

# 문자열



## 목차

1. 문자열의 이해 *개념, 특성, 용도*
2. Lab: 단어 카운팅
3. 문자열 서식 지정

## 학습목표

- 문자열의 개념과 메모리 공간에 대해 이해한다.
- ✓ 문자열의 인덱싱과 슬라이싱에 대해 학습한다.
- 문자열의 연산과 문자열 함수에 대해 알아본다.
- 문자열의 형식을 정하여 출력하는 서식 지정에 대해 이해한다.

## 01 문자열의 이해

## 1. 문자열의 개념

- **문자열(string)**: 애플리케이션을 만들거나 데이터를 분석할 때 매우 중요하게 다루어지는 자료형 중 하나  
*문자열의 연속 = 문자열 (text data)*

- **문자열은 특징**: 시퀀스 자료형(sequence data type) *UI/data analysts*

- 시퀀스 자료형: 데이터를 순차적으로 메모리에 저장하는 형식의 데이터

*★ Immutable (변형 불가능)*

a = "abcde"

*print(a[0], a[1], a[2])*

a	0100 1001
b	0100 1010
c	0100 1011
d	0100 1100
e	0100 1101

그림 6-1 시퀀스 자료형

**TIP** 정확히 표현하면 메모리에 물리적인 순서대로 저장되는 것이 아니고 단지 우리 눈에 차례차례 저장되는 것처럼 보이는 것.

5

## 2. 문자열과 메모리 공간

- 컴퓨터는 문자를 직접 인식하지 않고 이진수로 변환하여 저장함  
- 문자가 1바이트이니, 2의 8승( $2^8$ )의 공간에 문자에 대한 정보를 저장하는 것.  
*1 byte (= 8 bit) =  $2^8$*

- **인코딩(encoding)**: 문자를 처리하기 위해 이진수로 변환되는 표준 규칙

- ① 컴퓨터는 문자를 직접 인식하지 못한다.
- ② 컴퓨터는 문자를 숫자로 변환하여 인식한다.
- ③ 사람들은 문자를 숫자로 변환하기 위한 규칙을 만들었다.
- ④ 일반적으로 1개의 영문자를 1바이트, 즉 2의 8승( $2^8$ ) 정도의 공간에 저장한다.

*UTF-8*

7

## 2. 문자열과 메모리 공간

*Strings Memory Allocation*

- 문자열을 저장하기 위해 영문자 한 글자당 1바이트의 메모리 공간을 사용  
*(UTF-8 인코딩 가능)*

```
>>> import sys # sys 모듈을 호출
>>> print(sys.getsizeof("a"), sys.getsizeof("ab"), sys.getsizeof("abc"))
50 51 52 # "a", "ab", "abc" 각각의 메모리 크기 출력
```

*메모리 크기는 바이트 단위로 반환*

- **sys.getsizeof**: 특정 변수(또는 값)의 메모리 공간을 측정하는 함수로 a, ab, abc의 메모리 크기가 각각 50, 51, 52로 1씩 증가하는 것을 알 수 있음.

*"a" : 50 byte*

*"ab" : 51*

*"abc" : 52*

*Question) Why 50?*

*"a" 객체가 가지는 데이터*

*① 문자열 a.*

*+*

*② 객체 데이터 (meta data)*

*(문자열의 길이, 인코딩 정보, 카운트 정보....)*

6

## 2. 문자열과 메모리 공간

- 이러한 규칙을 이용하여 숫자와 문자를 맵핑하는 것이 운영체제와 인터프리터의 역할 중 하나이며, [그림 6-2]와 같은 형태로 저장됨.

000	(nul)	016	(dle)	032	sp	048	0	064	0	080	F	096	.	112	p
001	(soh)	017	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	(stx)	018	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	(etx)	019	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	(eot)	020	(dc4)	036	\$	052	4	068	D	084	T	100	d	116	t
005	(enq)	021	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	(ack)	022	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	(bel)	023	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	(bs)	024	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	(vt)	027	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	(np)	028	(fs)	044	,	060	<	076	L	092	\	108	l	124	
013	(cr)	029	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	(so)	030	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	(si)	031	(us)	047	/	063	?	079	O	095	_	111	o	127	o

그림 6-2 UTF-8의 유니코드(출처: Nicolas Bouliaue)

*인코딩 규칙에 따른 숫자와 문자 매핑 정보*

*A ↔ 65*

*문자 A는 숫자 코드 65로 표현됨*

*UTF-8이 인코딩*

8

### 3. 문자열의 인덱싱

- 인덱싱(indexing): 리스트처럼 글자 하나하나가 상대적인 주소(offset)를 가지는데, 이 주소를 사용해 저장된 값을 가져오는 것.



```
>>> a = "abcde"
>>> print(a[0], a[4]) # a 변수의 0번째, 4번째 주소에 있는 값
a e
>>> print(a[-1], a[-5]) # a 변수의 오른쪽에서 0번째, 4번째 주소에 있는 값
e a
```

*print(a[0], a[-1])*

9

### 4. 문자열의 슬라이싱

- 슬라이싱(slicing): 문자열의 주소 값을 이용해 문자열의 부분 값을 추출해내는 기법

```
>>> a = "TEAMLAB MOOC, AWESOME Python"
>>> print(a[0:6], " AND ", a[-9:]) # a 변수의 0부터 5까지, -9부터 끝까지
TEAML AND ME Python
>>> print(a[:]) # a 변수의 처음부터 끝까지
TEAMLAB MOOC, AWESOME Python
>>> print(a[-50:50]) # 범위를 넘어갈 경우 자동으로 최대 범위를 지정
TEAMLAB MOOC, AWESOME Python
>>> print(a[::2], " AND ", a[::-1])
TALBMO, AEOEPto AND nohtyP EMOSEWA ,COOM BALMAET
```

10

### 5. 문자열의 연산

- 기본적으로 문자열의 연산은 리스트 연산과 같음.

```
>>> a = "TEAM"
>>> b = "LAB"
>>> print(a + " " + b) # 덧셈으로 a와 b 변수 연결하기
TEAM LAB
>>> print(a * 2 + " " + b * 2) # 곱하기로 반복 연산 가능
TEAMTEAM LABLAB
>>> if 'A' in a: print(a) # 'A'가 a에 포함되었는지 확인
... else: print(b)
TEAM
```

*Concatenation +*

*repetition \**

*in 확인 (Inclusion)*

*if문*

*else문*

11

### 5. 문자열의 연산

- 덧셈 연산은 모든 변수가 문자열일 경우 텍스트 붙이기(concatenate)가 이루어짐
- 여기서 자주 하는 실수 중 하나가 print() 함수에서 정수형과 문자열을 같이 보여주려고 할 때 발생.
- 예) 다음과 같이 코드를 작성하면 문자열과 정수형의 연산으로 인식하여 덧셈 연산이 실행되지 않음.

```
>>> int_value = 2
>>> print("결과는" + int_value)
```

*TypeError: can only concatenate str (not "int") to str*

*다중 발생*

*정수형과 문자열은 + 안됨*

*str(int\_value)*

12

## 6. 문자열 함수

표 6-1 문자열 함수

함수명	기능
len()	문자열의 문자 개수를 반환
upper()	대문자로 변환
lower()	소문자로 변환
title()	각 단어의 앞글자만 대문자로 변환 <i>I Love You</i>
capitalize()	첫 문자를 대문자로 변환 <i>I love you</i>
count('찾을 문자열')	'찾을 문자열'이 몇 개 들어있는지 개수 반환
find('찾을 문자열')	'찾을 문자열'이 왼쪽 끝부터 시작하여 몇 번째에 있는지 반환 <i>i love you</i>
rfind('찾을 문자열')	find() 함수와 반대로 '찾을 문자열'이 오른쪽 끝부터 시작하여 몇 번째에 있는지 반환
startswith('찾을 문자열')	'찾을 문자열'로 시작하는지 여부 반환
endswith('찾을 문자열')	'찾을 문자열'로 끝나는지 여부 반환
strip()	좌우 공백 삭제
rstrip()	오른쪽 공백 삭제
lstrip()	왼쪽 공백 삭제
split()	문자열을 공백이나 다른 문자로 나누어 리스트로 반환 <i>구분자 (separator)</i>
isdigit()	문자열이 숫자인지 여부 반환
islower()	문자열이 소문자인지 여부 반환
isupper()	문자열이 대문자인지 여부 반환

13

## 6. 문자열 함수

- **upper()** 함수: 문자열을 대문자로 변환
- **lower()** 함수: 문자열을 소문자로 변환

```
>>> title = "TEAMLAB X Inflearn"
>>> title.upper()           # title 변수를 모두 대문자로 변환
'TEAMLAB X INFLEARN'
>>> title.lower()          # title 변수를 모두 소문자로 변환
'teamlab x inflearn'
```

- **title()** 함수: 영어신문의 헤드라인처럼 각 단어의 앞글자만 대문자로 바꾸는 함수
- **capitalize()** 함수: 첫 번째 글자만 대문자로 바꾸는 함수

```
>>> title = "TEAMLAB X Inflearn"
>>> title.title()           # title 변수의 각 단어의 앞글자만 대문자로 변환
'Teamlab X Inflearn'
>>> title.capitalize()     # title 변수의 첫 번째 글자만 대문자로 변환
'Teamlab x inflearn'
```

14

## 6. 문자열 함수

- **count()** 함수: 해당 문자열에서 특정 문자가 포함된 개수를 반환
- **isdigit()** 함수: 해당 문자열이 숫자인지를 True 또는 False 값으로 반환
- **startswith()** 함수: 해당 문자열로 시작하는지를 True 또는 False 값으로 반환

```
>>> title = "TEAMLAB X Inflearn"
>>> title.count("a")        # title 변수에 'a'의 개수 반환
1
>>> title.upper().count("a") # title 변수를 대문자로 만든 후, 'a'의 개수 반환
0
>>> title.isdigit()        # title 변수의 문자열이 숫자인지 여부 반환
False
>>> title.startswith("a")  # title 변수가 'a'로 시작하는지 여부 반환
False
```

15

## 6. 문자열 함수

여기서 잠깐 문자열 표현과 특수문자

파이썬에서 문자열을 표현할 때 작은따옴표나 큰따옴표를 사용한다. 하지만 다음과 같이 아포스트로피(')가 문장에 들어가면 작은따옴표를 사용하기 어렵다. 만약 작은따옴표로 문자열을 표현한다면 인터프리터는 이 문자가 제대로 닫히지 않았다고 판단하고 오류를 출력할 것이다.

It's OK.

또 다른 문제로는 다음과 같은 줄바꿈에 대한 것이다. 이러한 경우에도 문자열로 표현하기 어렵다.

"It's OK."  
"I'm Happy."  
See you.

이러한 문제를 해결하기 위해 파이썬에서는 여러 가지 기능을 지원한다. 먼저 문자열 자체에 작은따옴표나 큰따옴표가 들어가 있는 경우이다. 이 경우 가장 쉬운 방법은 작은따옴표가 들어간 문자열은 큰따옴표로 선언하고, 큰따옴표가 들어간 문자열은 작은따옴표로 선언하는 것이다. 매우 쉽지만 유용한 방법이다. 코드로 작성하면 다음과 같다.

a = "It's OK."

16

## 6. 문자열 함수

다음으로 파이썬의 특수문자 기능을 사용하는 것이다. 이 특수문자는 문자열에서 표현하기 어려운 여러 문자를 표현할 수 있도록 도와준다. 기본적으로 역슬래시(\) 기호를 사용하는데, 윈도우에서는 원(w) 표시이다. 파이썬의 특수문자는 [표 6-2]와 같으며 일반적으로 다른 프로그래밍 언어에서도 사용하는 특수문자들이다.

표 6-2 파이썬의 특수문자

특수문자	기능	특수문자	기능
\[Enter]	다음 줄과 연속임을 표현	\b	백스페이스
\	\ 문자 자체	\n	줄바꿈기
\'	' 문자	\t	[Tab] 키
\"	" 문자	\e	[Esc] 키

이러한 특수문자를 사용해 다음과 같이 아포스트로피(') 문자를 사용할 수 있다.

```
a = "It\'s OK."
```

## 6. 문자열 함수

두 줄 이상의 표현도 마찬가지로 가능하며, 두 가지로 표현할 수 있다. 하나는 큰따옴표(")나 작은따옴표(')를 3개로 연결하는 방법이다. 다음과 같이 선언하면 a변수에 여러 줄의 문자열을 저장할 수 있다.

```
a = """
It's Ok.
I'm Happy.
See you."""
```

17

18

## 02 Lab: 단어 카운팅

### LAB 단어 카운팅

#### 1. 실습 내용

- 팝그룹 비틀스의 라는 노래에서 'Yesterday'라는 단어가 몇 번 나오는지 맞는 단어 카운팅 프로그램 만들기
- 아래 코드를 그대로 입력하면 리스트 형태로 각 줄의 내용을 가져올 수 있음.

```
f = open("yesterday.txt", 'r')
yesterday_lyric = f.readlines()
f.close()
```

20

## 2. 실행 결과

## [실행결과]

```
Number of a Word 'Yesterday' 9
```

21

## 3. 문제 해결

## [코드 6-1]

```
1 f = open("yesterday.txt", 'r')
2 yesterday_lyric = f.readlines()
3 f.close()
4
5 contents = ""
6 for line in yesterday_lyric:
7     contents = contents + line.strip() + "\n"
8
9 n_of_yesterday = contents.upper().count("YESTERDAY")
10 print("Number of a Word 'Yesterday'" , n_of_yesterday)
```

22

- upper() 함수는 contents 변수에 값 자체를 변경하는 것이 아니라, 변경된 값을 반환해주는 함수이므로 contents.upper().count("YESTERDAY")처럼 함수를 붙여 써도 됨.

```
>>> title = "teamlab"
>>> title
'teamlab'
>>> title.upper()
'TEAMLAB'
>>> title
'teamlab'
```

23

## 03 문자열 서식 지정

*String Formatting*

22

## 1. 서식 지정의 개념

- **서식 지정(formatting):** print() 함수를 특정한 형식에 맞추어 결과를 출력해야 하는 것.
  - 기본적으로 print() 함수는 변수 또는 값을 콤마(,)로 띄어쓰기 하여 출력함.
  - 사용하다 보면 특정한 형식에 맞추어 결과를 출력해야 하는 경우도 발생
  - 특히 엑셀을 사용할 때 통화 단위, 세 자리 숫자 단위로 띄어쓰기, % 출력 등 다양한 형식에 맞춰 출력할 일이 생김.

25

## 2. % 서식과 format() 함수

- 문자열의 서식을 설정할 때 print() 함수는 기본적인 출력 형식 외 % 서식과 format() 함수를 구문으로 사용하여 출력 양식을 지정할 수 있음.

[코드 6-2]

```
1 print(1, 2, 3)
2 print("a" + " " + "b" + " " + "c")
3 print("%d %d %d" % (1, 2, 3))
4 print("{} {} {}".format("a", "b", "c"))
```

*% formatting*  
*format()*  
*{ } placeholder를 이용*

[실행결과]

```
1 2 3
a b c
1 2 3
a b c
```

26

## 2. % 서식과 format() 함수

- 서식을 지정하여 출력할 때의 장점
  - ① 데이터와 출력 형식을 분류할 수 있음
  - ② 데이터를 형식에 따라 다르게 표현할 수 있음.

[코드 6-3]

```
1 print('%s %s' % ('one', 'two'))
2 print('%d %d' % (1, 2))
```

[실행결과]

```
one two
1 2
```

27

## 2. % 서식과 format() 함수

### 2.1 % 서식

- % 서식의 출력 양식 형태:

```
'%자료형 % (값)'
```

[코드 6-4]

```
1 print("I eat %d apples." % 3)
2 print("I eat %s apples." % "five")
```

[실행결과]

```
I eat 3 apples.
I eat five apples.
```

28

## 2. % 서식과 format( ) 함수

- %d는 정수형의 변수를, %s는 문자열의 변수를 할당 받을 수 있음.  
☞ %d에는 '3'이, %s에는 'five'가 대응됨.

- 변수의 자료형에 따라 다양하게 설정할 수 있음.

표 6-3 변수의 자료형에 따른 서식

서식	설명
%s	문자열(string)
%c	문자 1개(character)
%d	정수(integer)
%f	실수(floating-point)
%o	8진수
%x	16진수
%%	문자 % 자체

29

## 2. % 서식과 format( ) 함수

- % 서식은 1개 이상의 값도 할당할 수 있음.  
☞ % 뒤에 괄호를 넣어 그 안에 순서 대로 값을 입력하면 됨.

```
>>> print("Product: %s, Price per unit: %f." % ("Apple", 5.243))
Product: Apple, Price per unit: 5.243000.
```

- 직접 값을 넣지 않고 number와 day 같은 변수명을 넣어도 문제없이 실행됨.

### [코드 6-5]

```
1 number = 3
2 day = "three"
3 print("I ate %d apples. I was sick for %s days." % (number, day))
```

### [실행결과]

```
I ate 3 apples. I was sick for three days.
```

30

## 2. % 서식과 format( ) 함수

### 2.2 format( ) 함수

- format( ) 함수의 서식 지정 형태

```
"{자료형}".format(인수)
```

```
>>> print("I'm {0} years old.".format(20))
I'm 20 years old.
```

- format( ) 함수를 사용한 가장 기본적인 표현 형태
- 숫자 20이 {0}에 할당되어 출력 → 기존 % 서식과 비교하면 자료형을 바로 지정해 주지 않고 순서대로 변수가 할당되는 장점이 있음.

31

## 2. % 서식과 format( ) 함수

- format( ) 함수는 % 서식처럼 변수의 이름을 사용하거나 변수의 자료형을 따로 지정하여 출력함

### [코드 6-6]

```
1 age = 40; name = 'Sungchul Choi'
2 print("I'm {0} years old.".format(age))
3 print("My name is {0} and {1} years old.".format(name, age))
4 print("Product: {0}, Price per unit: {1:.2f}.".format("Apple", 5.243))
```

### [실행결과]

```
I'm 40 years old.
My name is Sungchul Choi and 40 years old.
Product: Apple, Price per unit: 5.24.
```

32



### 3. 패딩

- **패딩(padding)**: 여유 공간을 지정하여 글자 배열을 맞추고 소수점 자릿수를 맞추는 기능

#### 3.1 % 서식의 패딩

```
>>> print("%10d" % 12)
      12
>>> print("%-10d" % 12)
12
```

- 실수에서도 자릿수와 소수점 자릿수를 지정할 수 있음.

```
>>> print("%10.3f" % 5.94343) # 10자리를 확보하고 소수점 셋째 자리까지 출력
5.943
>>> print("%10.2f" % 5.94343) # 10자리를 확보하고 소수점 둘째 자리까지 출력
5.94
>>> print("%-10.2f" % 5.94343)
5.94
```

33

### 3. 패딩

#### 3.2 format( ) 함수의 패딩

- format( ) 함수를 이용한 패딩도 % 서식과 비슷

```
>>> print("{0:>10s}".format("Apple"))
      Apple
>>> print("{0:<10s}".format("Apple"))
Apple
```

- 실수에서도 자릿수와 소수점 자릿수를 지정할 수 있음.

```
>>> "{1:>10.5f}".format("Apple", 5.243)
'      5.24300.'
>>> "{1:<10.5f}".format("Apple", 5.243)
'5.24300      .'
```

34

### 3. 패딩

#### 여기서 잠깐! 네이밍(naming)

서식 지정으로 print() 함수를 활용해 출력할 때 한 가지 더 알아야 하는 점은 변수명을 서식에 할당할 수 있는 네이밍이라는 기능이 있다는 것이다. 다음 코드에서 보듯이 기존 번호나 순서대로 자료형에 대응하는 것이 아닌, 'name'이나 'price' 처럼 특정 변수명을 사용하여 출력값에 직접 할당할 수 있다. 특히 한 번에 출력해야 하는 변수가 많을 때 개발자 입장에서 변수의 순서를 헷갈리지 않고 사용할 수 있다는 장점이 있다.

```
>>> print("Product: %(name)5s, Price per unit: %(price)5.5f." %
      {"name": "Apple", "price": 5.243})
Product: Apple, Price per unit: 5.24300.
>>> print("Product: {name:>5s}, Price per unit: {price:5.5f}".format(name="A
      pple", price=5.243))
Product: Apple, Price per unit: 5.24300.
```

35

## 자료구조



# 목차

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

# 학습목표

- 파이썬에서의 자료구조에 대해 이해한다.
- 스택, 큐, 튜플, 세트에 대해 학습한다.
- 파이썬에서의 딕셔너리에 대해 알아본다.
- collections 모듈에 대해 이해한다.

## 01 자료구조의 이해

### 1. 자료구조의 개념

- 자료구조(data structure): 데이터의 특징을 고려하여 저장하는 방법
- 실생활 속 데이터 저장 사례: 전화번호부



(a) 전화번호부

(b) 휴대전화의 연락처

그림 7-1 실생활 속 자료구조

## 2. 파이썬에서의 자료구조

표 7-1 파이썬에서 제공하는 자료구조

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

41

## 02 스택과 큐

### 1. 스택

- **스택(stack):** 컴퓨터공학과 학생들이 전공을 시작하면서 처음 배우는 자료구조로, 자료구조의 핵심 개념 중 하나
- **스택 자료구조(stack data structure):** 4, 10과 같은 데이터를 저장하는 공간, 리스트와 비슷하지만 저장 순서가 바뀌는 형태
- **푸시(push):** 스택에 데이터를 저장하는 것
- **팝(pop):** 데이터를 추출하는 것

### 1. 스택

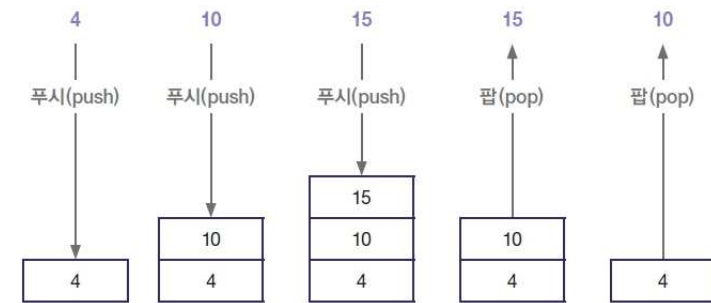


그림 7-2 스택

- 스택은 어떤 상황에서 사용할 수 있을까?
  - 택배 수화물을 저장하는 방식: 먼저 배달해야 하는 수화물은 트럭의 입구 쪽에, 나중에 배달해야 하는 수화물은 트럭의 안쪽에 넣어야 함



43

44

## 1. 스택

- 파이썬에서는 리스트를 사용하여 스택을 구현

☞ 리스트라는 저장 공간을 만든 후 `append()` 함수로 데이터를 저장(push)하고 `pop()` 함수로 데이터를 추출(pop)함.

*Last In First Out : LIFO*

```
>>> 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
```

45

## 1. 스택

- 입력한 텍스트를 역순으로 추출하는 프로그램

[코드 7-1]

```
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])
```

[실행결과]

```
Input a word: PYTHON
['P', 'Y', 'T', 'H', 'O', 'N']
['N', 'O', 'H', 'T', 'Y', 'P']
NOHTYP
```

← 사용자 입력(PYTHON)

46

## 2. 큐

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

표 7-3 대학생 인적사항

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

47

## 2. 큐

- 파이썬에서 큐를 구현하는 법

```
>>> 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
```

- 기본적으로 스택의 구현과 같은데, `pop()` 함수를 사용할 때 인덱스가 0번째인 값을 쓴다는 의미로 `pop(0)`을 사용하면 됨.
- `pop()` 함수가 리스트의 마지막 값을 가져온다고 했을 때 `pop(0)`은 맨 처음 값을 가져온다는 뜻.

48

## 03 튜플과 세트

## 1. 튜플

- 튜플(tuple): 리스트와 같은 개념이지 만 값을 변경하는 것이 불가능한 리스트로의 자료구조

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

- 만약 튜플의 값을 변경하고자 한다면 다음과 같이 오류가 발생함.

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

## 2. 세트

- 세트(set): 값을 순서 없이 저장하되 중복을 불허하는 자료형

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

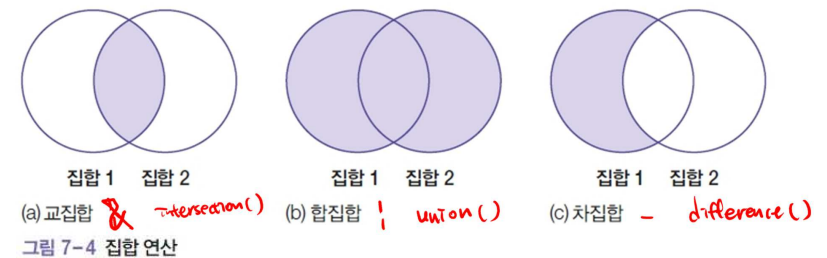
- 세트는 튜플과 다르게 삭제나 변경이 가능함.

```
>>> 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()
```

## 2. 세트

- 세트를 지원하는 함수

- 원소 하나를 추가하는 add()
- 원소 하나를 제거하는 remove() 또는 discard()
- 새로운 리스트를 그대로 추가하는 update()
- 모든 변수를 지우는 clear()



## 2. 세트

- 파이썬은 교집합, 합집합, 차집합 연산을 모두 지원함.

```
>>> s1 = set([1, 2, 3, 4, 5])
>>> s2 = set([3, 4, 5, 6, 7])
>>>
>>> s1.union(s2)           # s1과 s2의 합집합
{1, 2, 3, 4, 5, 6, 7}
>>> s1 | s2                # set([1, 2, 3, 4, 5, 6, 7])
{1, 2, 3, 4, 5, 6, 7}
>>> s1.intersection(s2)   # s1과 s2의 교집합
{3, 4, 5}
>>> s1 & s2                # set([3, 4, 5])
{3, 4, 5}
>>> s1.difference(s2)      # s1과 s2의 차집합
{1, 2}
>>> s1 - s2               # set([1, 2])
{1, 2}
```

53

## 2. 세트

표 7-2 세트의 집합 연산

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

- 합집합:** 두 집합의 중복 값을 제거하고 합치는 연산
- 교집합:** 두 집합 양쪽에 모두 포함된 값만 추출하는 연산
- 차집합:** 앞에 있는 집합 s1의 원소 중 s2에 포함된 원소를 제거하는 연산

54

## 04 딕셔너리

## 1. 딕셔너리의 개념

### ■ 딕셔너리(dictionary)

- 파이썬에서 가장 많이 사용하는 자료구조
- 영어사전에서 검색을 위해 영어 단어들을 저장해 놓은 방식과 비슷함.
  - 영어사전에서 각 단어를 검색할 수 있도록 색인(index)을 만들어 놓고 색인을 통해 그 단어를 찾아 의미를 파악함.
- 파이썬의 딕셔너리 구조에서는 데이터의 유일한 구분자인 키(key)라는 이름으로 검색할 수 있게 하고, 실제 데이터를 값(value)이라는 이름과 쌍으로 저장하여 프로그래머가 데이터를 쉽게 찾을 수 있도록 함.

key, value 쌍으로 저장  
key: 데이터의 <sup>유일한</sup> 식별자 (ID: ID number), 중복 X  
value: 이와 연결된 데이터, 중복 O

56

## 1. 딕셔너리의 개념

- [표 7-3]의 대학생 인적사항에서 학번이 나머지 정보를 구분하는 키.
- 학번을 키로 하여 이름 · 생년월일 · 주소를 리스트 형태로 저장한 다음 한 번에 검색할 수 있는 형태가 되면 학번을 이용해 다른 정보에 쉽게 접근할 수 있음.

표 7-3 대학생 인적사항

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

57

## 2. 파이썬에서의 딕셔너리

- 딕셔너리의 선언: 중괄호 { }를 사용하여 키와 값을 쌍으로 구성

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

예) [표 7-4]처럼 학번을 키로 사용하고, 이름을 값으로 지정할 수 있음.

표 7-4 키와 값의 샘플

학번(키)	이름(값)
20140012	Sungchul
20140059	Jiyong
20150234	Jaehong
20140058	Wonchul

58

## 2. 파이썬에서의 딕셔너리

- [표 7-4]의 정보를 간단히 파이썬으로 표현하기

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

- 해당 변수에서 특정 값을 호출하는 방법

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

- 재할당과 데이터 추가

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

59

## 3. 딕셔너리 함수

- 파이썬에서는 딕셔너리를 쉽게 사용할 수 있도록 다양한 함수를 제공.

- 국가명과 국가 전화번호를 묶어 보여주는 코드 작성

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

- 딕셔너리 변수 안의 키와 값을 출력하는 함수

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

60

### 3. 딕셔너리 함수

- 값을 출력하기 위해서 values() 함수 사용

```
>>> 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])          모든
```

- 키-값 쌍을 모두 보여주기 위해서 items() 함수 사용

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

61

### 3. 딕셔너리 함수

- for문과 함께 딕셔너리를 사용하여 키-값 쌍을 화면에 출력하기.

```
>>> 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
```

62

### 3. 딕셔너리 함수

- if문을 사용하여 특정 키나 값이 해당 변수에 포함되어 있는지 확인하기

```
>>> "Korea" in country_code.keys()        # 키에 "Korea"가 있는지 확인
True
>>> 82 in country_code.values()           # 값에 82가 있는지 확인
True
```

63

## 05 collections 모듈

64



## 1. deque 모듈

- **deque 모듈:** 스택과 큐를 모두 지원하는 모듈
- **deque의 사용:** 리스트와 비슷한 형식으로 데이터를 저장해야 함

```
>>> 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])
```

- **collections 모듈:** 리스트, 튜플, 딕셔너리 등을 확장하여 제작된 파이썬의 내장 모듈
- collections 모듈은 deque, OrderedDict, defaultdict, Counter, namedtuple 등을 제공함.
- 각 자료구조를 호출하는 코드

```
from collections import deque    스택과 큐의 기능을 모두 가진
from collections import OrderedDict    입력된 순서를 지키는 딕셔너리
from collections import defaultdict    기본값을 지정하여 딕셔너리를 생성
from collections import Counter    개수를 셀 때
from collections import namedtuple    이름이 있는 튜플로 필드명을 지정하여 다음
```

65

66

## 1. deque 모듈

- `deque_list.pop()`을 수행시키면 오른쪽 요소부터 하나씩 추출됨

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

67

## 1. deque 모듈

- deque에서 **큐**는 어떻게 사용할 수 있을까?
  - ~~pop(0)~~을 입력하여 실행하면 deque에서는 작동하지 않음.
  - deque는 appendleft() 함수로 새로운 값을 왼쪽부터 입력하도록 하여 먼저 들어간 값부터 출력될 수 있도록 함.

```
>>> 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])
```

68

## 1. deque 모듈

- deque 모듈의 장점:
  - ✓ deque는 연결 리스트의 특성을 지원함.
    - 연결 리스트(linked list): 데이터를 저장할 때 요소의 값을 한 쪽으로 연결한 후, 요소의 다음 값의 주소 값을 저장하여 데이터를 연결하는 기법

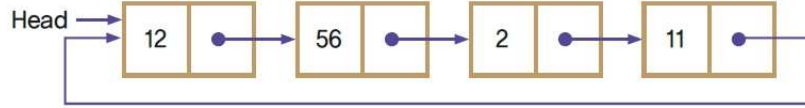


그림 7-5 연결 리스트의 형태

69

## 1. deque 모듈

- ✓ 연결 리스트는 다음 요소의 주소 값을 저장하므로 데이터를 원형으로 저장할 수 있음.
- ✓ 마지막 요소에 첫 번째 값의 주소를 저장시켜 해당 값을 찾아갈 수 있도록 연결시킴.

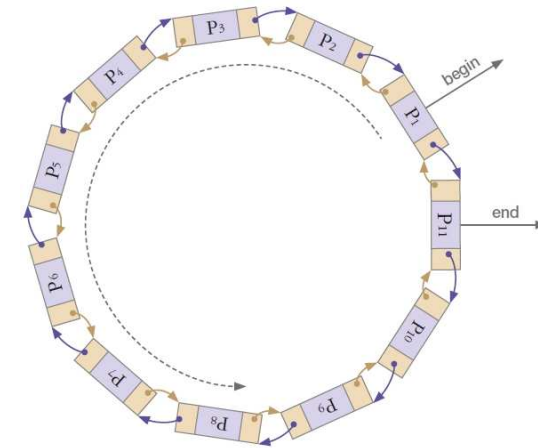


그림 7-6 원형 연결 리스트의 형태

70

## 1. deque 모듈

- rotate( )**: 기존 deque에 저장된 요소들 값의 인덱스를 바꾸는 기법  
*rotate(n): 요소들을 n번 순 이동*

```
>>> 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])
```

71

## 1. deque 모듈

- reversed( ) 함수**: 기존과 반대로 데이터를 저장하는 기법.

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

- extend( ) 함수**: 리스트가 통째로 오른쪽으로 추가되는 기법
- extendleft( ) 함수**: 리스트가 통째로 왼쪽으로 추가되는 기법

```
>>> 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])
```

72

## 2. OrderedDict 모듈

- OrderedDict 모듈: 순서를 가진 딕셔너리 객체

### [코드 7-2]

```
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)
```

### [실행결과]

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

73

## 2. OrderedDict 모듈

- 딕셔너리: 순서를 보장하지 않는 객체로, 딕셔너리 파일을 저장하면 키는 저장 순서와 상관없이 저장됨.

### [코드 7-3]

```
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)
```

### [실행결과]

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

74

## 2. OrderedDict 모듈

- 딕셔너리로 데이터 처리 시 키나 값으로 데이터를 정렬할 때 OrderedDict 모듈을 가장 많이 사용함.
- 키를 이용하여 주민등록번호를 번호 순서대로 정렬한 후 데이터를 출력하는 경우의 코드

### [코드 7-4]

```
1 def sort_by_key(t): 정렬기준(특목이 첫번째로 오니(2)은 기본으로 정렬)
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 정해진 결과 순서로 유지
11
12 for k, v in OrderedDict(sorted(d.items(), key=sort_by_key)).items(): 정렬
13     print(k, v)
```

75

## 2. OrderedDict 모듈

### [실행결과]

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

- 딕셔너리 값인 변수 d를 리스트 형태로 만든 다음, sorted() 함수를 사용하여 정렬.
- sorted(d.items(), key=sort\_by\_key)의 코드만 따로 실행하면 다음처럼 정렬되어 이차원 형태로 값이 출력됨.

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

- 만약, 값을 기준으로 정렬한다면 [코드 7-4]의 1행과 2행을 다음처럼 바꾸면 됨.

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

76

### 3. defaultdict 모듈

- 딕셔너리의 변수를 생성할 때 키에 기본값을 지정하는 방법
  - 새로운 키를 생성할 때 별다른 조치 없이 새로운 값을 생성할 수 있음.
  - 실제 딕셔너리에서는 [코드 7-5]처럼 키를 생성하지 않고 해당 키의 값을 호출하려고 할 때 오류가 발생

[코드 7-5]

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

[실행결과]

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

77

### 3. defaultdict 모듈

- defaultdict 모듈은 어떻게 사용할까?

[코드 7-6]

새로운 키에 접근할 때 기본값을 자동으로 생성

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

[실행결과]

0

78

### 3. defaultdict 모듈

- defaultdict의 초기 값은 리스트 형태로도 설정할 수 있음

[코드 7-7]

```
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])]
```

[실행결과]

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

79

### 4. Counter 모듈

- 시퀀스 자료형의 데이터 값의 개수를 딕셔너리 형태로 반환하는 방법

```
>>> 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
```

80

## 4. Counter 모듈

- Counter를 이용하면 각 문자의 개수 세는 작업을 매우 쉽게 할 수 있음

```
>>> 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})
```

81

## 4. Counter 모듈

- 딕셔너리 형태로 Counter 객체를 생성하는 방법

```
>>> 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
```

82

## 4. Counter 모듈

- 키워드 형태의 매개변수를 사용하여 Counter를 생성하는 방법

```
>>> 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']
```


- Counter는 기본 사칙연산(덧셈, 뺄셈, 논리 연산 등)을 지원

```
>>> 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})
```

83

## 4. Counter 모듈

```
>>> 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})
```

- + 기호: 두 Counter 객체에 있는 각 요소를 더한 것,
-  기호: 두 객체에 같은 값이 있을 때, 즉 교집합의 경우에만 출력함.
- | 기호: 두 Counter 객체에서 하나가 포함되어 있다면, 그리고 좀 더 큰 값이 있다면 그 값으로 합집합을 적용함.

84

## 5. namedtuple 모듈

- 튜플의 형태로 데이터 구조체를 저장하는 방법
  - 특정 데이터의 규정된 정보를 하나의 튜플 형태로 구성해 손쉽게 사용할 수 있는 자료구조

```
>>> from collections import namedtuple
>>>
>>> Point = namedtuple('Point', ['x', 'y'])  namedtuple 생성
>>> p = Point(11, y=22)  인스턴스 생성
>>> p
Point(x=11, y=22)
>>> p.x, p.y  필드 이름으로 접근
(11, 22)
>>> print(p[0] + p[1])  인덱스로 접근
33
```

85

## 06

### Lab: 텍스트 마이닝 프로그램

텍스트 데이터로 무엇 의미하는 객체 or 정보를 추출

#### LAB 텍스트 마이닝 프로그램

##### 1. 실습 내용

- 텍스트 마이닝(text mining): 텍스트를 분석하여 의미 있는 결과 도출하는 과정

- 이번 Lab에서는 다음 문장에 있는 단어의 개수를 파악해본다.

```
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. ...
```

- 프로그램 작성하는 규칙

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

87

#### LAB 텍스트 마이닝 프로그램

##### 2. 실행 결과

[실행결과]

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

88

## 3. 문제 해결

## [코드 7-8]

```

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)

```

- 1행: text 변수에 원하는 문장을 넣고, 이를 소문자로 바꾼 후 단어 단위로 자르는 코드
  - 이를 위해 lower()와 split() 함수를 연속으로 사용함
  - 이 코드의 결과를 확인하기 위해 파이썬 셸에 다음과 같이 입력하면 리스트의 결과를 볼 수 있음.

```

>>> 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.', '....']

```