

# 데이터 과학을 위한 파이썬 프로그래밍

2판



# Chapter 06

## 문자열



# 목차

1. 문자열의 이해
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
- **문자열은 특징**: 시퀀스 자료형(sequence data type)
  - 시퀀스 자료형: 데이터를 순차적으로 메모리에 저장하는 형식의 데이터

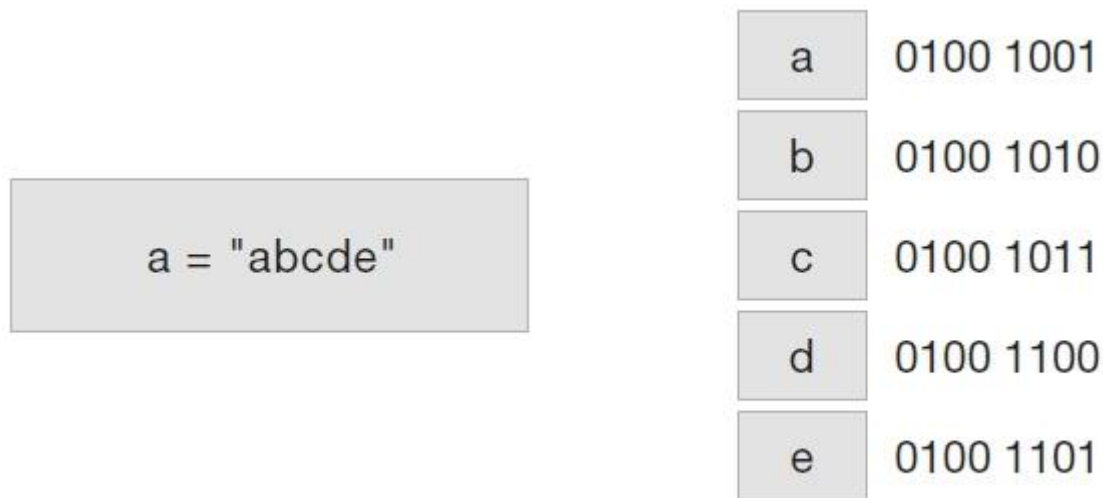


그림 6-1 시퀀스 자료형

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

## 2. 문자열과 메모리 공간

- 문자열을 저장하기 위해 영문자 한 글자당 1바이트의 메모리 공간을 사용

```
>>> 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
```

- **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.

## 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.
  
- **인코딩(encoding):** 문자를 처리하기 위해 이진수로 변환되는 표준 규칙
  - ❶ 컴퓨터는 문자를 직접 인식하지 못한다.
  - ❷ 컴퓨터는 문자를 숫자로 변환하여 인식한다.
  - ❸ 사람들은 문자를 숫자로 변환하기 위한 규칙을 만들었다.
  - ❹ 일반적으로 1개의 영문자를 1바이트, 즉 2의 8승( $2^8$ ) 정도의 공간에 저장한다.



## 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	@	080	P	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	L (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 Bouliane)

### 3. 문자열의 인덱싱

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



그림 6-3 문자열의 처리

```
>>> 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
```

## 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
TEAMLA 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
```

## 5. 문자열의 연산

- 기본적으로 문자열의 연산은 리스트 연산과 같음.

```
>>> a = "TEAM"
>>> b = "LAB"
>>> print(a + " " + b)
TEAM LAB
>>> print(a * 2 + " " + b * 2)
TEAMTEAM LABLAB
>>> if 'A' in a: print(a)
... else: print(b)
...
TEAM
```

# 덧셈으로 a와 b 변수 연결하기

# 곱하기로 반복 연산 가능

# 'A'가 a에 포함되었는지 확인

## 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)
```

## 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
capitalize()	첫 문자를 대문자로 변환
count('찾을 문자열')	'찾을 문자열'이 몇 개 들어있는지 개수 반환
find('찾을 문자열')	'찾을 문자열'이 왼쪽 끝부터 시작하여 몇 번째에 있는지 반환
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
isdigit()	문자열이 숫자인지 여부 반환
islower()	문자열이 소문자인지 여부 반환
isupper()	문자열이 대문자인지 여부 반환

## 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'
```

## 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
```



## 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."
```

## 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
\ 	다음 줄과 연속임을 표현	\b	백스페이스
\\	\\ 문자 자체	\n	줄바꾸기
\'	' 문자	\t	 키
\"	" 문자	\e	 키

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

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

## 6. 문자열 함수

두 줄 이상의 표현도 마찬가지로 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."""
```

**02**

**Lab: 단어 카운팅**

## 1. 실습 내용

- 팝그룹 비틀스의 라는 노래에서 'Yesterday'라는 단어가 몇 번 나 오는지 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()
```

## 2. 실행 결과

[실행결과]

Number of a Word 'Yesterday' 9

### 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)
```

- `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'
```



**03**

## **문자열 서식 지정**

# 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.

## 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"))
```

### [실행결과]

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

## 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
```

## 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.
```

## 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

## 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.
```

## 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.



## 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.
```

## 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
```

## 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      .'
```

### 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.
```

# Thank you!