06

문자열

목차

01. 문자열의 이해

문자열의 개념

- 문자열은 시퀀스 자료형(sequence data type)이다.

```
a = "abcde"
```

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

[시퀀스 자료형]

01. 문자열의 이해

문자열과 메모리 공간

- 일반적으로 문자열을 저장하기 위해서는 영문자 한 글자당 1바이트의 메모리 공간을 사용
- 다음과 같은 코드로 문자열이 저장된 공간의 크기를 확인할 수 있다.

```
>>> 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
```

- 컴퓨터에 a라고 알려 줘도 컴퓨터는 정확히 a라는 텍스트를 인식하는 것이 아니다. 대신 컴퓨터는 이 정보를 이진수로 변환하여 저장한다.

01. 문자열의 이해

문자열과 메모리 공간

- 컴퓨터 공학자들은 이러한 문자를 처리하기 위해 이진수로 변환되는 표준 규칙을 만들었다. ASCII, CP949, MS949, UTF-8 등 이러한 규칙을 인코딩(encoding)이라고 한다.

① 컴퓨터는 문자를 직접 인식하지 못한다.

② 컴퓨터는 문자를 숫자로 변환하여 인식한다.

③ 사람들은 문자를 숫자로 변환하기 위한 규칙을 만들었다.

④ 일반적으로 이 규칙은 1개의 영문자를 1바이트, 즉 2의 8승(28) 정도의 공간에 저장될 수 있도록 정하였다.

01. 문자열의 이해

■ 문자열과 메모리 공간

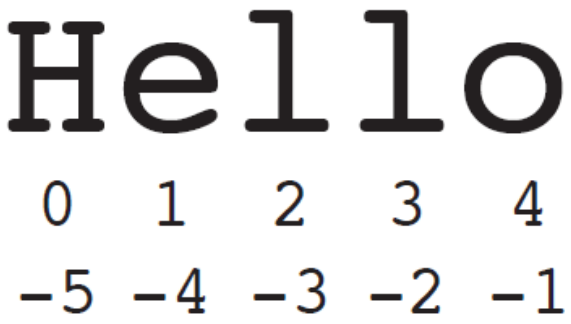
000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(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	∟	(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	⌂

[UTF-8의 유니코드(출처: Nicolas Bouliane)]

01. 문자열의 이해

문자열의 인덱싱

- 리스트처럼 글자 하나하나가 상대적인 주소(offset)를 가지는데, 이 주소를 사용해 할당된 값을 가져오는 인덱싱을 사용할 수 있다.



[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
```

01. 문자열의 이해

문자열의 슬라이싱

- **슬라이싱(slicing)** : 문자열의 주소값을 기반으로 문자열의 부분값을 반환하는 기법이다.

```
>>> a = "TEAMLAB MOOC, AWESOME Python"
>>> print(a[0:6], " AND ", a[-9:])          # a 변수의 0부터 5까지, -9부터 끝까지
TEAMLA  AND  ME Pyhon
>>> 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
```

01. 문자열의 이해

문자열의 연산

- 가장 기본적인 연산은 리스트의 연산과 같다. 예를 들어, 문자열 변수 'a'와 정수형인 2의 'a+2'와 같은 연산은 동작하지 않는다. 하지만 'a*2'와 같은 연산은 지원한다.

```
>>> 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
```

01. 문자열의 이해

문자열의 연산

- 다음과 같이 코드를 작성하면 문자열과 정수형의 연산으로 인식하여 덧셈 연산이 실행되지 않는다.

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

01. 문자열의 이해

문자열의 연산

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 들어 있는지 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

01. 문자열의 이해

문자열의 연산

함수명	기능
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

01. 문자열의 이해

■ 문자열의 연산

- **upper( ) 함수 :** 문자열을 대문자로 변환하는 함수
- **lower( ) 함수 :** 소문자로 변환하는 함수
- 참고로 문자열 함수를 사용하는 방법은 문자열 변수 다음에 '.문자열 함수'를 입력하면 된다.

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

01. 문자열의 이해

문자열의 연산

- **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'
```

01. 문자열의 이해

문자열의 연산

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

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

01. 문자열의 이해

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.
```

- 이러한 문제를 지원하기 위해 파이썬에서는 다양한 기능을 제공한다. 먼저 문자열 자체에 작은따옴표나 큰따옴표 가 들어가 있는 경우이다.

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

01. 문자열의 이해

여기서 잠깐! 문자열 표현과 특수문자

- 다음으로 파이썬의 특수문자 기능을 사용하는 것이다. 아래의 특수문자를 사용할 경우 다음과 같이 아포스트로피(')를 사용할 수 있다.

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

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

01. 문자열의 이해

여기서 잠깐! 문자열 표현과 특수문자

- 또 다른 문제로는 다음과 같은 줄 바꿈 표현이 있다. 이러한 경우에도 문자열로 표현하기 어렵다.

```
It's Ok.
I'm Happy.
See you.
```

- 두 줄 이상의 표현도 마찬가지이며, 두 가지로 표현할 수 있다. 하나는 큰따옴표(")나 작은따옴표(')를 3개로 연결하는 방법이다. 다음과 같이 선언한다

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

02. Lab: 단어 카운팅

실습 내용

- 앞에서 배운 문자열의 여러 기능을 사용하여 단어 카운팅 프로그램을 만들어 보자.
- 이번에 진행할 Lab은 팝 그룹 비틀스의 <Yesterday>라는 노래에서 'Yesterday'라는 단어가 몇 번 나오는지 맞히는 단어 카운팅 프로그램이다.

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

실행 결과

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

02. Lab: 단어 카운팅

문제 해결

코드 6-1 yesterday.py

```
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)
```

03. 문자열 서식 지정

서식 지정의 개념

- print( ) 함수를 사용하다 보면 어떤 형식에 맞추어 결과를 출력해야 할 일이 발생하기도 한다. 특히 엑셀을 사용할 때 통화 단위, 세 자리 숫자 단위 띄어쓰기, % 출력 등 다양한 형식에 맞추어 출력할 일이 생기는데, 이를 서식 지정(formatting)이라고 한다.

03. 문자열 서식 지정

% 서식과 format( ) 함수

- 문자열의 서식(format)을 설정할 때, print( ) 함수는 기본적인 출력 형식 외에 % 서식과 format( ) 함수를 구문으로 사용하여 출력 양식을 지정할 수 있다.

코드 6-2 formatting1.py

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

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

03. 문자열 서식 지정

% 서식과 format( ) 함수

- 이런 식으로 서식을 지정하여 출력하면 어떤 장점이 있을까?

① 데이터와 출력 형식을 분류할 수 있다. 같은 내용을 여러 번 출력하기 위해 기존 print( )문에 띄어쓰기를 넣어 + 기호로 문자열 형태를 붙여 주는 것보다 시각적으로 훨씬 이해하기 쉽게 코드를 표현할 수 있다.

② 데이터를 형식에 따라 다르게 표현할 수 있다. [코드 6-3]을 보면 문자열 형태인 ('one', 'two') 구문과 정수형인 (1, 2) 구문이 각각 %s와 %d로 다르게 할당되는 것을 확인할 수 있다. 서식 지정 기능은 각 변수의 자료형에 맞게 다른 서식으로 지정한다

코드 6-3 formatting2.py

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

```
one two
1 2
```

03. 문자열 서식 지정

% 서식과 format( ) 함수 : % 서식

- % 서식은 다음과 같은 형태로 출력 양식을 표현하는 기법이다.

```
'%자료형 % (값)'
```

- % 서식을 사용한 가장 간단한 표현 형식은 [코드 6-4]와 같다.

코드 6-4 formatting3.py

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

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

03. 문자열 서식 지정

% 서식과 format( ) 함수 : % 서식

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

[변수의 자료형에 따른 서식]

03. 문자열 서식 지정

% 서식과 format( ) 함수 : % 서식

- %는 1개 이상의 값도 할당할 수 있다. 다음 코드처럼 % 뒤에 괄호를 쓰고, 그 안에 순서대로 값을 입력하면 된다.

```
>>> 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 실행된다.

코드 6-5 formatting4.py

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

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

03. 문자열 서식 지정

% 서식과 format( ) 함수 : format( ) 함수

- **format( ) 함수** : % 서식과 거의 같지만, 문자열 형태가 있는 함수를 사용한다는 차이점이 있다. 문자열 서식은 함수이므로 다음과 같은 형태로 서식을 지정할 수 있다.

```
"{자료형}".format(인수)
```

- 다음 코드는 format( ) 함수를 사용한 가장 기본적인 표현 형태로, 숫자 20이 {0}에 할당되어 출력된다. 기존 % 서식과 비교하면, 자료형을 바로 지정해 주지 않고 순서대로 변수가 할당된다는 장점이 있다.

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

03. 문자열 서식 지정

% 서식과 format( ) 함수 : format( ) 함수

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

코드 6-6 formatting5.py

```
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))
```

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

➡ 4행의 Price per unit: {1:.2f}는 기존 format( ) 함수의 쓰임과 다르게 .2f라는 구문이 추가되었다. 2는 소수점 둘째 자리까지 출력하라는 뜻이다.

03. 문자열 서식 지정

패딩

- 파이썬의 서식 지정 기능에는 여유 공간을 지정하여 글자 배열을 맞추고 소수점 자릿수를 맞추는 패딩(padding)기능이 있다. % 서식과 format( ) 함수 모두 패딩 기능을 제공한다.

03. 문자열 서식 지정

■ 패딩 : % 서식의 패딩

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

➡ 첫 번째 줄의 print("%10d" % 12)는 10자리의 공간을 확보하고, 우측 정렬로 12를 출력하라는 명령이다. 기본 정렬이 우측 정렬이므로 좌측에서 아홉 번째 칸부터 12가 출력된다. 좌측 정렬을 하기 위해서는 세 번째 줄처럼 - 부호를 붙이면 된다.

03. 문자열 서식 지정

■ 패딩 : % 서식의 패딩

```
>>> 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
```

➡ 실수에서도 자릿수와 소수점 자릿수를 지정할 수 있다.

첫 번째 줄의 print("%10.3f" % 5.94343)은 10자리의 공간을 확보하고 소수점 셋째 자리까지 출력하라는 뜻이다. 이때 10자리 안에는 소수점이 포함된다. 역시 우측 정렬 기준이며, 좌측 정렬을 하기 위해서는 - 부호를 붙이면 된다.

03. 문자열 서식 지정

■ 패딩 : format( ) 함수의 패딩

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

➡ 첫 번째 줄의 print("{0:>10s}".format("Apple"))은 10자리의 공간을 확보하고, 우측 정렬로 문자열 'Apple'을 출력하라는 명령이다. 좌측 정렬을 하기 위해서는 '{0:<10s}'처럼 < 부호를 사용하면 된다.

03. 문자열 서식 지정

■ 패딩 : format( ) 함수의 패딩

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

➡ 실수에서도 자릿수와 소수점 자릿수를 지정할 수 있다.

첫 번째 줄의 "{1:>10.5f}.".format("Apple", 5.243)을 입력하면, 10자리의 공간을 확보하고, 소수점 다섯 번째 자리까지 실수를 출력한다. 이때 10자리 안에는 소수점이 포함된다. 역시 우측 정렬 기준이며, 좌측 정렬을 위해서는 < 부호를 사용한다.

03. 문자열 서식 지정

여기서 잠깐! 네이밍(naming)

- 서식 지정을 활용하여 print( ) 함수를 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 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.
```