

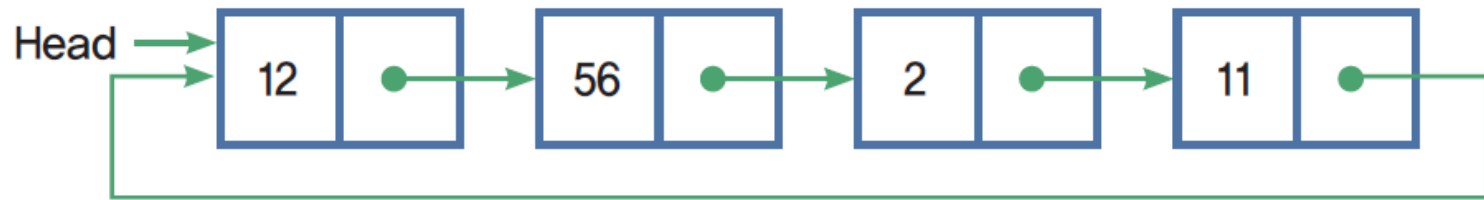
■ deque 모듈

- 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])
```

■ deque 모듈

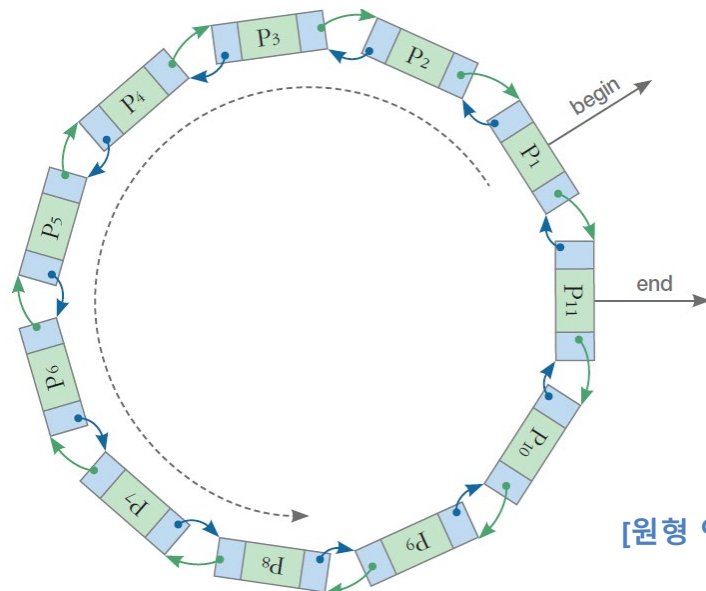
- **deque 모듈의 장점** : deque는 연결 리스트의 특성을 지원한다. 연결 리스트는 데이터를 저장할 때 요소의 값을 한 쪽으로 연결한 후, 요소의 다음 값의 주소값을 저장하여 데이터를 연결하는 기법이다.



[연결 리스트의 형태]

■ 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]

■ 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)
...
<del>>> print(deque_list)
<del>deque([0, 1, 2, 3, 4])</del>
>>> 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])
```

■ 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])
```


연습: deque를 stack처럼 사용하기

실습 1: 예제 z_deque.py

```
type(st)= <class 'collections.deque'> | st: deque([])
st: deque([0])
st: deque([0, 1])
st: deque([0, 1, 2])
st: deque([0, 1, 2, 3])
st: deque([0, 1, 2, 3, 4])
popped item= 4 | st: deque([0, 1, 2, 3])
popped item= 3 | st: deque([0, 1, 2])
popped item= 2 | st: deque([0, 1])
popped item= 1 | st: deque([0])
popped item= 0 | st: deque([])
```

Stack 동작

오른쪽으로 밀어 넣고 오른 쪽에서 빼어 간다.

⇒ 맨 나중에 저장한 것이 제일 먼저 인출된다. Last In First Out

```
28 st = deque()
29 print('type(st)=', type(st), '| st:', st)
30
31 # 1) push 동작
32 for i in range(5):
33     st.append(i)
34     print('st:', st)
35
36 # 2) pop 동작
37 print('popped item=', st.pop(), '| st:', st)
38 print('popped item=', st.pop(), '| st:', st)
39 print('popped item=', st.pop(), '| st:', st)
40 print('popped item=', st.pop(), '| st:', st)
41 print('popped item=', st.pop(), '| st:', st)
42 #print('popped item=', st.pop(), '| st:', st)
# 오류 발생 종료. IndexError: pop from an empty deque
```

연습: deque를 queue처럼 사용하기

실습 2: 예제 z_deque.py

```
57 que = deque()
58 print('type(que)=', type(que), '\nque:', que)
59 # type(que)= <class 'collections.deque'>
60 # que: deque([])
61
62 # 1) push 동작
63 for i in range(5):
64     que.appendleft(i)
65     print('que:', que)
66
67 # 2) pop 동작
68 print('popped item=', que.pop(), '| que:', que)
69 print('popped item=', que.pop(), '| que:', que)
70 print('popped item=', que.pop(), '| que:', que)
71 print('popped item=', que.pop(), '| que:', que)
72 print('popped item=', que.pop(), '| que:', que)
73 # print('popped item=', que.pop(), '| que:', que)
74 # 수행오류. IndexError: pop from an empty deque
```

```
type(que)= <class 'collections.deque'>
que: deque([])
que: deque([0])
que: deque([1, 0])
que: deque([2, 1, 0])
que: deque([3, 2, 1, 0])
que: deque([4, 3, 2, 1, 0])
popped item= 0 | que: deque([4, 3, 2, 1])
popped item= 1 | que: deque([4, 3, 2])
popped item= 2 | que: deque([4, 3])
popped item= 3 | que: deque([4])
popped item= 4 | que: deque([])
```

list 자료를 큐로 활용 때는 좀 다르다. 이때는 오른쪽에서 밀어 넣고, 왼쪽에서 빼어간다.

우측에서 밀어 넣기 .append() ↔ .pop(0) 좌측에서 빼내기

Queue 동작

좌측에서 밀어 넣고 오른쪽에서 빼어간다.

⇒ 제일 먼저 저장한 것이 제일 먼저 인출된다. First In First Out

연습: deque의 여러 methods

실습 3: 예제 z_deque.py

```
83 d = deque([1, 2, 'q'])
84 d.append('A')           # 원소 1개만 오른 쪽에 추가.
85 d.append((8, 9))
86 d.appendleft('0')       # 원소 1개만 왼 쪽에 추가
87 print('1)', d)
88 # 1) deque(['0', 1, 2, 'q', 'A', (8, 9)])
89
90
91 # extend/extendleft: 여러 개의 원소를 추가한다.
92 # 넣을 원소들을 tuple, list의 형태로 하나의 자료형으로 만들어야 한다.
93 d.extend(('b', 13))     # deque의 오른 쪽에서 2개의 자료가 밀려 들어 감.
94 d.remove('q')
95 print('2)', d)
96 # 2) deque(['0', 1, 2, 'A', (8, 9), 'b', 13])
97
98
99 d.extendleft([1, 2])    # 앞의 1부터 deque의 왼쪽으로 들어 감
100 print('3)', d)
101 # 3) deque([2, 1, '0', 1, 2, 'A', (8, 9), 'b', 13])
102
103 print('4)', d.pop(), d)
104 # 4) 13 deque([2, 1, '0', 1, 2, 'A', (8, 9), 'b'])
105
106 print('5)', d.popleft(), d)
107 # 5) 2 deque([1, '0', 1, 2, 'A', (8, 9), 'b'])
```

```
1) deque(['0', 1, 2, 'q', 'A', (8, 9)])
2) deque(['0', 1, 2, 'A', (8, 9), 'b', 13])
3) deque([2, 1, '0', 1, 2, 'A', (8, 9), 'b', 13])
4) 13 deque([2, 1, '0', 1, 2, 'A', (8, 9), 'b'])
5) 2 deque([1, '0', 1, 2, 'A', (8, 9), 'b'])
```

연습: deque의 여러 methods

실습 3: 예제 z_deque.py

```
112 d = deque(range(5))
113 print('1)', d)
114 # 1) deque([0, 1, 2, 3, 4])
115
116 d.rotate(2)          # 오른쪽으로 회전
117 print('2)', d)
118 # 2) deque([3, 4, 0, 1, 2])
119
120 d.rotate(-1)         # 왼쪽으로 회전
121 print('3)', d)
122 # 3) deque([4, 0, 1, 2, 3])
```

```
1) deque([0, 1, 2, 3, 4])
2) deque([3, 4, 0, 1, 2])
3) deque([4, 0, 1, 2, 3])
```

연습: deque의 여러 methods

실습 3: 예제 z_deque.py

```
126 d = deque(i for i in range(8))
127 print('1)', d, '| len(d)=', len(d))
128 # 1) deque([0, 1, 2, 3, 4, 5, 6, 7]) | len(d)= 8
129 d = deque(maxlen=5) # 최대 길이 제한이 있는 deque를 선언
130 print('2)', d, '| len(d)=', len(d))
131 # 2) deque([], maxlen=5) | len(d)= 0
132 d.extend([1, 2, 3, 4, 5, 6]) # 최대 길이가 5이므로 앞의(왼쪽) 것 하나를 버린다.
133 print('3)', d, '| len(d)=', len(d))
134 # 3) deque([2, 3, 4, 5, 6], maxlen=5) | len(d)= 5
135 d.append('A') # 오른쪽에서 추가하므로 왼쪽에 있는 원소가 소멸된다.
136 print('4)', d, '| len(d)=', len(d))
137 # 4) deque([3, 4, 5, 6, 'A'], maxlen=5) | len(d)= 5
138 d.appendleft('0') # 왼쪽에서 추가하므로 오른쪽의 원소가 소멸된다.
139 print('5)', d, '| len(d)=', len(d))
140 # 5) deque(['0', 3, 4, 5, 6], maxlen=5) | len(d)= 5
141 d.extend([i for i in range(8)]) # 8개를 선언하였기 때문에 3개는 앞에 선언된 순으로 없어진다.
142 print('6)', d, '| len(d)=', len(d))
143 # 6) deque([3, 4, 5, 6, 7], maxlen=5) | len(d)= 5
144 d.reverse()
145 print('7)', d, '| len(d)=', len(d))
146 # 7) deque([7, 6, 5, 4, 3], maxlen=5) | len(d)= 5
```

```
1) deque([0, 1, 2, 3, 4, 5, 6, 7]) | len(d)= 8
2) deque([], maxlen=5) | len(d)= 0
3) deque([2, 3, 4, 5, 6], maxlen=5) | len(d)= 5
4) deque([3, 4, 5, 6, 'A'], maxlen=5) | len(d)= 5
5) deque(['0', 3, 4, 5, 6], maxlen=5) | len(d)= 5
6) deque([3, 4, 5, 6, 7], maxlen=5) | len(d)= 5
7) deque([7, 6, 5, 4, 3], maxlen=5) | len(d)= 5
```

OrderedDict 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)
```

일반 사전형 자료는 입력한 순서대로 데이터가 나열되는 것을 보장하지 않는다. 현재의 사례는 입력한 대로 나열되어 있지만, 보장된 것이 아니다.

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

■ OrderedDict 모듈

***** Python 3.6's dict is now ordered by insertion order (reducing the usefulness of OrderedDict) !!!!

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)
```

OrderedDict로 선언한 사전형자료는 입력 순서를 기억하여 나열되는 것이 보장된다.

다만, 본 예제로는 그것을 증명하여 보여주지는 못하고 있다.

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

OrderedDict 모듈

코드 7-4 ordereddict3.py

```
1 def sort_by_key(t):
2     return t[0]
3
4 from collections import OrderedDict      # OrderedDict 모듈 선언
5
6 d = dict()
7 d['x'] = 100
8 d['y'] = 200
9 d['z'] = 300
10 d['l'] = 500
11
12 for k, v in OrderedDict(sorted(d.items(), key=sort_by_key)).items():
13     print(k, v)
```

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


■ 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]
```

사전연습: lambda function

z_practice_lambda.py

람다 함수는 입력파라미터와 수행표현식만 제시하는 1줄 짜리 *anonymous function*이다.
(함수의 이름은 안 밝히고,) *argument*만 전달하면 실제 수행되는 루틴은 *expression*이다.
표현 식 => *lambda arguments: expression*

```
11 # 이름 없는 함수의 수행 결과를 출력한다.
12 # 그 함수는 입력받은 데이터를 1 증가시켜 반환하는 동작을 수행한다.
13
14 print((lambda x: x + 1)(2)) # (lambda x: x + 1)=> 함수이름, (2)=> 인자=2
15 # arguments=x, expression=x+1
16 # (lambda x: x + 1)(2) = lambda 2: 2 + 1
17 # = 2 + 1
18
19 add_one = lambda x: x + 1 # 함수이름을 add_one으로 명시하였다고 생각할 수 있음.
20 print(add_one(2))
21
22 # 2개의 인자를 갖는 람다 함수를 full_name으로 정의한다.
23 full_name = lambda first, last: f'Full name: {first.title()} {last.title()}'
24 print(full_name('guido', 'van rossum'))
25 # 'Full name: Guido Van Rossum'
```

3

3

Full name: Guido Van Rossum

사전연습: sort(), sorted() 함수

z_practice_sorted.py

```
15 a = [3, 8, 2, 7, 3, 1]
16 a.sort()      # list에만 적용되는 sort 메소드. 반환값 없음.
17 print(a)      # [1, 2, 3, 3, 7, 8]
18
19 b = sorted([5, 2, 3, 1, 4])      # 파이썬 내장함수 sort
20 print(b)      # [1, 2, 3, 4, 5]
21
22 # 사전형 자료에 대해서는 키를 정렬한다.
23 c = sorted({9: 'D', 2: 'B', 7: 'B', 5: 'E', 4: 'A'})
24 print(type(c), c)      # <class 'list'> [2, 4, 5, 7, 9]
```

```
[1, 2, 3, 3, 7, 8]
[1, 2, 3, 4, 5]
<class 'list'> [2, 4, 5, 7, 9]
```

사전연습: sort(), sorted() 함수

z_practice_sorted.py

```
29 # list.sort()와 sorted()는 모두 비교하기 전에
30 # 각 리스트 요소에 대해 호출할 함수를 지정하는 key 매개 변수를 가지고 있다.
31 # key 매개 변수의 값은 단일 인자를 취하고
32 # 정렬 목적으로 사용할 키를 반환하는 함수여야 한다.
33 # sorted(자료, key=Something) 함수는 호출되면
34 # 자료의 각 원소를 key에서 지정한 기준에 따라 소팅한다.
35 print(sorted("Hello, I am Jin".split(), key=str.lower))
36 # ['am', 'Hello,', 'I', 'Jin'] 소문자 관점에서 정렬
37
38 # lambda 함수는 lambda arguments: expression 형식으로
39 # arguments를 받아 expression을 수행한다.
40 # 아래 사례에서는 sorted(자료, key=Something)에서 Something이 lamda 함수이다.
41 # arguments로서 tuple 1세트를 전달받고,
42 # expression에서는 그 튜플의 [2]번 원소를 반환한다.
43 # 이로서 소팅의 기준 key=튜플의 [2]번 원소가 되어 이를 소팅의 기준으로 삼게 된다.
44 student_tuples = [('john', 'A', 15), ('Jane', 'B', 12), ('dave', 'B', 10)]
45 a = sorted(student_tuples, key=lambda student: student[2]) # sort by age
46 print(a) # [('dave', 'B', 10), ('Jane', 'B', 12), ('john', 'A', 15)]
47
48 a = sorted(student_tuples, key=lambda abc: abc[0]) # sort by name
49 print(a) # [('Jane', 'B', 12), ('dave', 'B', 10), ('john', 'A', 15)]
```

```
['am', 'Hello,', 'I', 'Jin']
[('dave', 'B', 10), ('Jane', 'B', 12), ('john', 'A', 15)]
[('Jane', 'B', 12), ('dave', 'B', 10), ('john', 'A', 15)]
```

연습: OrderedDict

z_orderedDict.py

```
26 d = dict()
27 d['d'] = 100; d['c'] = 200; d['B'] = 300; d['a'] = 500
28 print(f'1.1) {type(d)}\n{d}')
29 # 1) <class 'dict'>
30 # {'d': 100, 'c': 200, 'B': 300, 'a': 500}
31
32 # (key, value) 중에서 index 0라 함은 key가 소팅의 판단기준이 된다.
33 sd = sorted(d.items(), key=lambda abc: abc[0])
34 print(f'2) {type(sd)}\n{sd}')
35 # 2) <class 'list'>
36 # [('B', 300), ('a', 500), ('c', 200), ('d', 100)]
37
38 od = OrderedDict(sd)
39 print(f'3) {type(od)}\n{od}')
40 # 3) <class 'collections.OrderedDict'>
41 # OrderedDict([('B', 300), ('a', 500), ('c', 200), ('d', 100)])
42
43 for k, v in od.items():
44     print(k, v, end=' ')
45 # B 300 a 500 c 200 d 100
46 # 줄번호 33에서 abc[0] -> abc[1]으로
47 # 고쳐 적으면 정렬 기준은 value가 된다.
48 # d 100 c 200 B 300 a 500
```

```
1.1) <class 'dict'>
{'d': 100, 'c': 200, 'B': 300, 'a': 500}
2) <class 'list'>
[('B', 300), ('a', 500), ('c', 200), ('d', 100)]
3) <class 'collections.OrderedDict'>
OrderedDict([('B', 300), ('a', 500), ('c', 200), ('d', 100)])
B 300 a 500 c 200 d 100
```

defaultdict is faster for larger data sets with more homogenous key sets.

■ 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'
```

■ defaultdict 모듈

- 그렇다면 defaultdict 모듈은 어떻게 작동할까?

코드 7-6 defaultdict2.py

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

Default 값을 0으로 설정

람다함수 해석:
어떤 argument가 오든지 반환값은 0이다.

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.

연습: defaultdict 모듈

defaultdict2.py

```
3 # lambda: 0는 return 0로 이해한다.
4 # 즉, 어떤 파라미터가 들어오더라도 0을 반환한다는 것이다.
5 d = defaultdict(lambda: 0) # Default 값을 0으로 설정
6
7 print(d["first"])          # 0
8 print(d[3])                # 0
9 print(type(d), len(d))
10 # <class 'collections.defaultdict'> 2
11
12 d = defaultdict()
13 print(type(d), len(d))
14 # <class 'collections.defaultdict'> 0
15
16 print(d[3])                # 수행 오류 발생. KeyError: 3
```

초기화를 미리 행해두지 않으면 defaultdict라 해도 접근 오류가 발생한다.

```
0
0
<class 'collections.defaultdict'> 2
<class 'collections.defaultdict'> 0
Traceback (most recent call last):
```


■ defaultdict 모듈

- defaultdict의 초기값은 [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])])
```

■ Counter 모듈

- Counter 모듈은 시퀀스 자료형의 원소와 그 출현 빈도를 딕셔너리 형태로 반환하는 자료 구조(클래스 함수)이다. 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
```

■ 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 썼는데, Counter를 이용하면 그런 작업을 매우 쉽게 할 수 있다.

■ 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})
```

■ 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']
```

■ 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']
```

■ 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})
```

■ 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})
```


연습: Counter 모듈

실습 1: z_Counter.py

리스트나 문자열과 같은 시퀀스 자료형 안의 요소 중 값이 같은 것이 몇 개 있는지 확인한다.

```
11 a = 'gallahad'
12 text = list(a)
13 print('1)', text)
14
15 cnt = Counter(text)
16 print(f'2) {type(cnt)}\n{cnt}')
17 print('3)', cnt['e'], cnt['a'])
18
19 d = dict(cnt)
20 print('4)', type(d), d)
```

```
1) ['g', 'a', 'l', 'l', 'a', 'h', 'a', 'd']
2) <class 'collections.Counter'>
Counter({'a': 3, 'l': 2, 'g': 1, 'h': 1, 'd': 1})
3) 0 3
4) <class 'dict'> {'g': 1, 'a': 3, 'l': 2, 'h': 1, 'd': 1}
```

연습: Counter 모듈

실습 2: z_Counter.py

```
33 dic = {'eagle': 3, 'tiger': 1, 'horse': 2}
34 print(1, type(dic), dic)
35 # 1 <class 'dict'> {'eagle': 3, 'tiger': 1, 'horse': 2}
36
37 c = Counter(dic) # 1) dict 자료 지정
38 # Counter({'eagle': 3, 'horse': 2, 'tiger': 1})
39
40 print(2, c)
41 # 2 Counter({'eagle': 3, 'horse': 2, 'tiger': 1})
42
43 a = list(c.elements())
44 print(3, a)
45 # 3 ['eagle', 'eagle', 'eagle', 'tiger', 'horse', 'horse']
46
47 c = Counter(cats=2, dogs=3) # 2) keyword=수량 지정
48 print(4, c)
49 # 4 Counter({'dogs': 3, 'cats': 2})
50
51 a = list(c.elements())
52 print(5, a)
53 # 5 ['cats', 'cats', 'dogs', 'dogs', 'dogs']
```

```
1 <class 'dict'> {'eagle': 3, 'tiger': 1, 'horse': 2}
2 Counter({'eagle': 3, 'horse': 2, 'tiger': 1})
3 ['eagle', 'eagle', 'eagle', 'tiger', 'horse', 'horse']
4 Counter({'dogs': 3, 'cats': 2})
5 ['cats', 'cats', 'dogs', 'dogs', 'dogs']
```

연습: Counter 모듈

실습 3: z_Counter.py

```
64 a = Counter(a=4, b=3, c=5, d=-5)
65 b = Counter(a=-1, b=2, c=0, d=4)
66 c = Counter(a=-9, b=2, c=-1, d=4)
67
68 print(1, a.subtract(b))      # None. 반환값 없음
69 print(2, 'a=', a)
70 # a= Counter({'a': 5, 'c': 5, 'b': 1, 'd': -9})
71
72 print(3, 'b+c=', b+c)
73 # b+c= Counter({'d': 8, 'b': 4})
74 # + 연산은 두 Counter 객체에 있는 각 요소를 더한 것.
75 # 수가 음수이면 원소에서 사라진다.
76
77 print(4, 'a+b=', a+b)
78 # a+b= Counter({'c': 5, 'a': 4, 'b': 3})
79 # 'd' 원소가 없어졌다? 음수이므로...
80
81 print(5, 'a & b=', a & b)
82 # a & b= Counter({'b': 1})
```

```
1 None
2 a= Counter({'a': 5, 'c': 5, 'b': 1, 'd': -9})
3 b+c= Counter({'d': 8, 'b': 4})
4 a+b= Counter({'c': 5, 'a': 4, 'b': 3})
5 a & b= Counter({'b': 1})
```

05. collections 모듈

■ 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'
```

연습: namedtuple 모듈(1): 일반 tuple

실습 1: z_namedtuple.py

```
5 # --- 1. index로 요소 접근하기
6 print('\n1. access by tuple index -----')
7 bob = ('Bob', 30, 'male')
8 print('1) Representation:', bob)
9
10 jane = ('Jane', 29, 'female')
11 print('2) Field by index:', jane[0])      # index로 접근
12
13 print('3) Fields by index:')
14 for p in [bob, jane]:
15     #print('%s is a %d year old %s' % p)
16     print(f'{p[0]} is a {p[1]} year old {p[2]}')
```

```
1. access by tuple index -----
1) Representation: ('Bob', 30, 'male')
2) Field by index: Jane
3) Fields by index:
Bob is a 30 year old male
Jane is a 29 year old female
```

연습: namedtuple 모듈

실습 2: z_namedtuple.py

```
23 # namedtuple: 튜플 자료를 index로 접근하지 않고 항목에 붙은 이름으로 접근할 수 있게 한다.
24 Person = collections.namedtuple('Person', 'name age gender')
25 print('1) Type of Person:', type(Person), '| Person=', Person)
26
27 bob = Person(name='Bob', age=30, gender='male')
28 print('2)', type(bob))
29 print('3) bob=', bob.name, bob.age, bob.gender)
30
31 jane = Person(name='Jane', age=29, gender='female')
32 her = Person('Suji', 26, 'female')
33 print('4) her=', her.name, her.age, her.gender)
34
35 print('\n5) Fields by index:')
36 for p in [bob, jane, her]:
37     print('%s is a %d year old %s' % p)
38
39 print('\n6) Access by name:')
40 for p in [bob, jane, her]:
41     print(f'{p.name} is a {p.age} year old {p.gender}')
```

1) Type of Person: <class 'type'>
| Person= <class '__main__.Person'>
2) <class '__main__.Person'>
3) bob= Bob 30 male
4) her= Suji 26 female

5) Fields by index:
Bob is a 30 year old male
Jane is a 29 year old female
Suji is a 26 year old female

6) Access by name:
Bob is a 30 year old male
Jane is a 29 year old female
Suji is a 26 year old female

06

Lab: 텍스트 마이닝 프로그램

06. Lab: 텍스트 마이닝 프로그램

■ 실습 내용

- 앞에서 배운 딕셔너리와 Collections 모듈을 이용하여 텍스트 마이닝 프로그램을 만들어 보자.

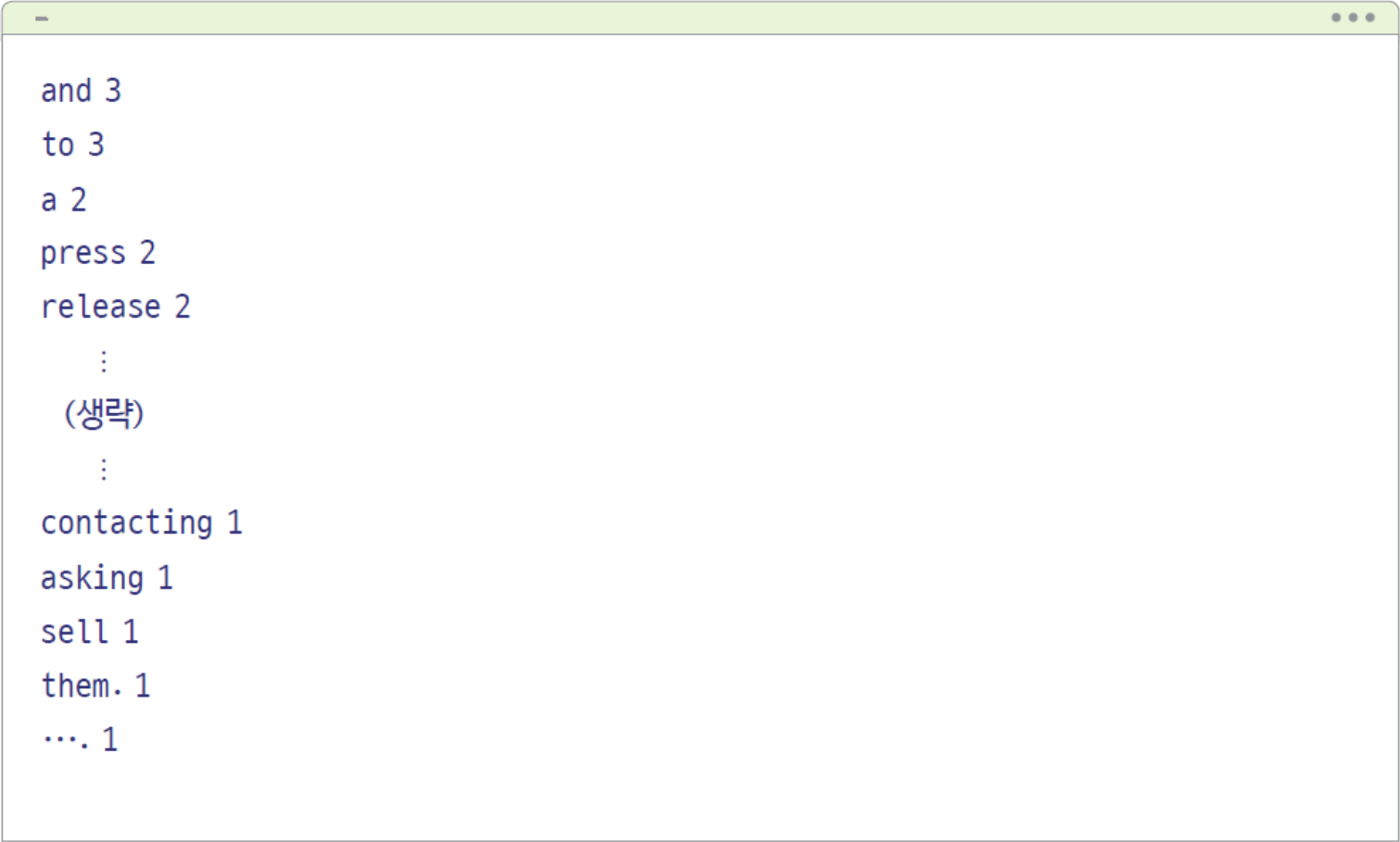
A press release is the quickest and easiest way to get free publicity. If well written, a press release can result in multiple published articles about your firm and its products. And that can mean new prospects contacting you asking you to sell to them. ...

- 이 프로그램을 작성하는 규칙은 다음과 같다.

- 문장의 단어 개수를 파악하는 코드를 작성한다.
- defaultdict 모듈을 사용한다.
- 단어의 출현 횟수를 기준으로 정렬된 결과를 보여 주기 위해 OrderedDict 모듈을 사용한다.

06. Lab: 텍스트 마이닝 프로그램

■ 실행 결과



```
and 3
to 3
a 2
press 2
release 2
:
(생략)
:
contacting 1
asking 1
sell 1
them. 1
... 1
```

06. Lab: 텍스트 마이닝 프로그램

textmining.py

■ 문제 해결

코드 7-8 textmining.py

```
1 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()
2
3 from collections import defaultdict
4
5 word_count = defaultdict(lambda: 0)           # Default 값을 0으로 설정
6 for word in text:
7     word_count[word] += 1
8
9 from collections import OrderedDict
10 for i, v in OrderedDict(sorted(word_count.items(), key=lambda t: t[1],
    reverse=True)).items():
11     print(i, v)                               역순으로. 큰 것(빈도수 높은 것)부터 정렬
```

Value 기준으로 정렬

06. Lab: 텍스트 마이닝 프로그램

■ 문제 해결 : [코드 7-8] 해석

- 1행은 text 변수에 문장을 넣고, 이를 소문자로 바꾼 후 단어 단위로 자르는 코드이다. 이를 위해 lower()와 split() 함수를 연속으로 사용하였다. 이 코드의 결과를 확인하기 위해 파이썬 셸에 다음과 같이 입력하면 리스트의 결과를 볼 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()
>>> print (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.', '...']
```

06. Lab: 텍스트 마이닝 프로그램

■ 문제 해결 : [코드 7-8] 해석

- 다음으로 이 리스트에서 각각의 단어가 몇 개 있는지 헤아리는 코드가 필요하다. 3~7행을 보면 defaultdict 모듈을 사용하여 딕셔너리의 키값을 설정 없이 단어가 출현할 때마다 `word_count[word] += 1`을 통해 단어의 수를 증가시키는 것을 확인할 수 있다.
- 다음으로 단어의 출현 횟수를 기준으로 정렬된 결과를 보여 주고 싶다면, 9~13행과 같이 `OrderedDict` 모듈을 사용하여 코드를 구성할 수 있다.

연습: 단순 데이터 마이닝

textmining.py

```
8  # 아래 문장에 나오는 단어의 수를 세는 프로그램
9  text = 'AA bbbb CCCC ss eeeeeeeeeee 11'
10 word_count = defaultdict(lambda: 0) # Default 0
11 for word in text:
12     word_count[word] += 1
13
14 from collections import OrderedDict
15 # t[1]이면 value를 기준으로 소팅했는데,
16 # reverse가 됨으로써 빈도수가 많은 단어 순으로 재정렬된다.
17 for i, v in OrderedDict(sorted(word_count.items(),
18                                key=lambda t: t[1], reverse=True)).items():
19     print(i, v)
```

e	10
	5
b	4
C	4
A	2
s	2
1	2

Thank You !