

문자열 자료형 (string)

- 문자열(string)은 연속된 문자로 구성된 순서가 있는 자료형
 - 문자열은 일단 생성하면 구성된 데이터 문자들을 바꿀 수 없음 (immutable)
 - indexing / slicing 기법을 사용하여 문자열의 데이터를 참조할 수 있음
- 큰따옴표(""), 작은 따옴표('), 큰따옴표 3개 연속('''), 작은따옴표 3개 연속(''') 중, 어느 것으로 시작해서 양쪽 둘러싸기로 문자열을 정의함
- 파이썬에서는 문자 하나를 다루는 문자형 자료형이 별도로 없음

```
>>> "Hello"
'Hello'
>>> msg = 'Hello'
>>> msg
'Hello'
>>> print(msg)
Hello
>>> 'The string'
'The string'
>>> '''The string'''
'The string'
```

큰따옴표("")로 시작했다가
작은따옴표(')로 끝내면 문법적인
오류임. 동일한 문자로 끝나야 함.

문자열 자료형 (string)

- 컴퓨터는 문자열(string)을 처리하기 위해 규칙에 따라 각 문자를 이진수로 변환하며, 이를 인코딩이라 함
- 파이썬에서는 문자열을 유니코드(Unicode)내의 ASCII 문자 코드로 인코딩함
- 한글 문자들도 유니코드로 인코딩함

DEC	HEX	OCT	Char	DEC	HEX	OCT	Char	DEC	HEX	OCT	Char
0	00	000	Ctrl-@ NUL	43	2B	053	,	86	56	126	V
1	01	001	Ctrl-A SOH	44	2C	054	.	87	57	127	W
2	02	002	Ctrl-B STX	45	2D	055	-	88	58	130	X
3	03	003	Ctrl-C ETX	46	2E	056	_	89	59	131	Y
4	04	004	Ctrl-D EOT	47	2F	057	/	90	5A	132	Z
5	05	005	Ctrl-E ENQ	48	30	060	0	91	5B	133	[
6	06	006	Ctrl-F ACK	49	31	061	1	92	5C	134	\
7	07	007	Ctrl-G BEL	50	32	062	2	93	5D	135]
8	08	010	Ctrl-H BS	51	33	063	3	94	5E	136	^
9	09	011	Ctrl-I HT	52	34	064	4	95	5F	137	_
10	0A	012	Ctrl-J LF	53	35	065	5	96	60	140	`
11	0B	013	Ctrl-K VT	54	36	066	6	97	61	141	a
12	0C	014	Ctrl-L FF	55	37	067	7	98	62	142	b
13	0D	015	Ctrl-M CR	56	38	070	8	99	63	143	c
14	0E	016	Ctrl-N SO	57	39	071	9	100	64	144	d
15	0F	017	Ctrl-O SI	58	3A	072	:	101	65	145	e
16	10	020	Ctrl-P DLE	59	3B	073	;	102	66	146	f
17	11	021	Ctrl-Q DC1	60	3C	074	<	103	67	147	g
18	12	022	Ctrl-R DC2	61	3D	075	=	104	68	150	h
19	13	023	Ctrl-S DC3	62	3E	076	>	105	69	151	i
20	14	024	Ctrl-T DC4	63	3F	077	?	106	6A	152	j
21	15	025	Ctrl-U NAK	64	40	100	@	107	6B	153	k
22	16	026	Ctrl-V SYN	65	41	101	A	108	6C	154	l
23	17	027	Ctrl-W ETB	66	42	102	B	109	6D	155	m
24	18	030	Ctrl-X CAN	67	43	103	C	110	6E	156	n
25	19	031	Ctrl-Y EM	68	44	104	D	111	6F	157	o
26	1A	032	Ctrl-Z SUB	69	45	105	E	112	70	160	p
27	1B	033	Ctrl-[ESC	70	46	106	F	113	71	161	q
28	1C	034	Ctrl-\ FS	71	47	107	G	114	72	162	r
29	1D	035	Ctrl-] GS	72	48	110	H	115	73	163	s
30	1E	036	Ctrl-^ RS	73	49	111	I	116	74	164	t
31	1F	037	Ctrl_ US	74	4A	112	J	117	75	165	u
32	20	040	Space	75	4B	113	K	118	76	166	v
33	21	041	!	76	4C	114	L	119	77	167	w
34	22	042	"	77	4D	115	M	120	78	170	x
35	23	043	#	78	4E	116	N	121	79	171	y
36	24	044	\$	79	4F	117	O	122	7A	172	z
37	25	045	%	80	50	120	P	123	7B	173	{
38	26	046	&	81	51	121	Q	124	7C	174	
39	27	047	'	82	52	122	R	125	7D	175	}
40	28	050	(83	53	123	S	126	7E	176	~
41	29	051)	84	54	124	T	127	7F	177	DEL
42	2A	052	*	85	55	125	U				

문자열 자료형 (string)

- 문자열 데이터에 작은따옴표 또는 큰따옴표가 있는 경우

```
>>> "the law's requirements"
```

```
"the law's requirements"
```

```
>>> 'the law's requirements'
```

```
"the law's requirements"
```

```
>>> "'Hi! Good morning?'"
```

```
"'Hi! Good morning?'"
```

```
>>> "'Hi! Good morning?'"
```

```
"'Hi! Good morning?'"
```

```
>>>
```

→ 문자열에 작은따옴표(')를 포함하기
위해서 문자열을 큰따옴표(")로
둘러쌘

→ 문자열에 큰따옴표(")를 포함시키기
위해서 문자열을 작은따옴표(')로
둘러쌘

백슬래시(\)를 작은따옴표나 큰따옴표 앞에
삽입하여 문자열의 표시와 구분

문자열 자료형 (string)

- 문자열의 데이터가 여러 줄인 경우
 - 문자열 데이터에 줄바꿈 문자 **\n** 을 삽입

```
>>> string = "This is first line.\n Second line.....\nFinal line."  
>>> string  
'This is first line.\n Second line.....\nFinal line.'  
>>> print(string)  
This is first line.  
Second line.....  
Final line.
```

- 연속된 작은따옴표 3개 또는 큰따옴표 3개 사용

```
>>> string = """This is first line. -----> 줄바꿈 문자 삽입 필요 없이  
Second line.....  
Final line. """  
엔터키 입력 후 데이터를 입력  
하면 됨  
>>> string  
'This is first line.\n Second line.....\nFinal line.'  
>>> print(string)  
This is first line.  
Second line.....  
Final line.
```

문자열 자료형 (string)

- 줄바꿈이 아닌, 긴 문자열을 여러줄에 걸쳐 입력할 경우는 **\(문자 \w와 같음)** 기호를 사용하여 작성할 수 있음

```
>>> string = "This is long sentence.\w  
without newline character....\w  
single line sentence"  
>>> string  
'This is long sentence.without newline character....single line sentence'  
>>> print(string)  
This is long sentence.without newline character....single line sentence
```

특수 문자 (Escape Sequence)

- 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"
 - 주로 출력물을 보기 좋게 정렬하는 용도로 이용
 - backslash(\)와 단일 문자로 구성된 조합으로 코딩시 표현할 수 없는 문자 또는 control들을 표시

notation	meaning	notation	meaning
\\	backslash (\)	\n	Newline(줄바꿈)
\'	single quote(')	\t	horizontal tab
\"	double quote(")		

문자열 연산

- + 연산자 : 두 문자열 연결하기

```
>>> 'Hello' + 'World!'
'HelloWorld!'
>>> message = "파이썬에" + " 오신 것을" + " 환영합니다"
>>> message
'파이썬에 오신 것을 환영합니다'
```

- * 연산자 : str * n인 경우 문자열 str을 n번 반복

```
>>> print("=" * 10)
=====
```

```
>>> a="=" * 10; print(a)
=====
```

- in 연산자
 - 지정한 부분이 문자열 안에 존재하는지를 확인
 - 있으면 True를 반환, 없으면 False를 반환
- not in 연산자 (in 연산자 반대 개념)

```
>>> "Sog" in "Sogang University"
True
```

문자열 인덱싱(Indexing)

- 인덱스는 문자열의 각 문자마다 번호를 매기는 것

`s1 = "Hi there!"`
`len(s1) : 9`

0	1	2	3	4	5	6	7	8
H	i		t	h	e	r	e	!
-9	-8	-7	-6	-5	-4	-3	-2	-1

`s2 = "안녕하세요?"`
`len(s2) : 6`

0	1	2	3	4	5
안	녕	하	세	요	?
-6	-5	-4	-3	-2	-1

- 인덱스 범위
 - 양수 인덱스 : 0 ~ (len(문자열)-1)
 - 음수 인덱스 : -len(문자열) ~ -1 **교과내용에서 다루지 않음**
 - len(문자열)** 함수 : 입력 받은 문자열의 길이(문자 개수)를 반환
- 인덱스 표기 방식은 대괄호([]) 안에 인덱스 번호를 기입하여 해당 문자열에서 지정한 문자를 참조하는 것

```
>>> s1[3]
't'
```

```
>>> c = s2[5]; c
'?'
```

```
>>> s1[len(s1)-1]
'!'
```

```
s1[len(s1)]
```

Traceback (most recent call last):
File "<pyshell#1>", line 1, in <module>
s1[len(s1)]
IndexError: string index out of range

문자열 슬라이싱(slicing)

- 문자열의 인덱스를 기반으로 문자열의 부분 문자열을 반환(원본 문자열은 그대로 유지)하는 방법
- 문자열에서 일부 문자들 추출에 사용
- 인덱싱과 슬라이싱 기법은 다른 sequence 데이터 형(list, tuple)에서도 사용 가능
- 문자열 slicing 표기법
 - **string[start : end : step]**
 - start : 시작 인덱스 값. default 값은 0
 - end : (end-1) 인덱스까지 추출, default 값은 len(s)
 - step : 추출 간격, default 값은 1

```
s1 = "My number is 010-8888-1234"
print(s1[13::])    # 010-8888-1234
print(s1[3:9:2])   # nme
print(s1[::-1])    # 4321-8888-010 si rebmun yM (문자열을 거꾸로)
```

문자열 슬라이싱(slicing)

- 문자열의 일부를 추출하여 새로운 변수에 저장

: 문자열을 슬라이싱해서 새로운 변수 a에 저장해도 원본 문자열은 변하지 않음

```
>>> s = "Good Morning"
>>> a = s[0:4]    #s[0:4:1]과 동일
>>> b = s[5:12]   #s[5::], s[5::1]과 동일
>>> a
'Good'
>>> b
'Morning'
>>> s
'Good Morning'
>>>
```

문자열 slicing

- 문자열의 데이터는 변경할 수 없음
 - 문자열 객체는 변경 불가능. (immutable한 자료형)
 - 문자열 객체가 일단 생성되면, 그 객체의 내용은 변경될 수 없음
- "Pithon"이라는 문자열을 "Python"으로 변경하고자 할 때

```
>>> a = "Pithon"
```

```
>>> a[1]
```

```
'i'
```

```
>>> a[1] = 'y'
```

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

a[0]=1

TypeError: 'str' object does not support item assignment

인덱스를 사용하여 a[1]의 값 i를 y로 바꾸려고 하면 에러 발생

문자열 slicing

- 문자열의 데이터를 바꾸려면 슬라이싱을 이용
: 슬라이싱을 이용하여 새로운 문자열을 생성하여 변경
- 데이터의 일부분을 변경하고자 할 때

```
>>> a = "Python is bad...for me."  
>>> a[:10]  
'Python is '  
>>> a[13:]  
'...for me.'  
>>> a = a[:10] + "good" + a[13:]  
>>> a  
'Python is good...for me.'
```

문자열 포매팅(Formatting) - % 사용

- 문자열과 다른 유형의 데이터를 혼합하여 원하는 문자열을 만들 때 사용
 - 여러 데이터(숫자, 문자열, 변수)를 원하는 위치에 삽입 가능함
 - 문자열에 값을 삽입하고 싶은 위치에 % 기호와 출력 형식 지정 문자를 첨가
 - 문자열 뒤에 % 기호와 삽입할 값을 지정

```
>>> num = 10
>>> "I eat %d apples." % 3
'I eat 3 apples.'
>>> new = "I eat %d apples." % num
>>> new
'I eat 10 apples.'
>>> "%s eats %d apples." %("Tom", num)
'Tom eats 10 apples.'
>>>
```

문자열 포매팅(Formatting) - % 사용

- 문자열 포맷 코드(출력 형식 지정 문자)

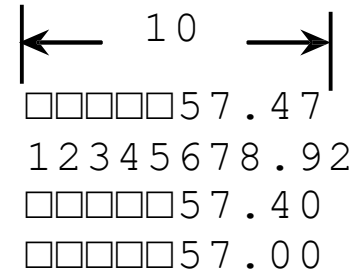
코드	설명
%s	문자열 (String). 어떤 데이터 형의 값이든 문자열로 변환
%c	문자 1개(character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8진수
%x	16진수
%%	문자 % 자체

```
>>> "rate is %s" % 3.234    # "rate is %s" % "3.234"
'rate is 3.234'
>>> "rate is %d" % 3.234
'rate is 3'
>>> "rate is %f" % 3.234
'rate is 3.234000'
>>> "Error is %d%%." % 98
'Error is 98%.'
```

문자열 포매팅(Formatting) - % 사용

- 특정 서식에 맞춰 숫자를 출력하는 것이 필요할 때 사용
- 실수 서식 지정

```
print("average = %10.2f " % 57.467657) print("average = %10.2f" % 12345678.923)
print("average = %10.2f" % 57.4) print("average = %10.2f" % 57)
```



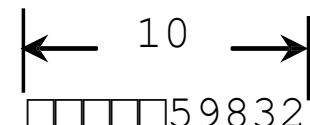
전체 10 자리 중, 소수점 이하 2 자리

```
a = 1/3
print("1/3 =", a, "(too many digits).")
print("a = %.3f" % a)
```

출력 1/3 = 0.3333333333333333 (too many digits).
a = 0.333

- 정수 서식 지정

```
print("average = %10d" % 59832)
```



문자열 포매팅 – 문자열 메소드 str.format() 사용

- str.format() : 문자열 포매팅의 %와 유사
 - 문자열 객체의 메소드 format()을 사용
 - 문자열에 값을 삽입하고 싶은 위치에 중괄호 {} 지정
 - format() 메소드의 인수로 삽입될 데이터를 지정

```
>>> num = 10
>>> "I eat {} apples.".format(3)
'I eat 3 apples.'
>>> "I eat {} apples.".format(num)
'I eat 10 apples.'
>>> "{} eats {} apples.".format("Tom", num)
'Tom eats 10 apples.'
>>>
```

8.3f	소숫점이하 3자리 실수(8자리)	,	천 단위마다 콤마 삽입
8.3e	소숫점이하 3자리 지수(8자리)	10,d	10자리 정수(천 단위마다 콤마)
5d	5자리 정수	08d	8자리 정수(남는 자리는 0을 붙임)
10	10 글자로 표시	+.2f	소수점이하 2자리 실수(항상 부호)
*^30	30글자 가운데(좌우는 *로 채움)	.2%	소수점이하 2자리 백분율

문자열 포매팅 – str.format() 사용

Bar(|) mark는 단순히 실제 출력된
글자 수를 보이기 위한 구분용임

```
print("{2}, {0}, {1}".format(10,20,30))           # 30, 10, 20
print("{:8.3f}|".format(3.1415926))              # |  3.142|
                                                    # └── 8자리
print("{:+.2f}".format(3.1415926))                # +3.14
print("|{:8.3e}|".format(314.15926))              # |3.142e+02|
                                                    # └── over되면 그대로 출력
print("|{:10d}|".format(12345))                   # |  12345|
                                                    # |00,012,345|
print("|{:010,d}|".format(12345))
print("{:.2%}".format(1/3))                       # 33.33%
                                                    # 소수점이하 2자리
                                                    # 백분율
x = [10,20,30]; y = "abc"
print("{0[1]},{1[2]}".format(x,y))               # 20,c
                                                    # x[1] y[2]
print("{:*^12}".format("center"))                # ***center***
                                                    # 12글자 center(좌우는 *로 채움)
```

문자열 포매팅 – str.format() 사용

```
number = 3
```

```
print("I eat {:10} apples.".format(number))    # I eat          3 apples.
```

```
print("I eat {:<10} apples.".format(number))    # I eat 3        apples.
```

← 왼쪽 정렬

```
print("The light was {:10}.".format('good'))    # The light was good  .
```

문자열은 디폴트로 왼쪽 정렬

문자열 포매팅 : f-string

- python version 3.6 부터 기능 지원
- f 접두사 붙여서 지정

```
a = 10; b = "score"
print(f"{b} is {a}")           # score is 10

x = 10; y = 20
print(f"x*y = {x*y}")         # x*y = 200

t = ("computer", "Mid", 90)
print("%s" %str(t))           #tuple 출력시 str로 변환 과정이 필요
print(f"{t}")                 # ('computer', 'Mid', 90)

x = [10,20,30]; y = "abc"
print("{0[1]},{1[2]}".format(x,y))
print(f"{x[1]},{y[2]}")       # 20,c

print("|{:8.3f}|".format(3.1415926))
print(f"|{3.1415926:8.3f}|")   # | 3.142|
```

문자열 메소드

- `str.split()` / `str.split('구분자')`
 - `split()` 메소드의 인수로 받은 구분자를 기준으로 문자열을 분리하여 분리된 문자열들을 원소로하는 리스트를 반환
 - `split()` 메소드의 인수가 없는 경우, default 구분자는 공백(space) 문자
 - 분리된 각각은 또 다른 문자열

```
>>> a = "Life is too short"
>>> L = a.split(); print(L)
['Life', 'is', 'too', 'short']
>>> print(type(L))
<class 'list'>
>>> StartDay = "2020/09/01"
>>> year, month, day = StartDay.split("/")
>>> year
'2020'
>>> month
'09'
>>> day
'01'
```

문자열 메소드

메소드	설명
lower()	문자열 데이터를 모두 소문자로 바꾼 문자열 을 반환
upper()	문자열 데이터를 모두 대문자로 바꾼 문자열 을 반환
islower()	문자열 내의 모든 문자들이 소문자이면, True 를 반환
isupper()	문자열 내의 모든 문자들이 대문자이면, True 를 반환
isalpha()	문자열이 알파벳(영문, 한글등)으로만 구성되어 있으면 True 를 반환
isnumeric() isdigit()	문자열이 숫자로만 구성되어 있으면 True 반환
isalnum()	문자열이 알파벳과 숫자로만 구성되어 있으면 True 반환
swapcase()	문자열 데이터의 대소문자를 상호 변환한 문자열 반환
title()	각 단어의 제일 앞 글자만 대문자로 변환한 문자열 반환

문자열 메소드

메소드	설명
replace()	문자열 내에서 지정한 문자열을 새로운 문자열로 바꾼 전체 문자열을 반환
startswith()	문자열이 매개변수로 입력한 문자열로 시작하면 True 반환
endswith()	문자열이 매개변수로 입력한 문자열로 끝나면 True 반환
find()	문자열 내에서 매개변수로 입력한 문자열이 시작하는 인덱스를 반환 (존재하지 않으면 -1을 반환). 여러 번 존재하면 첫번째 발견하는 인덱스 반환
rfind()	문자열 내에서 매개변수로 입력한 문자열을 뒤에서부터 찾아서 인덱스 반환 (존재하지 않으면 -1을 반환)
count()	문자열 내에서 매개변수로 입력한 문자열이 몇 번 있는지 그 개수를 반환
strip()	문자열 양쪽에 있는 공백을 제거한 문자열 반환 (lstrip() / rstrip(): 왼쪽/오른쪽에서 공백제거)
center(width)	주어진 폭의 가운데 중심으로 정렬된 문자열을 반환 (ljust(width) / rjust(width): 왼쪽 / 오른쪽 중심으로 정렬)

문자열 메소드 example

```
s1 = "Hi!"; s2 = "hi!"; s3 = "HI7"; s4 = " "; s5 = "123"
print(s1.isalpha())      # False 영문자로만 구성?
print(s2.islower())      # True 소문자로만 구성?
print(s3.isalnum())      # True 영문자와 숫자만으로 구성?
print(s4.isspace())      # True 빈칸으로만 구성?
print(s5.isnumeric())    # True 숫자(0~9)로만 구성?
```

```
s6 = "aba"; s = "123abababa"
print(s.count(s6))       # 2 s에서 s6의 반복 횟수(중복 없는)
print(s.find(s6))        # 3 s에서 s1이 있는 1st index
```

```
print(s1.upper())        # HI! 모두 대문자로 바꿔 반환,s1은 바뀌지 않음
print(s3.lower())        # hi7 모두 소문자로 바꿔 반환,s3는 바뀌지 않음
```

```
s = "Hi everybody!"
s1 = s.replace("Hi", "Hello") # Hi를 Hello로 바꿔 반환 ,s는 바뀌지 않음
s2 = s.replace("!", "")      # !를 제거하고 반환,s는 바뀌지 않음
print(s1)                   # Hello everybody!
print(s2)                   # Hi everybody
```

문자열 메소드 example

```
s = "sogang university"
s1 = s.title()    # 각 단어의 앞 글자 대문자로 변환
print(s1)        # Sogang University
```

```
s = "sogang university"
print(s.startswith("sog")) # True
print(s.endswith("ts"))   # False
```

```
s = "  sogang university  "
print(s.lstrip())  # sogang university ____
print(s.rstrip())  # _sogang university
print(s.strip())   # sogang university
```


문자열 메소드 example

- `str.join(iter)` : 문자열로 구성된 `iter`(리스트, 튜플, 집합)의 원소들 사이에 `str`을 삽입해서 새로 구성한 문자열 반환

```
L = ["a", "b", "c"]
sep = ','
s = sep.join(L)    # (원소간 콤마로 묶음)
print(s)           # a, b, c

print(' '.join(L)) # a b c (원소간 빈칸으로 묶음)
print(''.join(L))  # abc   (원소간 빈칸 없이 묶음)
print('**'.join(L)) # a**b**c (원소간 **로 묶음)
print("_".join(("student", "name"))) # student_name (원소간 _로 묶음)

L1 = ["a", 10, "c"]
print(' '.join(L1)) # 에러 발생
```

```
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    print(' '.join(L1))
TypeError: sequence item 1: expected str instance,
int found
```

- **%nd** : **n** 자리 정수로 출력

```
a = -123
print("a = %d" %a)      # 자릿수 지정 안 함
print("a = %4d" %a)     # 정확한 자릿수 지정(부호 포함)
print("a = %1d" %a)     # 자릿수 모자라도 문제 없음
print("a = %5d" %a)     # 자릿수 남으면 빈칸으로 채움(좌측)
```

출력

```
a = -123
a = -123
a = -123
a = -123
```



- **%n.mf** : 전체 **n**자리에서 소수점 이하 **m**자리로 출력

```
a = -12345.1234567890123456789
```

```
print("a = %f" %a)
```

자릿수 지정 안 함

```
Print("a = %.19f" %a)
```

소수점 이하 19자리(오차 보임)

```
Print("a = %.3f" %a)
```

소수점 이하 3자리

```
print("a = %10.3f" %a)
```

정확한 자릿수 지정

```
print("a = %5.3f" %a)
```

자릿수 모자라도 문제 없음

```
print("a = %12.3f" %a)
```

남는 자릿수 빈칸으로 채움(좌측)

출력

```
a = -12345.123457
```

```
a = -12345.1234567890114703914
```

```
a = -12345.123
```

```
a = -12345.123
```

```
a = -12345.123
```

```
a = -12345.123
```

- **%n.me** : 전체 **n**자리에서 소수점 이하 **m**자리 지수형으로 출력

```
a = -12345.1234567890123456789
print("a = %e" %a)           # 자릿수 지정 안 함
print("a = %.22e" %a)        # 소수점 이하 22자리(오차 보임)
print("a = %.3e" %a)         # 소수점 이하 3자리
print("a = %10.3e" %a)       # 정확한 자릿수 지정
print("a = %5.3e" %a)        # 자릿수 모자라도 문제 없음
print("a = %12.3e" %a)       # 남는 자릿수 빈칸으로 채움(좌측)
```

출력

```
a = -1.234512e+04
a = -1.2345123456789011470391e+04
a = -1.235e+04
a = -1.235e+04
a = -1.235e+04
a = -1.235e+04
```