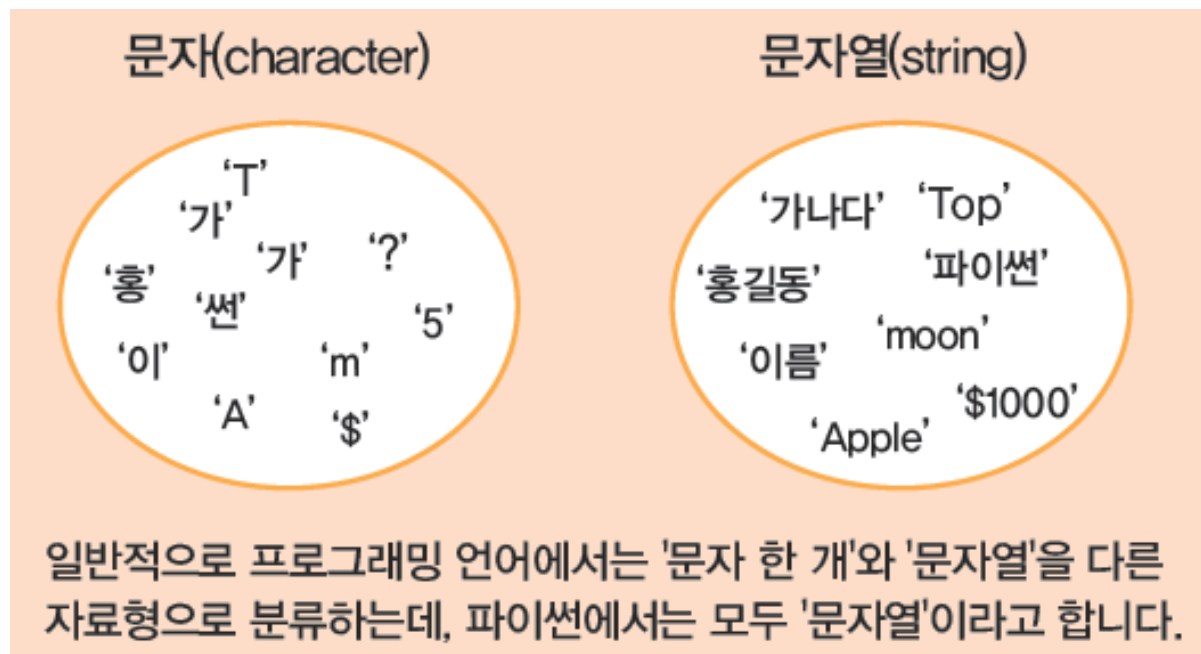


## 4. 문자열 자료형

1. 문자열 객체와 인덱스
2. 문자열 만들기
3. 문자열은 immutable 객체입니다
4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기
5. 문자열에 +, \*, in, not in, del 연산자 사용하기
6. 아스키 코드와 ord()/chr() 함수
7. 문자열에 함수 적용하기 - len(), max(), min(), sum(), sorted(), reversed()
8. 문자열 메소드
9. 정리

# 1. 문자열

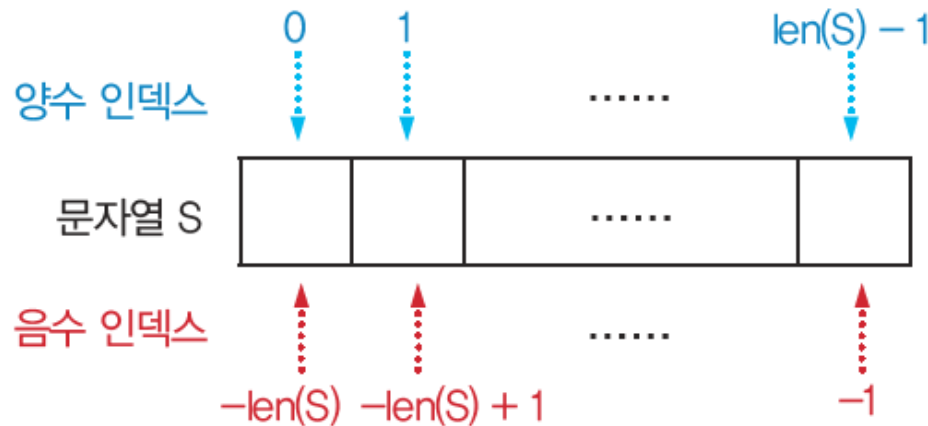
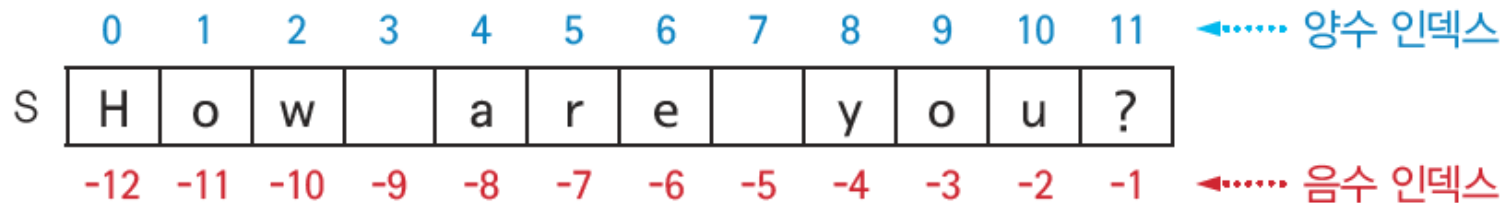
- ◆ 문자 자료형은 일반적으로 한 개의 문자를 의미함.
- ◆ 문자열 자료형은 문자가 1개 이상 연이어 있는 것을 의미함. (즉, 문자 1개도 문자열이라고 할 수 있음)
- ◆ 파이썬에서는 '문자열' 자료형만 있음.



# 1. 문자열 객체와 인덱스

## ◆ 문자열 만들기

S = 'How are you?'



$\text{len}(S)$  자리에는 문자열 S의 길이가 대체됩니다( $\text{len}(s)$ 는 12입니다).

# 1. 문자열 객체와 인덱스

## ◆ 문자열 일부분에 접근하기

- 문자열에서 문자 한 개 또는 여러 개를 사용하려면 대괄호 [] 기호를 사용함.

```
>>> S = 'How are you?'
```

```
>>> S [0]      # S [-12], greeting[-len(greeting)]도 같은 곳을 가리켜요.
```

```
'H'
```

```
>>> S [5]      # S [-7] 이라고 해도 되겠죠.
```

```
'r'
```

```
>>> S [5.0]    # 인덱스는 반드시 정수로 써야 합니다.
```

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

S [5.0]

TypeError: string indices must be integers

## 2. 문자열 만들기

### ◆ 문자열 만드는 4가지 방법

>>> s1 = 'hello world'      # 작은따옴표 한 개

>>> s2 = "hello world"      # 큰따옴표 한 개

>>> s3 = '''hello world'''    # 작은따옴표 3개

>>> s4 = """hello world"""    # 큰따옴표 3개

0	1	2	3	4	5	6	7	8	9	10
h	e	l	l	o		w	o	r	l	d

## 2. 문자열 만들기

### ◆ 빈 문자열 만들기

① 빈 따옴표 이용	② str() 함수 이용
<pre> &gt;&gt;&gt; s = ''      # 빈 문자열 &gt;&gt;&gt; print(s)    # 아무 것도 안 나옴 &gt;&gt;&gt; len(s)      # 빈 문자열 길이는 0 &gt;&gt;&gt; type(s) &lt;class 'str'&gt; </pre>	<pre> &gt;&gt;&gt; w = str( ) &gt;&gt;&gt; print(w) &gt;&gt;&gt; len(w) 0 &gt;&gt;&gt; type(w) &lt;class 'str'&gt; </pre>

```

>>> t = ' '      # 스페이스가 있습니다.
>>> print(t)
>>> len(t)      # 스페이스 한 개 있습니다.
1
>>> type(t)
<class 'str'>

```

스페이스도 한 개의 문자임.

## 2. 문자열 만들기

### ◆ 따옴표 사용하기

① 출력된 문자열에 작은 따옴표가 나오게 하고 싶은 경우

- 오른쪽과 같은 결과가 나오게 하고 싶다.

```
>>> print(s)  
one 'two' three
```

- 다음 세 경우 중에 하나를 사용한다.

```
s = "one 'two' three"
```

```
s = '''one 'two' three'''
```

```
s = """one 'two' three"""
```

## 2. 문자열 만들기

### ◆ 따옴표 사용하기

- ② 출력된 문자열에 큰따옴표가 나오게 하고 싶은 경우

```
>>> s = 'one "two" three'
>>> print(s)
one "two" three
```

```
>>> s = '''one "two" three'''
>>> print(s)
one "two" three
```

```
>>> s = """"one "two" three""""
>>> print(s)
one "two" three
```

- ③ 출력된 문자열에 작은따옴표와 큰따옴표가 모두 나오게 하고 싶은 경우

```
>>> s = '''one 'two' and 'three' four'''
>>> print(s)
one 'two' and 'three' four
```

```
>>> s = """"one 'two' and "three" four""""
>>> print(s)
one 'two' and "three" four
```



## 2. 문자열 만들기

### ◆ 따옴표 사용하기

- ④ 여러 줄에 걸친 긴 문장을 하나의 줄로 인식하고 싶을 때
- 다음과 같이 한 줄을 끝내지 않고 엔터를 치면 에러가 발생함.

```
>>> print('Python is widely used high-level programming language
```

```
SyntaxError: EOL while scanning string literal
```

- 역슬래쉬 ('\\') 기호를 이용하면 다음 줄과 연결됨.

```
>>> print('Python is widely used high-level programming language \\  
for general-purpose programming.')  
Python is widely used high-level programming language for general-purpose  
programming.
```

## 2. 문자열 만들기

### ◆ 따옴표 사용하기

- ④ 여러 줄에 걸친 긴 문장을 하나의 줄로 인식하고 싶을 때
  - 작은 따옴표 또는 큰 따옴표 세 개를 사용할 수 있음

```
>>> print("""Python is widely used high-level programming language  
for general-purpose programming.""")  
Python is widely used high-level programming language  
for general-purpose programming.
```

## 2. 문자열 만들기

### ◆ 따옴표 사용하기

- ⑤ 큰따옴표 세 개를 사용하는 경우
  - 작은 따옴표 세 개와 똑같이 사용할 수 있음.
  - 큰 따옴표 세 개를 주석으로 사용할 때 ‘docstring’이라고 부름.  
(나중에 함수, 모듈, 클래스에서 많이 사용함)

## 2. 문자열 만들기

### ◆ 이스케이프 시퀀스 (escape sequence)

- ‘\’ 기호와 문자 한 개가 합해져서 특별한 의미를 갖는 문자
- 많이 사용하는 이스케이프 시퀀스

이스케이프 시퀀스	의미	예제
\	작은따옴표를 그대로 사용함	>>> print('Let\'s learn python.') Let's learn python.
\"	큰따옴표를 그대로 사용함	>>> print("Say \"hello\" to your mom.") Say "hello" to your mom.
\n	줄바꿈 (엔터 효과)	>>> print("hello \nworld") hello world
\t	문자 사이에 탭 간격을 줌	>>> print("hello \tworld") hello      world
\\	문자 \를 그대로 사용함	>>> print("hello \\ world") hello \ world

## 2. 문자열 만들기

### ◆ 수치 자료형을 문자열로 변환하기

- 정수나 실수로 된 자료형을 문자열로 변환해야 하는 경우, `str()` 함수를 사용함.

문자열 ← 정수	<pre>&gt;&gt;&gt; str(10), str(5), str(1234) ('10', '5', '1234')</pre>
문자열 ← 실수	<pre>&gt;&gt;&gt; str(3.14), str(5.0), str(123.123) ('3.14', '5.0', '123.123')</pre>

## 2. 문자열 만들기

### ◆ 참고

- 컴퓨터는 어떤 문자이든지 따옴표가 붙어 있으면 문자열로 인식함.
- 아래 변수 a는 숫자 5를 문자열로 저장했기 때문에 a의 자료형은 정수가 아니라 문자열임.
- IDLE에서 출력하기

```
>>> a = '5'      # 변수 a의 자료형은 '문자열'이에요.
```

```
>>> print(a)     # print() 함수로 출력했더니 숫자 5만 출력됩니다.
```

```
5
```

```
>>> a            # 변수명만 넣었더니 따옴표가 붙어 나옵니다.
```

```
'5'
```

### 3. 문자열은 immutable 객체입니다

- ◆ 문자열은 한 번 만들면 바꿀 수 없음 (immutable)

```
>>> s = 'hello world'
```

```
>>> s[0] = 'H'          # Hello world라고 수정하려는데 에러가 발생합니다.
```

```
Traceback (most recent call last):
```

```
File "<pyshell#129>", line 1, in <module>
```

```
    s[0] = 'H'
```

```
TypeError: 'str' object does not support item assignment
```

```
>>> t = 'friend'
```

```
>>> t[-2] = 'n'         # friend라고 수정하려는데 에러가 발생합니다.
```

```
Traceback (most recent call last):
```

```
File "<pyshell#133>", line 1, in <module>
```

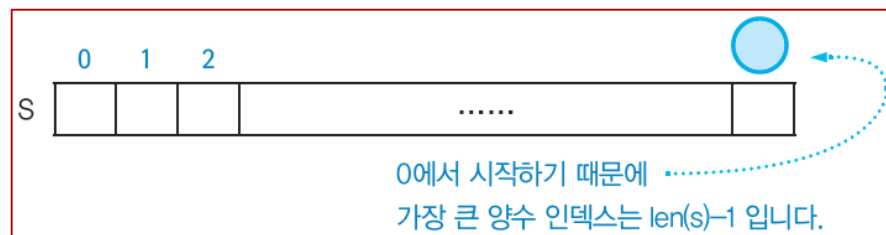
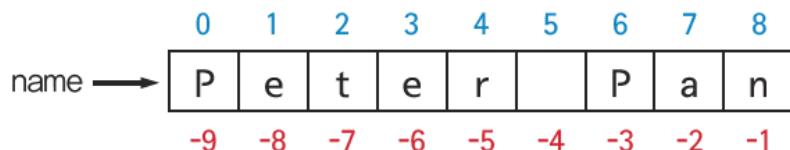
```
    t[-2] = 'n'
```

```
TypeError: 'str' object does not support item assignment
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

### ◆ 인덱싱

- 인덱스를 이용하여 문자열의 특정 위치에 있는 문자에 접근하는 것
- 인덱싱할 때는 인덱스 범위가 중요함



```
>>> name = 'Peter Pan'
>>> print(name[2])
t
>>> print(name[-3])
P
```

```
>>> print(name[9])    # 없는 인덱스
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    print(name[9])
IndexError: string index out of range
```



## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

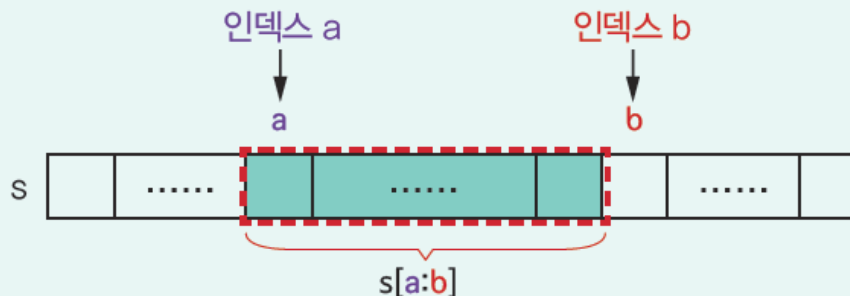
### ◆ 슬라이싱

- 문자열의 일부분을 잘라보는 것
- 두 가지 형태가 있음 - `s[a:b]`, `s[a:b:c]`

- 문자열 `s`에서 인덱스 `a`부터 시작해서 인덱스 `b` 바로 전까지 슬라이싱함.
- 반드시 인덱스 `a`가 인덱스 `b`의 왼쪽에 있어야 함.

문자열 `s`에서 인덱스 `a`부터 시작해서 인덱스 `b` 바로 전까지 슬라이싱

`s[a:b]`

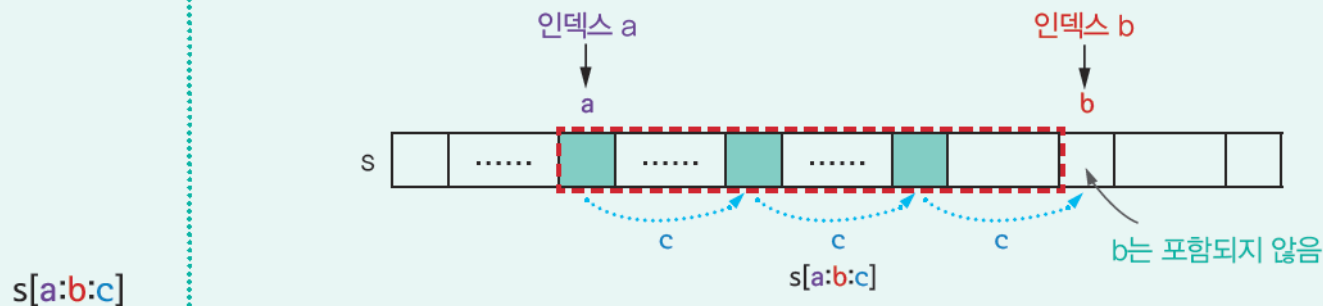


# 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

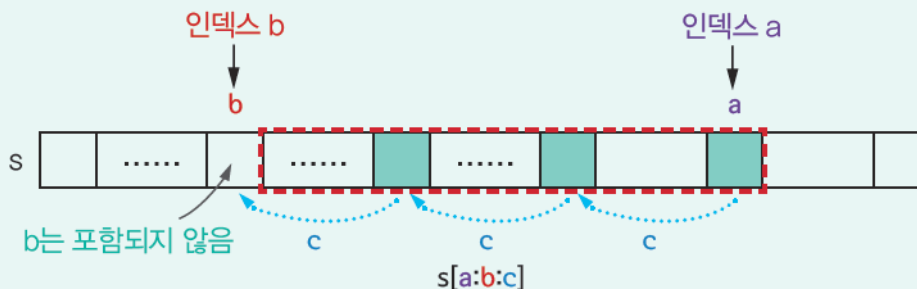
## ◆ 슬라이싱

- 문자열  $s$ 에서 인덱스  $a$ 부터 시작해서 인덱스  $b$  바로 전까지  $c$  간격으로 슬라이싱함.
- $c$ 가 양수인 경우와 음수인 경우를 나누어서 봐야 함.

<  $c$ 가 양수인 경우 > -  $a$ 가  $b$ 의 왼쪽에 있어야 함.



<  $c$ 가 음수인 경우 > -  $a$ 가  $b$ 의 오른쪽에 있어야 함.



## 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

### ◆ 슬라이싱 예제 `alpha[a:b]` 형태

`alpha = 'abcdefghijklmnopqrstuvwxyz'`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

① `alpha[a:b]` - 'a부터 b 바로 전까지'

```
>>> alpha[3:10]
'defghij'
```

```
>>> alpha[20:25]
'uvwxy'
```

```
>>> alpha[-15:-7]
'lmnopqrs'
```

```
>>> alpha[9:-5]
'jklmnopqrstu'
```

만약에 인덱스 a가 인덱스 b보다 오른쪽에 있다면 빈 문자열이 나온다

```
>>> alpha[10:5]
''
```

```
>>> alpha[25:20]
''
```

```
>>> alpha[-7:-15]
''
```

```
>>> alpha[-5:9]
''
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

② alpha[:b] - alpha[0:b]와 같다 (인덱스 0 생략)

```
>>> alpha[:5]
'abcde'
```

```
>>> alpha[:-10]
'abcdefghijklmnop'
```

③ alpha[a:] - alpha[a:len(alpha)]와 같다 (len(alpha) 생략)

```
>>> alpha[20:]
'vwxyz'
>>> alpha[20:len(alpha)]
'vwxyz'
>>> alpha[20:len(alpha)-1] # 'z' 포함안됨
'vwxy'
```

```
>>> alpha[-10:]
'qrstuvwxyz'
>>> alpha[-10:-1]
'qrstuvwxy'
>>> alpha[-10:0] # -10이 0보다 오른쪽.
''
```

④ alpha[:] - 문자열 전체

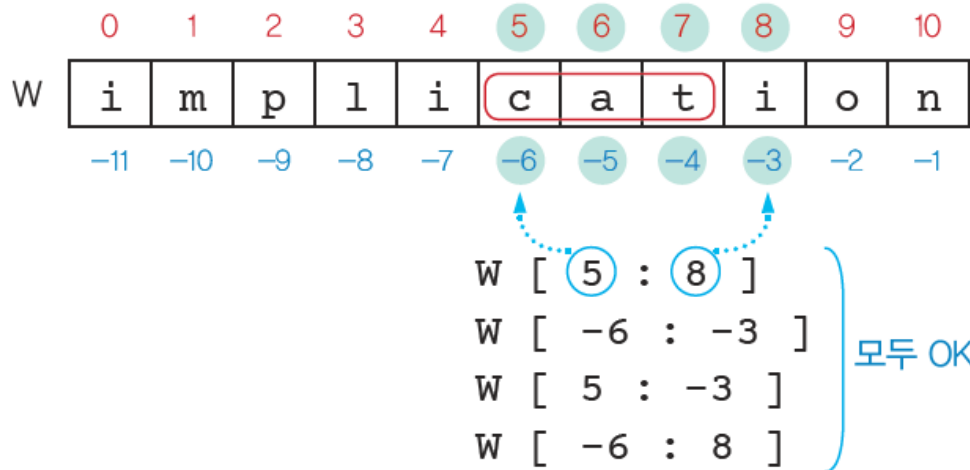
```
>>> alpha[:]
'abcdefghijklmnopqrstuvwxyz'
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

- [참고] 인덱스에 양수와 음수를 섞어서 사용할 수 있다



나? 나만 꺼내고 싶은데..



```
>>> w = 'implication'
>>> w[5:8]
'cat'
>>> w[-6:-3]
'cat'
>>> w[5:-3]
'cat'
>>> w[-6:8]
'cat'
```

## 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

### ◆ 슬라이싱 예제 `alpha[a:b:c]` 형태

`alpha = 'abcdefghijklmnopqrstuvwxyz'`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

① `alpha[a:b:c]` - 'a부터 b 바로 전까지 c 간격으로'

- c가 양수 - 반드시 a가 b보다 왼쪽에 있는 인덱스여야 함

```
>>> alpha[10:20:2]
'kmoqs'
```

```
>>> alpha[-20:-12:4]
'gk'
```

```
>>> alpha[3:-10:2]
'dfhjlnp'
```

- c가 양수인데 a가 b보다 오른쪽에 있으면 빈 문자열이 나옴

```
>>> alpha[5:1:2]
''
```

```
>>> alpha[-5:-10:2]
''
```

```
>>> alpha[10:-20:2]
''
```

## 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

① alpha[a:b:c] - 'a부터 b 바로 전까지 c 간격으로'

- c가 음수 - 반드시 a가 b보다 오른쪽에 있는 인덱스여야 함

```
>>> alpha[23:11:-3]
'xuro'
```

```
>>> alpha[-5:-20:-5]
'vql'
```

```
>>> alpha[20:-20:-3]
'uroli'
```

- c가 음수인데 a가 b보다 왼쪽에 있으면 빈 문자열이 나옴

```
>>> alpha[10:20:-2]
''
```

```
>>> alpha[-20:-10:-2]
''
```

```
>>> alpha[10:-10:-2]
''
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

② alpha[:b:c] - 인덱스 a를 생략한 경우

- c가 양수 - [0:b:c]와 같음 (0부터 b 전까지 c 간격으로)

```
>>> alpha[:10:3]
'adgj'
```

```
>>> alpha[:-10:3]
'adgjmp'
```

```
>>> alpha[:-1:2]
'acegikmoqsuwy'
```

- c가 음수 - [-1:b:c]와 같음 (-1부터 b 전까지 c 간격으로)

```
>>> alpha[:10:-2]
'zxvtrpnl'
```

```
>>> alpha[:-5:-2]
'zx'
```

```
>>> alpha[:0:-1]
'zyxwvutsrqponmlkjihgfedcb'
```



## 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

③ alpha[a::c] - 인덱스 b를 생략한 경우

- c가 양수 - a부터 끝까지 c 간격으로

```
>>> alpha[10::2]
'kmoqsuwy'
```

```
>>> alpha[3::5]
'dinsx'
```

```
>>> alpha[-10::3]
'qtwz'
```

- C가 음수 - a부터 0 까지 c 간격으로

```
>>> alpha[10::-2]
'kigeca'
```

```
>>> alpha[-5::-7]
'voha'
```

```
>>> alpha[-1::-1]
'zyxwvutsrqponmlkjihgfedcba'
```

## 4. 문자열 인덱싱 (indexing), 슬라이싱 (slicing) 하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

### ④ alpha[::-c]

- c가 양수 - 0부터 끝까지 c 간격으로
- c가 음수 - -1부터 0 까지 c 간격으로

```
>>> alpha[::2]
'acegikmoqsuwy'
```

```
>>> alpha[::-2]
'zxvtrpnljhfdb'
```

```
>>> alpha[::-1]
'zyxwvutsrqponmlkjihgfedcba'
```

- [::-1]은 문자열을 거꾸로 만들어 줌.

```
>>> a = 'python'
>>> b = a[::-1]
>>> print(a, b)
python nohtyp
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

⑤ 간격 c를 생략한 경우 - 콜론이 한 개인 경우와 같음

```
>>> alpha[2:5]
'cde'
>>> alpha[-5:-10:]
''
>>> alpha[: -10:]
'abcdefghijklmnop'
```

```
>>> alpha[2:5]
'cde'
>>> alpha[-5:-10]
''
>>> alpha[: -10]
'abcdefghijklmnop'
```

## 4. 문자열 인덱싱(indexing), 슬라이싱(slicing)하기

alpha = 'abcdefghijklmnopqrstuvwxyz'

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
alpha	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

⑥ alpha[:] - 문자열 전체

```
>>> alpha[:]
'abcdefghijklmnopqrstuvwxyz'
```

[ : ], [ :: ] 문자열 전체

[ :: -1 ] 거꾸로 된 문자열

## 5. +, \*, in, not in, del 연산자

### ◆ + 연산 : 두 문자열 연결하기

```
>>> first_name = 'Bill'
>>> last_name = 'Gates'
>>> full_name = first_name + last_name      # 두 문자열을 연결함
>>> full_name
'BillGates'
>>> full_name = first_name + ' ' + last_name  # 중간에 스페이스를 넣음
>>> full_name
'Bill Gates'
```

## 5. +, \*, in, not in, del 연산자

### ◆ + 연산 : 두 문자열 연결하기

- 문자열 수정에도 + 연산은 유용함

```
>>> s = 'computer on a desk'
>>> s = s[:12] + 'the' + s[13:]      # 'a'를 'the'로 수정함
>>> s
'computer on the desk'
```

## 5. +, \*, in, not in, del 연산자

### ◆ + 연산 : 두 문자열 연결하기

- 문자열은 immutable하기 때문에 다음과 같이 다루어짐

```
>>> s = 'computer on a desk'
```

```
>>> id(s)
```

```
35482944
```

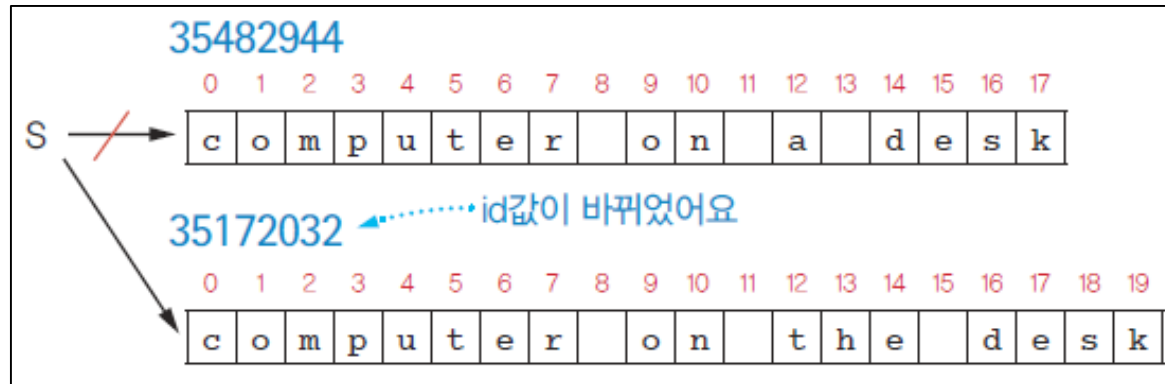
```
>>> s = s[:12] + 'the' + s[13:]
```

```
>>> id(s)
```

```
35172032
```

```
>>> s
```

```
'computer on the desk'
```



## 5. +, \*, in, not in, del 연산자

### ◆ + 연산 : 두 문자열 연결하기

- += 연산자를 사용할 수 있음

```
>>> title = 'Girl'
```

```
>>> title += ' in a Blue Coat'    # 'Girl' 뒤에 'in a Blue coat'를 연결함
```

```
>>> title
```

```
'Girl in a Blue Coat'
```



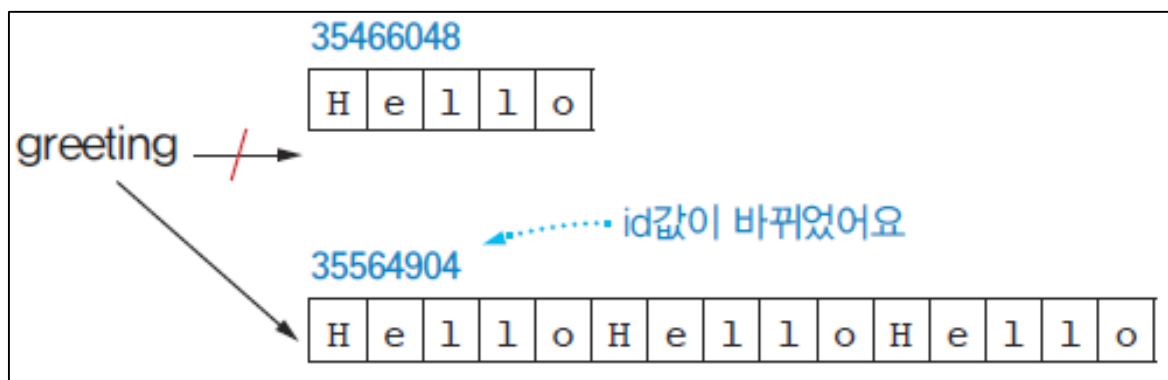
## 5. +, \*, in, not in, del 연산자

### ◆ \* 연산 : 문자열 반복하기

```
>>> greeting = 'Hello'
>>> greeting * 3
'HelloHelloHello'
>>> greeting      # greeting은 그대로임
'Hello'
```

```
>>> greeting = 'Hello'
>>> greeting *= 3
>>> greeting      # greeting이 변함
'HelloHelloHello'
```

```
>>> greeting = 'Hello'
>>> id(greeting)
35466048
>>> greeting *= 3
>>> id(greeting)
35564904
>>> greeting
'HelloHelloHello'
```



## 5. +, \*, in, not in, del 연산자

- ◆ in / not in - 문자열에 포함 관계를 알 수 있음 (True/False)

```
>>> vowels = 'aeiouAEIOU'
```

```
>>> 'o' in vowels
```

```
True
```

```
>>> 'M' in vowels
```

```
False
```

```
>>> 'K' not in vowels
```

```
True
```

```
>>> x = 'programming'
```

```
>>> 'gram' in x
```

```
True
```

```
>>> 'mm' in x
```

```
True
```

```
>>> 'mm' not in x
```

```
False
```

## 5. +, \*, in, not in, del 연산자

### ◆ del 연산자

```
>>> name = 'Harry'
>>> del name[2]          # 'r'을 하나 지우려고 함
Traceback (most recent call last):
.....
TypeError: 'str' object doesn't support item deletion
```

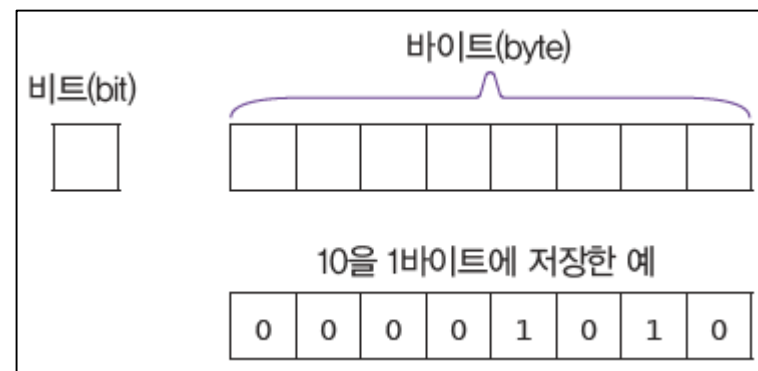
- del 연산자로 문자열을 통째로 삭제할 수 있음

```
>>> name = 'Harry'
>>> id(name)
58434368
>>> del name          # 문자열 객체를 통째로 삭제함
>>> name
.....
NameError: name 'name' is not defined
```

## 6. 아스키 코드와 ord() / chr() 함수

### ◆ 2진수와 비트(bit), 바이트(byte)

- 컴퓨터는 0 또는 1만 저장함 - bit
- byte = 8bits
- bin() 함수 - 10진수를 2진수로 변환해서 반환하는 함수

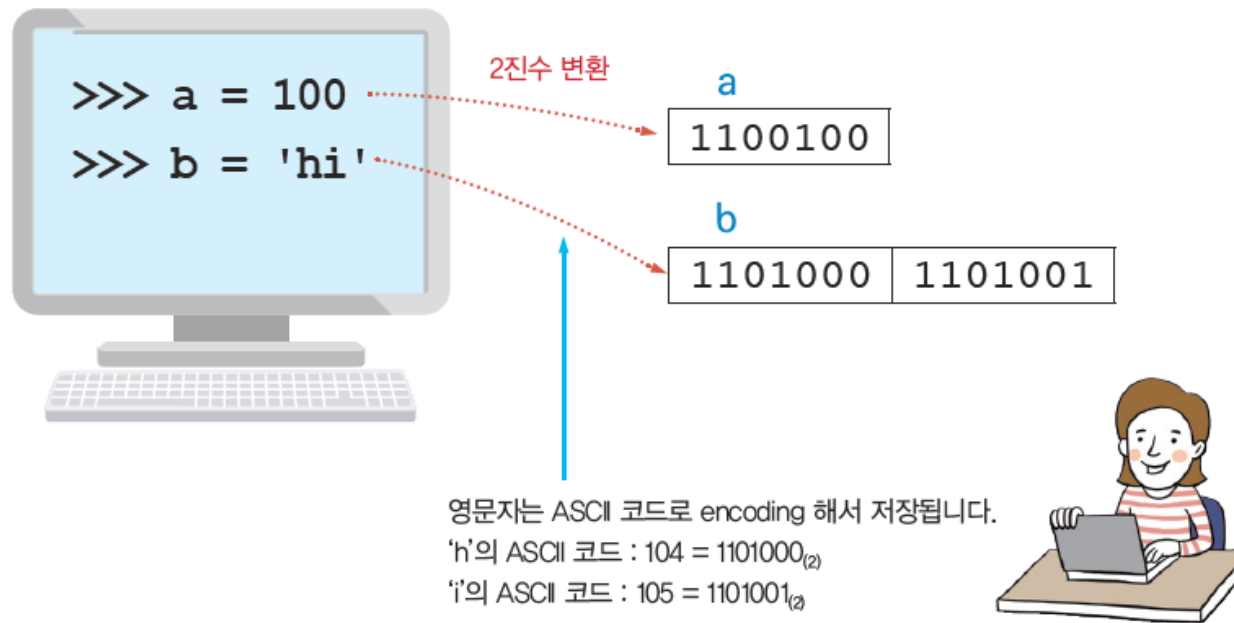


```
>>> a = 10
>>> x = bin(a)    # bin() 함수는 괄호 안에 넣은 수를 이진수로 변환시킴.
>>> print(x)      # 앞에 붙는 '0b'는 이진수라는 뜻임.
0b1010
```

## 6. 아스키 코드와 ord() / chr() 함수

### ◆ 아스키 코드 (ASCII)

- 영어 인코딩 방식
- 영어 대소문자, 특수 문자, 숫자 등에 아스키 코드가 할당되어 있음



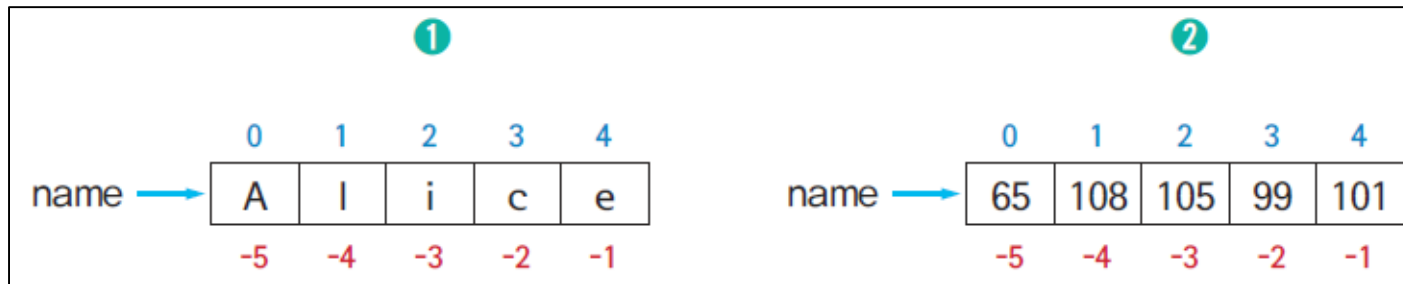
교재 p.88의 아스키 코드표 참고할 것

## 6. 아스키 코드와 ord() / chr() 함수

### ◆ 아스키 코드 (ASCII)

```
>>> name = 'Alice'
```

①처럼 저장될거라고 생각하지만, 실제로 컴퓨터에는 ②와 같이 저장됨



### ▪ ord(x)/chr(x) 함수

ord(x)	인수 x 에는 문자 한 개를 넣어야 합니다. 반환값은 x의 아스키코드 값입니다.	<pre>&gt;&gt;&gt; a = ord('D') &gt;&gt;&gt; print(a) 68 &gt;&gt;&gt; b = chr(100) &gt;&gt;&gt; print(b) d</pre>
chr(x)	인수 x 에는 아스키코드 한 개를 넣어야 합니다. 반환값은 아스키코드 x에 해당하는 문자입니다.	

## 7. 문자열에 함수 적용하기

◆ `len()`, `max()`, `min()`, `sum()`, `sorted()`, `reversed()`

<code>len(S)</code>	문자열 S의 길이를 반환합니다.
<code>max(S)</code>	문자열 S의 문자들 중에서 ASCII 코드 값이 가장 큰 문자를 반환합니다.
<code>min(S)</code>	문자열 S의 문자들 중에서 ASCII 코드 값이 가장 작은 문자를 반환합니다.
<code>sum(S)</code>	문자열에는 <code>sum</code> 함수를 적용할 수 없습니다.
<code>sorted(S)</code>	문자열 S의 문자들을 <code>ascii</code> 코드 값으로 정렬하여 결과를 반환합니다. 결과는 리스트로 반환합니다.
<code>reversed(S)</code>	문자열 S를 역순으로 바꾸어 줍니다.

## 7. 문자열에 함수 적용하기

### ◆ len() 함수

```
>>> language = 'python'
>>> len(language)
6
>>> book_title = 'Alice Wonderland'
>>> len(book_title)
16
```

# 스페이스도 하나의 문자임.

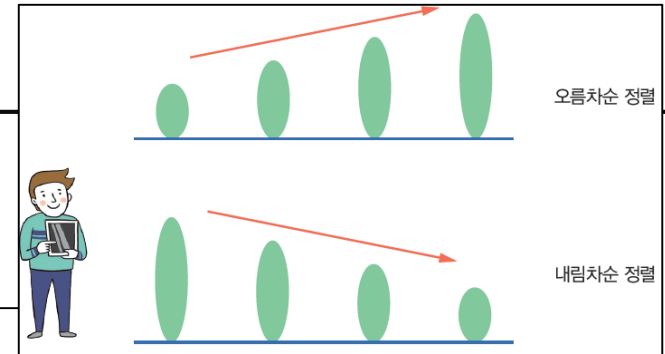
### ◆ max() / min() 함수

<pre>&gt;&gt;&gt; device = 'computer' &gt;&gt;&gt; max(device) 'u' &gt;&gt;&gt; min(device) 'c'</pre>	<pre>&gt;&gt;&gt; full_name = 'Alice Lee' &gt;&gt;&gt; max(full_name) 'l' &gt;&gt;&gt; min(full_name) ''</pre>	<pre>&gt;&gt;&gt; name = 'Alice' &gt;&gt;&gt; max(name) 'i' &gt;&gt;&gt; min(name) 'A'</pre>
---	--	--



# 7. 문자열에 함수 적용하기

## ◆ sorted() 함수



```
>>> device = 'computer'
```

```
>>> sorted(device)    # 아스키 코드값이 작은 문자부터 큰 문자 순서대로 리스트로 출력함.
```

```
['c', 'e', 'm', 'o', 'p', 'r', 't', 'u']
```

```
>>> language = 'python'
```

```
>>> sorted(language)
```

```
['h', 'n', 'o', 'p', 't', 'y']
```

```
>>> book = 'Harry Potter'    # 아스키코드는 소문자>대문자>스페이스 순서로 작아요
```

```
>>> sorted(book)
```

```
[' ', 'H', 'P', 'a', 'e', 'o', 'r', 'r', 'r', 't', 't', 'y']
```

```
>>> sorted(language, reverse=True)    # reverse=True 추가하면 내림차순 정렬함.
```

```
['y', 't', 'p', 'o', 'n', 'h']
```

## 7. 문자열에 함수 적용하기

- ◆ reversed() 함수 - 문자열을 역순으로 바꾸어서 반환하는 함수

```
>>> thing = 'desk'
>>> R = reversed(thing)          # thing을 역순으로 바꾸어 변수 R에 저장함.
>>> print(R) # R에는 reversed 객체가 저장됨.
<reversed object at 0x021D2E50>
>>> L = list(R)                  # R의 결과를 리스트로 보고 싶으면 list() 함수를 적용해야 함.
>>> print(L)
['k', 's', 'e', 'd']
```

## 8. 문자열 메소드

### ◆ 내장 함수와 메소드

- 내장 함수 - 파이썬이 기본적으로 제공하는 일반적인 함수들
  - `print()`, `id()`, `type()`, `len()`, `max()`, `min()`, `print()`, `bin()`, `ord()`, `chr()`, `sorted()`, ...
  - `dir(__builtins__)`라고 하면 내장 함수 목록을 볼 수 있음
- 문자열 메소드
  - 문자열에만 적용할 수 있는 함수들
  - `dir(str)`이라고 하면 문자열 메소드 목록을 볼 수 있음

## 8. 문자열 메소드

### ◆ 문자열 메소드 예제

- upper() 메소드 - 문자열을 대문자로 만드는 메소드

```
>>> book = 'Alice in Wonderland'
```

```
>>> book2 = book.upper( )      # 문자열 book에 upper( ) 메소드를 적용함.
```

```
>>> print(book2)               # book2는 book을 모두 대문자로 만든 문자열임.
```

```
ALICE IN WONDERLAND
```

```
>>> print(book)                # book은 그대로임.
```

```
Alice in Wonderland
```

## 8. 문자열 메소드

### ◆ 문자열 메소드 예제

내장 함수 `int()`는 독립적으로 쓰였어요. 즉, `int()` 앞에 아무 것도 붙지 않아요.

```
>>> a = int(10.5)
>>> print(a)
10
>>> b = pow(2, 10)
>>> print(b)
1024
```

`pow()`도 내장 함수니까 앞에 아무 것도 붙지 않아요.

`upper()` 메소드는 문자열에만 적용 가능한 메소드예요. 그래서 문자열.`upper()` 형태로 써야 해요. `upper()` 메소드는 문자열을 모두 대문자로 변환해 줍니다.

```
>>> book = 'Alice in Wonderland'
>>> book2 = book.upper()
>>> print(book2)
ALICE IN WONDERLAND
>>> book3 = book.title()
>>> print(book3)
Alice In Wonderland
```

`title()` 메소드 역시 문자열에만 적용 가능한 메소드예요. 그리고 문자열에 있는 각 단어의 첫 번째 문자만을 대문자로 변환해 줍니다.

## 8. 문자열 메소드

### ◆ 문자열 메소드 예제

```
>>> abs(-10)
>>> y = abs(-20)
>>> s = 'alice'
>>> t = s.upper()
```

프롬프트

내장 함수

문자열. 메소드()

이 메소드는 이 문자열에만 적용함.

## 8. 문자열 메소드

### ◆ 문자열 메소드 목록

```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

## 8. 문자열 메소드

### ◆ 대소문자 변환하기

- capitalize(), title(), upper(), lower() swapcase()
- capitalize() - 첫 문자만을 대문자로 변환함

<pre>&gt;&gt;&gt; x = 'hello world' &gt;&gt;&gt; y = x.capitalize() &gt;&gt;&gt; x      # x는 그대로임. 'hello world' &gt;&gt;&gt; y      # y는 새 문자열임. 'Hello world'</pre>	<pre>&gt;&gt;&gt; x = 'HELLO world' &gt;&gt;&gt; y = x.capitalize() &gt;&gt;&gt; x 'HELLO world' &gt;&gt;&gt; y 'Hello world'</pre>	<pre>&gt;&gt;&gt; x = 'hELLO wORLD' &gt;&gt;&gt; y = x.capitalize() &gt;&gt;&gt; x 'hELLO wORLD' &gt;&gt;&gt; y 'Hello world'</pre>
---	---	---



## 8. 문자열 메소드

### ◆ 대소문자 변환하기

- title() - 각 단어의 첫 문자만을 대문자로 변환함

```
>>> x = 'how are you?'  
>>> y = x.title()  
>>> x  
'how are you?'  
>>> y  
'How Are You?'
```

```
>>> x = 'HOW ARE YOU?'  
>>> y = x.title()  
>>> x  
'HOW ARE YOU?'  
>>> y  
'How Are You?'
```

```
>>> x = 'hOW aRE yOU?'  
>>> y = x.title()  
>>> x  
'hOW aRE yOU?'  
>>> y  
'How Are You?'
```

## 8. 문자열 메소드

### ◆ 대소문자 변환하기

- `upper()` - 문자열을 모두 대문자로 변환하여 반환함
- `lower()` - 문자열을 모두 소문자로 변환하여 반환함

```
>>> x = 'helLO woRLd'
>>> y = x.upper()
>>> x          # x는 그대로임.
'helLO woRLd'
>>> y
'HELLO WORLD'
```

```
>>> x = 'Happy NEW Year 2018'
>>> x.upper()
'HAPPY NEW YEAR 2018'
>>> x.lower()
'happy new year 2018'
```

**참고** 문자열 메소드들은 모두 `문자열.메소드()`로 사용합니다. 이때 문자열 자리에 직접 문자열을 넣어도 되요.

```
>>> y = 'good'.upper()  # 문자열을 직접 넣고 메소드를 적용했어요.
>>> y
'GOOD'
```

## 8. 문자열 메소드

### ◆ 정렬하기

#### ▪ center()

인수	1개	문자열.center(문자열 폭), 만약에 문자열 폭이 문자열 길이보다 작으면 무시합니다.
	2개	문자열.center(문자열 폭, 빈 칸을 채울 문자)
반환값	가운데로 정렬한 문자열	

#### 인수 1개인 경우

```
>>> y = x.center(10)
>>> x # x는 그대로임.
'python'
>>> y # 10칸 안에 가운데 정렬함.
'  python  '
```

```
>>> x = 'hello world'
>>> x.center(15)
'  hello world  '
>>> x.center(3)
'hello world' # x의 길이가 3보다 크기
               때문에 3은 무시함.
```

#### 인수 2개인 경우

```
>>> x = 'happy'
>>> x.center(20, '*')
'*****happy*****'
>>> x.center(10, '@')
'@@happy@@@'
# 빈 칸 채울 문자
```

```
>>> x = 'programming'
>>> x.center(len(x)+4, '$')
'$$programming$$'
>>> x.center(5, '$')
'programming' # x의 길이가 5보다 크
                기 때문에 5는 무시함.
```

## 8. 문자열 메소드

### ◆ 정렬하기

- ljust() - 왼쪽 정렬

인수 1개인 경우	<pre>&gt;&gt;&gt; x = 'python' &gt;&gt;&gt; y = x.ljust(10) &gt;&gt;&gt; x      # x는 그대로임. 'python' &gt;&gt;&gt; y      # 반환된 문자열 'python   '</pre>	<pre>&gt;&gt;&gt; x = 'geometry' &gt;&gt;&gt; x.ljust(15)  # 15칸 잡아서 x를 'geometry      '  # 왼쪽 정렬함. &gt;&gt;&gt; x.ljust(5)   # x의 길이가 5보다 크기 'geometry'       # 때문에 5를 무시함.</pre>
인수 2개인 경우	<pre>&gt;&gt;&gt; book = 'geometry' &gt;&gt;&gt; book.ljust(15, '+') 'geometry+++++++' &gt;&gt;&gt; book.ljust(3, '+') 'geometry' # 빈 칸 채울 문자</pre>	<pre>&gt;&gt;&gt; fruit = 'watermelon' &gt;&gt;&gt; fruit.ljust(len(fruit)+5, '@') 'watermelon@@@@@'</pre>

## 8. 문자열 메소드

### ◆ 정렬하기

- `rjust()` - 오른쪽 정렬

인수 1개인 경우	<pre>&gt;&gt;&gt; x = 'python' &gt;&gt;&gt; y = x.rjust(10) &gt;&gt;&gt; x 'python' &gt;&gt;&gt; y '   python'</pre>	<pre>&gt;&gt;&gt; x = 'geometry' &gt;&gt;&gt; x.rjust(10) '  geometry' &gt;&gt;&gt; x.rjust(5) 'geometry'</pre>
인수 2개인 경우	<pre>&gt;&gt;&gt; book = 'geometry' &gt;&gt;&gt; book.rjust(15, '+') '+++++++geometry' &gt;&gt;&gt; book.rjust(3, '+') 'geometry'</pre>	<pre>&gt;&gt;&gt; fruit = 'watermelon' &gt;&gt;&gt; fruit.rjust(len(fruit)+5, '#') '#####watermelon'</pre>

## 8. 문자열 메소드

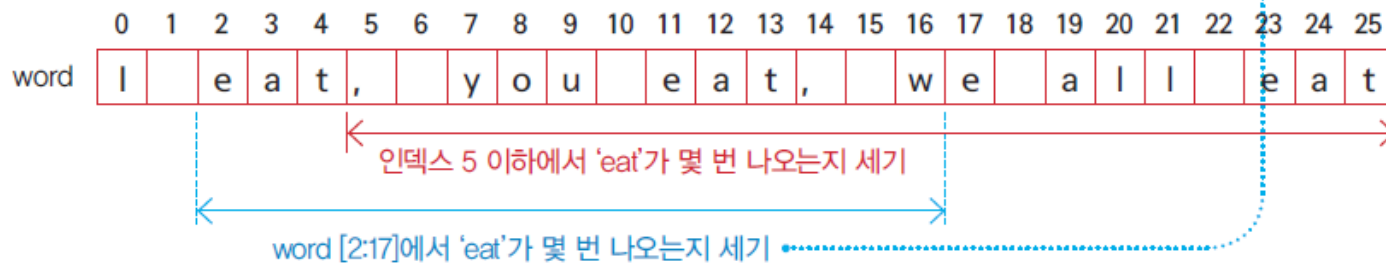
### ◆ 특정 문자열 개수 세기 - count()

- count(x) - 문자열에서 x가 몇 번 나오는지 찾아서 반환함

인수	1개	문자열.count(x)
	2개	문자열.count(x, a) - 문자열[a:]에서 문자열 x를 찾습니다.
	3개	문자열.count(x, a, b) - 문자열[a:b]에서 문자열 x를 찾습니다.
반환값	문자열에서 x가 몇 번 나오는지 그 횟수를 반환합니다. 만약에 문자열에서 x가 한 번도 나오지 않는다면 0을 반환합니다.	

## 8. 문자열 메소드

인수 1개인 경우	<pre>&gt;&gt;&gt; x = 'library' &gt;&gt;&gt; y = x.count('r') &gt;&gt;&gt; print(y) # 'r'2번 2</pre>	<pre>&gt;&gt;&gt; text = 'I love python.' &gt;&gt;&gt; text.count('python') 1 &gt;&gt;&gt; text.count('.') 1</pre>
인수 2개인 경우	<pre>&gt;&gt;&gt; x = 'ambassador' &gt;&gt;&gt; x.count('a', 2) 2 &gt;&gt;&gt; x.count('a', 4) 1</pre>	<pre>&gt;&gt;&gt; word = 'I eat, you eat, we all eat' &gt;&gt;&gt; word.count('eat', 5) ← 2 &gt;&gt;&gt; word.count('eat', 15) 1</pre>
인수 3개인 경우	<pre>&gt;&gt;&gt; x = 'professor' &gt;&gt;&gt; x.count('o', 1, 4) 1 &gt;&gt;&gt; x.count('o', 1, 8) 2</pre>	<pre>&gt;&gt;&gt; word = 'I eat, you eat, we all eat' &gt;&gt;&gt; word.count('eat', 2, 17) ← 2 &gt;&gt;&gt; word.count('you', 2, 10) 1</pre>



## 8. 문자열 메소드

### ◆ 특정 문자열의 인덱스 찾기 - index(), rindex(), find(), rfind()

- index(x) - 문자열에서 x가 몇 번 나오는지 찾아서 반환함

인수	1개	문자열.index(x)
	2개	문자열.index(x, a) 문자열[a:]에 x가 있으면 x가 시작하는 인덱스를 반환합니다.
	3개	문자열.index(x, a, b) 문자열[a:b]에 x가 있으면 x의 시작 인덱스를 반환합니다.
반환값	문자열에서 x가 시작하는 위치의 인덱스를 반환합니다. ◀.....x가 없으면 ValueError가 발생합니다.	



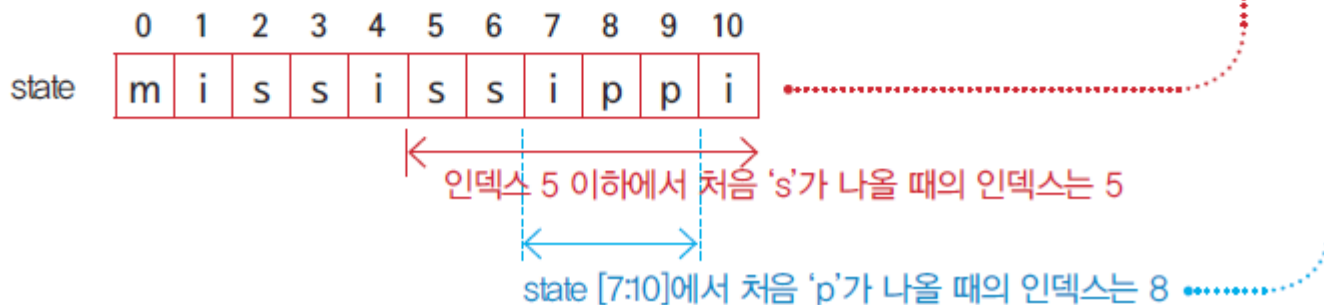
## 8. 문자열 메소드

### ◆ 특정 문자열의 인덱스 찾기 - index(), rindex(), find(), rfind()

- index(x) - 문자열에서 x가 몇 번 나오는지 찾아서 반환함

```
>>> s = 'python'
>>> s.index('t') # s에서 't'의 위치 알려줌.
2
>>> s.index('ho')
3
>>> s.index('a') # 'a'는 s에 없음.
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in
    <module>
      s.index('a')
ValueError: substring not found
```

```
>>> state = 'mississippi'
>>> state.index('s')
2
>>> state.index('s', 5)
5
>>> state.index('p', 7, 10)
8
```




## 8. 문자열 메소드

- ◆ 특정 문자열의 인덱스 찾기 - `index()`, `rindex()`, `find()`, `rfind()`
  - `rindex(x)` - right index의 의미임.

```
>>> x = 'this is a chair and that is a desk'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
t	h	i	s		i	s		a		c	h	a	i	r		a	n	d		t	h	a	t		i	s		a		d	e	s	k



x[2:10]에서 오른쪽으로부터 가장 먼저 나오는 'is'의 인덱스는 5

```
>>> x.rindex('is')
```

```
25
```

```
>>> x.rindex('is', 15) # x[15:]
```

```
25
```

```
>>> x.rindex('is', 2, 10) # x[2:10]
```

```
5
```

```
>>> x.rindex('is', 10, 15) # x[10:15]에는 'is' 없음.
```

```
.....
ValueError: substring not found
```

## 8. 문자열 메소드

### ◆ 특정 문자열의 인덱스 찾기 - index(), rindex(), find(), rfind()

- find(x) / rfind(x) - index(x)/rindex(x)와 기본적으로 같음. 다른 점은 없는 문자열에 대해서 ValueError를 내지 않고, -1을 반환함.

```
>>> x = 'programming language'
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
p	r	o	g	r	a	m	m	i	n	g			l	a	n	g	u	a	g	e

```
>>> x.find('r') # 첫번째 'r'의 인덱스 반환함.
1
>>> x.find('a') # 첫번째 'a'의 인덱스 반환함.
5
>>> x.find('a', 10) # x[10:]
13
>>> x.find('a', 2, 7) # x[2:7]
5
>>> x.find('b') # 'b'는 x에 없음.
-1
```

```
>>> x.rfind('r') ←····· # 오른쪽으로부터
4                       첫 번째 'r'의 인덱
>>> x.rfind('a')       스 반환함.
17
>>> x.rfind('a', 10)   # x[10:]
17
>>> x.rfind('a', 2, 7) # x[2:7]
5
>>> x.rfind('b')
-1
```

## 8. 문자열 메소드

### ◆ 문자열의 시작과 끝 확인하기 - startswith() / endswith()

- startswith(x) - 문자열이 특정 문자열 x로 시작하는지 판단함. 결과는 True/False임.

인수	1개	문자열.startswith(x)
	2개	문자열.startswith(x, a) 문자열[a:]이 x로 시작하는지 판단합니다.
	3개	문자열.startswith(x, a, b) 문자열[a:b]이 x로 시작하는지 판단합니다.
반환값	True / False	

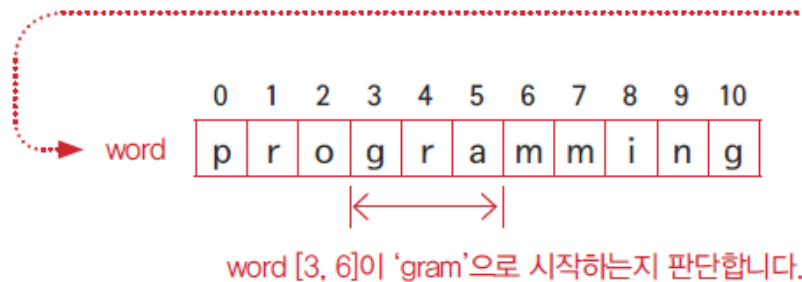
## 8. 문자열 메소드

### ◆ 문자열의 시작과 끝 확인하기 - startswith() / endswith()

```
>>> word = 'programming'

>>> word.startswith('p')
True
>>> word.startswith('pr')
True
>>> word.startswith('prop')
False
>>> word.startswith('programming')
True
>>> word.startswith('') # 빈 문자열
True
>>> word.startswith(' ') # 스페이스
False

>>> word.startswith('gram', 2)
False
>>> word.startswith('gram', 3)
True
>>> word.startswith('gram', 3, 6)
False
>>> word.startswith('gram', 3, 7)
True
>>> word.startswith('g', len(word)-1)
True
>>> word.startswith('g', len(word))
False
```



## 8. 문자열 메소드

### ◆ 문자열의 시작과 끝 확인하기 - startswith() / endswith()

- endswith(x) - 문자열이 특정 문자열 x로 끝나는지 판단함. 결과는 True/False임.

인수	1개	문자열.endswith(x)
	2개	문자열.endswith(x, a) 문자열[a:]이 x로 끝나는지 판단합니다.
	3개	문자열.endswith(특정 문자열, a, b) 문자열[a:b]가 x로 끝나는지 판단합니다.
반환값	True / False	

## 8. 문자열 메소드

## ◆ 출력 포맷 지정하기 - format()

- ### ① {}가 포함된 문자열에 format() 메소드 적용하기

```
>>> x = 'I like { } and { }'  
>>> x.format('apple', 'strawberry')  
'I like apple and strawberry'  
>>> x.format('math', 'english')  
'I like math and english'
```

첫 번째 인수    두 번째 인수

'I like {} and {}'.format('apple', 'strawberry')

'I like {} and {}'.format('math', 'english')

## 8. 문자열 메소드

### ◆ 출력 포맷 지정하기 - format()

#### ② {숫자}가 포함된 문자열에 format() 메소드 적용하기

```
>>> x = 'I like {0} with {1}. {0} is my favorite drink'
>>> x.format('coffee', 'milk')
'I like coffee with milk, coffee is my favorite drink'
>>> x.format('milk', 'chocolate')
'I like milk with chocolate, milk is my favorite drink'
```

{0} 자리에 들어감.

{1} 자리에 들어감.

인수 0      인수 1

↓            ↓

'I like {0} with {1}. {0} is my favorite drink'.format('coffee', 'milk')

↑            ↑            ↑            ↑

'I like {0} with {1}. {0} is my favorite drink'.format('milk', 'chocolate')



## 8. 문자열 메소드

### ◆ 출력 포맷 지정하기 - format()

#### ③ { 변수 }가 포함된 문자열에 format() 메소드 적용하기

```
>>> x = '{name} is {age} years old'
>>> x.format(name='Alice', age=10)
'Alice is 10 years old.'
>>> x.format(name='Paul', age=7)
'Paul is 7 years old.'
```

'{name} is {age} years old'.format(name = 'Alice', age = 10)



'{name} is {age} years old'.format(name = 'Paul', age = 7)



## 8. 문자열 메소드

### ◆ 출력 포맷 지정하기 - format\_map()

- format() 메소드를 변형한 형태
- format\_map() 메소드의 인수는 사전 형태이어야 함. { 변수 }가 포함된 문자열에 format() 메소드 적용하기



```
>>> v = {'a':10, 'b':20, 'c':30}
>>> '{a}, {b}, {c}'.format_map(v) # {a}에 10, {b}에 20, {c}에 30이 들어갑니다.
'10, 20, 30'
```


## 8. 문자열 메소드

- ◆ 문자열 결합하기/분리하기 - join(), split(), rsplit(), splitlines()
  - join() - 여러 개의 문자열을 하나의 문자열로 합해서 반환함.

인수 자료형	사용 예제
리스트	<pre>&gt;&gt;&gt; fruits = ['melon', 'apple', 'kiwi'] # 문자열로 구성된 리스트여야 합니다. &gt;&gt;&gt; s = '***' &gt;&gt;&gt; t = s.join(fruits) &gt;&gt;&gt; print(t) melon***apple***kiwi</pre> <p>fruits의 문자열들을 s로 연결합니다.</p>
튜플	<pre>&gt;&gt;&gt; book = ('java', 'python', 'c++') # 문자열로 구성된 튜플이어야 합니다. &gt;&gt;&gt; s = '~~~~' &gt;&gt;&gt; t = s.join(book) &gt;&gt;&gt; print(t) java~~~~python~~~~c++</pre> <p>book의 문자열들을 s로 연결합니다.</p>

## 8. 문자열 메소드

### ◆ 문자열 결합하기/분리하기 - join(), split(), rsplit(), splitlines()

집합	<pre>&gt;&gt;&gt; color = {'red', 'blue', 'white', 'black'} # 문자열로 구성된 집합이어야 합니다. &gt;&gt;&gt; s = ' * ' &gt;&gt;&gt; t = s.join(color) # 집합은 순서 개념이 없어서 어떤 순서로 결합될지 알 수 없음. &gt;&gt;&gt; print(t) # 저장 순서대로 결합하지 않을 수 있어요. blue * red * black * white</pre>
사전	<p>사전은 '키'들을 결합해 줘요. 그래서 사전의 '키'가 문자열로 되어 있어야 join() 메소드를 적용할 수 있습니다.</p>  <pre>&gt;&gt;&gt; english = {'one':1, 'two':2, 'three':3} # 키가 문자열이어야 합니다. &gt;&gt;&gt; s = '^^^' &gt;&gt;&gt; t = s.join(english) # '^^^'로 키들만 결합합니다. &gt;&gt;&gt; print(t) one^^^two^^^three</pre>
문자열	<pre>&gt;&gt;&gt; name = 'Peter Pan' # 문자 하나하나를 분리한 후에 결합합니다. &gt;&gt;&gt; s = '~*~' &gt;&gt;&gt; t = s.join(name) # '~*~'로 name의 문자들을 결합합니다. &gt;&gt;&gt; print(t) P~*~e~*~t~*~e~*~r~*~ ~*~P~*~a~*~n</pre>

## 8. 문자열 메소드

### ◆ 문자열 결합하기/분리하기 - join(), split(), rsplit(), splitlines()

#### ▪ join()

```
>>> score = [80, 90, 77] # 정수로 구성된 리스트에는 join() 메소드가 수행되지 않아요.
>>> s = '*****'
>>> s.join(score) # 에러 메시지를 보면 str(문자열)이 와야 한다고 되어 있어요.
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    s.join(score)
TypeError: sequence item 0: expected str instance, int found
```



결과 = 문자열.join(    )



여기에는 문자열들로 구성된 iterable 데이터가 들어가야 합니다.  
컨테이너 자료형인 문자열, 리스트, 튜플, 집합, 사전이 iterable 자료형입니다.

## 8. 문자열 메소드

- ◆ 문자열 결합하기/분리하기 - `join()`, `split()`, `rsplit()`, `splitlines()`
  - `split()` - `join()`의 반대 메소드로, 하나의 문자열을 분리하여 리스트로 반환함.

인수	없는 경우	문자열.split() 스페이스로 문자열을 분리합니다.
	1개	문자열.split(구별자) 구별자에도 문자열이 와야 하고, 구별자로 문자열을 분리합니다.
	2개	문자열.split(구별자, n) 구별자에는 문자열, n에는 정수가 와야 합니다. 문자열을 구별자로 분리하는데 n번 분리합니다.
반환값	분리된 문자열들을 리스트에 넣어 반환합니다.	

## 8. 문자열 메소드

### ◆ 문자열 결합하기/분리하기 - join(), split(), rsplit(), splitlines()

인수가 없는 경우	<pre>&gt;&gt;&gt; text = 'Python is a widely used computer language.' &gt;&gt;&gt; result = text.split() # 스페이스로 문자열을 분리합니다. &gt;&gt;&gt; result                # 문자열을 분리해서 리스트에 저장합니다. ['Python', 'is', 'a', 'widely', 'used', 'computer', 'language.']</pre>
인수가 1개인 경우	<pre>&gt;&gt;&gt; friends = 'Paul--Tom--Alice--Kelly--Cindy' &gt;&gt;&gt; friends.split('--') # 문자열 friends를 '--'로 분리합니다 ['Paul', 'Tom', 'Alice', 'Kelly', 'Cindy']</pre>
인수가 2개인 경우	<pre>&gt;&gt;&gt; words='desk-pencil-computer-book-eraser-chair' &gt;&gt;&gt; words.split('-', 1) # 문자열 words를 '-'로 한 번 분리합니다. ['desk', 'pencil-computer-book-eraser-chair'] &gt;&gt;&gt; words.split('-', 2) # 문자열 words를 '-'로 두 번 분리합니다. ['desk', 'pencil', 'computer-book-eraser-chair'] &gt;&gt;&gt; words.split('-', 3) # 문자열 words를 '-'로 세 번 분리합니다. ['desk', 'pencil', 'computer', 'book-eraser-chair']</pre>

## 8. 문자열 메소드

### ◆ 문자열 결합하기/분리하기 - join(), split(), rsplit(), splitlines()

- rsplit() - right split

```
>>> words = 'desk-pencil-computer-book-eraser-chair'
>>> words.rsplit('-')      # 그냥 split() 메소드와 차이가 없음.
['desk', 'pencil', 'computer', 'book', 'eraser', 'chair']
>>> words.rsplit('-', 3)   # 오른쪽 끝에서부터 세 번 분리함.
['desk-pencil-computer', 'book', 'eraser', 'chair']
```

- splitlines() - '\n'을 기준으로 문자열을 자르는 메소드

```
>>> letter = 'Dear Alice, \n hello to your mom. \n from Carol'
>>> letter.splitlines()
['Dear Alice,', 'Say hello to your mom.', 'from Carol']
>>> letter = 'Dear Alice, Say hello to your mom. from Carol' # '\n' 없음.
>>> letter.splitlines()
['Dear Alice, Say hello to your mom. from Carol']
```



## 8. 문자열 메소드

- ◆ 문자열에서 불필요한 문자 떼어내기 - strip(), rstrip(), lstrip()
  - strip() - 양쪽에서 불필요한 문자 떼어냄
  - rstrip() - 오른쪽에서 불필요한 문자 떼어냄
  - lstrip() - 왼쪽에서 불필요한 문자 떼어냄

인수	없는 경우	문자열.strip() 문자열 양쪽에 ' ', '\t', '\n', '\r'이 있다면 떼어냅니다.
	1개	문자열.strip(x) 문자열 양쪽에서 x를 떼어 냅니다(x도 문자열이어야 해요).
반환값	특정 문자열을 떼어낸 나머지 문자열을 반환합니다.	

## 8. 문자열 메소드

### ◆ 문자열에서 불필요한 문자 떼어내기 - strip(), rstrip(), lstrip()

인수가 없는 경우

문자 양 끝에 있는 스페이스를 떼어냅니다.

```
>>> line1 = ' hello world '
>>> result = line1.strip() # line1 양쪽에서 ' '(스페이스) 떼어낸 문자
>>> result                # 열을 반환함.
'hello world'
>>> len(line1), len(result) # 떼어낸 후 문자열의 길이가 줄어듦.
(18, 13)
>>> line1.rstrip()        # 오른쪽에서 떼어낸 문자열을 반환함.
' hello world'
>>> line1.lstrip()        # 왼쪽에서 떼어낸 문자열을 반환함.
'hello world '
```

---

```
>>> line2 = 'hello world\n'
>>> result = line2.strip() # '\n' 떼어냄.
>>> result
'hello world'
>>> len(line2), len(result)
(12, 11)
```

## 8. 문자열 메소드

### ◆ 문자열에서 불필요한 문자 떼어내기 - strip(), rstrip(), lstrip()

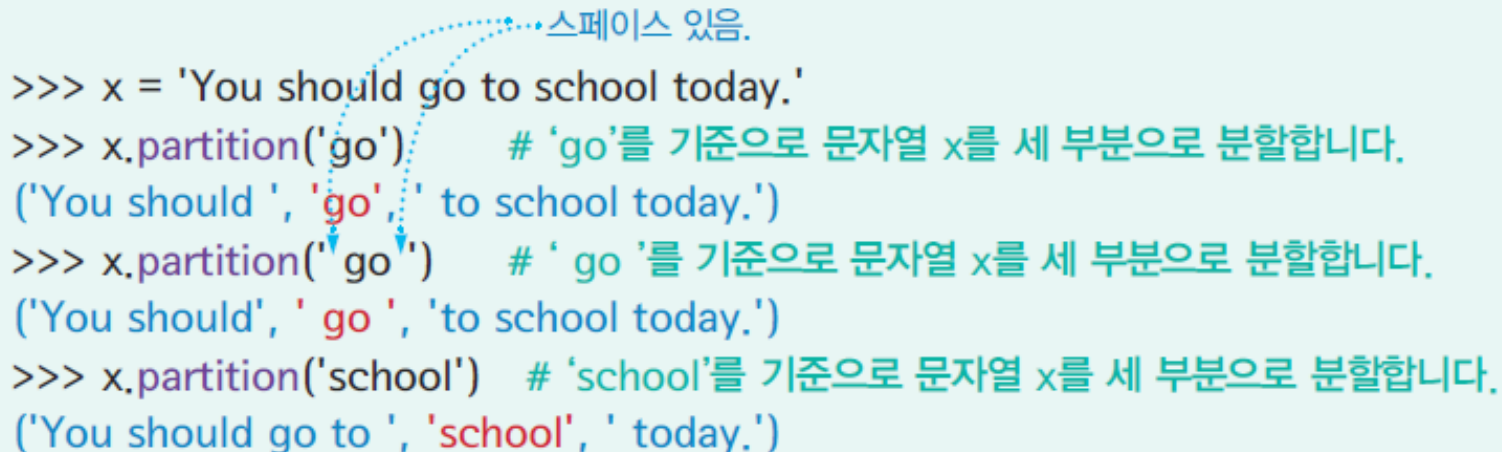
인수가 있는 경우

```
>>> line = '***hello*****world*****'
>>> result1 = line.strip('*') # 양 옆에서 '*'떼어냄.
>>> result1
'hello*****world'
>>> result2 = line.rstrip('*') # 오른쪽 끝에서 '*'떼어냄.
>>> result2
'***hello*****world'
>>> result3 = line.lstrip('*') # 왼쪽 끝에서 '*'떼어냄.
>>> result3
'hello*****world*****'
```

## 8. 문자열 메소드

### ◆ 문자열 분할하기 - partition(), rpartition()

- 문자열 인수 한 개는 반드시 필요함. 그 문자열을 기준으로 문자열을 세 부분으로 분할해서 튜플로 반환함.



```
>>> x = 'You should go to school today.'
>>> x.partition('go')      # 'go'를 기준으로 문자열 x를 세 부분으로 분할합니다.
('You should ', 'go', ' to school today.')
>>> x.partition('go ')     # ' go '를 기준으로 문자열 x를 세 부분으로 분할합니다.
('You should', ' go ', 'to school today.')
>>> x.partition('school')  # 'school'를 기준으로 문자열 x를 세 부분으로 분할합니다.
('You should go to ', 'school', ' today.')
```

## 8. 문자열 메소드

### ◆ 문자열 분할하기 - partition(), rpartition()

- 분할 기준이 되는 문자열이 여거 개 있으면 가장 먼저 나오는 문자열을 기준으로 분할함.

```
>>> y = 'I learn swimming and I learn dancing'
>>> y.partition('learn') # 문자열 y를 앞에 나오는 'learn'을 기준으로 분할합니다.
('I ', 'learn', 'swimming and I learn dancing')
```

learn이 2개인데 앞의 learn을 기준으로 분할함.

- rpartition은 오른쪽에서 왼쪽으로 가면서 분할함.

```
>>> y = 'I learn swimming and I learn dancing'
>>> y.rpartition('learn') # 오른쪽부터 보았을 때 먼저 나오는 'learn'을 기준으로 분할합니다.
('I learn swimming and I ', 'learn', ' dancing')
```

## 8. 문자열 메소드

### ◆ 문자열 분할하기 - partition(), rpartition()

- 만약에 인수가 문자열에 없다면, 다음과 같이 분할함.

```
>>> x = 'You should go to school today.'  
>>> x.partition('gg')    # x에 'gg'가 없습니다.  
( 'You should go to school today.', '', '' )  
>>> x.rpartition('gg')  
( '', '', 'You should go to school today.' )
```

## 8. 문자열 메소드

### ◆ 각각의 문자를 대체하기 - maketrans(), translate()

- maketrans() - 길이가 같은 문자열 두 개를 인수로 넣어야 함. 첫 번째 인수는 문자열에 있는 문자들이어야 하고, 두 번째 인수는 새 문자들임.

```
>>> x = str.maketrans('abc', 'ABC') # str.maketrans( , )로 사용합니다.  
>>> x      # {'a': 'A', 'b': 'B', 'c': 'C'} 사전을 반환합니다(아스키코드로 저장).  
{97: 65, 98: 66, 99: 67}
```

- translate() - maketrans() 메소드 결과로 나온 사전을 인수로 넣어야 함.

```
>>> s = 'ambassador'  
>>> t = str.maketrans('adr', 'ADR') # 'a'를 'A'로, 'd'를 'D'로, 'r'을 'R'로 변환하려고 합니다.  
>>> print(t)  
97: 65, 100: 68, 114: 82      # a(97), d(100), r(114), A(65), D(68), R(82)  
>>> s.translate(t)           # s를 t에 따라 적용해 줍니다.  
'AmbAssADoR'
```

## 8. 문자열 메소드

### ◆ 문자열 대체하기 - replace()

- 문장에서 단어를 대체할 때 유용함.

인수	2개	문자열.replace(x, y) 문자열 x를 모두 문자열 y로 대체합니다.
	3개	문자열.replace(x, y, n), 문자열 x n개를 문자열 y로 대체합니다.
반환값	대체한 결과 문자열을 반환합니다.	

```
>>> text = 'I scream you scream we all scream'
>>> text.replace('scream', 'play')           # 'scream'을 'play'로 대체합니다.
'I play you play we all play'
>>> text.replace('scream', 'play', 2)        # 'scream'을 'play'로 대체합니다(두 개만).
'I play you play we all scream'
>>> text.replace('scream', 'play', 1)        # 'scream'을 'play'로 대체합니다(한 개만).
'I play you scream we all scream'
```



## 8. 문자열 메소드

### ◆ 문자열에 0 채우기 - zfill()

```
>>> x = 'hello world'
>>> x.zfill(20)      # 문자열 x의 앞 부분에 0을 채워서 문자열의 길이를 20으로 만듦.
'000000000hello world'
>>> x                # x는 그대로입니다.
'hello world'
>>> x.zfill(10)
'hello world'
```

문자열의 길이보다 작은 수이므로 0을 못채웁니다.

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- istitle() - 문자열을 구성하는 단어들이 모두 대문자로 시작하는지 판단함.

```
>>> x = 'Alice wonderland'
>>> x.istitle()
False
>>> y = 'Alice Wonderland'
>>> y.istitle()
True
```

```
>>> s = 'Snow WhiTe'
>>> s.istitle()
False
>>> t = 'Snow 123 White' # 숫자 무시합니다.
>>> t.istitle()
True
```

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- `isalnum()` - 문자열이 영문자와 숫자로만 구성되어 있는지 판단함.

```
>>> a = 'python2018'
>>> b = 'python'
>>> c = '2018'
>>> d = 'python_2018'
>>> e = 'python-version'
>>> f = 'python 2018'
```

```
>>> a.isalnum() # 영문자와 숫자로 구성되면 True입니다.
True
>>> b.isalnum() # 영문자로만 구성되어도 True입니다.
True
>>> c.isalnum() # 숫자로만 구성되어도 True입니다.
True
>>> d.isalnum() # 특수문자 '_' 있음.(underscore)
False
>>> e.isalnum() # 특수문자 '-' 있음.(dash)
False
>>> f.isalnum() # 스페이스 때문에 False입니다.
False
```

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- `isalpha()` - 문자열이 모두 알파벳인 경우에만 True 반환함.

```
>>> a = 'python'
>>> b = 'python programming'
>>> a.isalpha()
True
>>> b.isalpha()    # 스페이스가 중간에 들어가서 False예요.
False
```

- `isdigit()` - 문자열이 모두 숫자인 경우에만 True 반환함.

```
>>> number1 = '12489'
>>> number1.isdigit()
True
>>> number2 = '123-456'    # '-'가 중간에 있어서 False예요.
>>> number2.isdigit()
False
```

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- `isidentifier()` - 문자열이 식별자이면 True 반환함. 대표적인 식별자는 변수명임.

```
>>> number = 10
>>> number1 = 20
>>> number_2 = 30
>>> number-3 = 40 # '-'는 사용할 수 없어요.
SyntaxError: can't assign to operator
>>> 5number = 50 # 숫자로 시작하면 안 돼요.
SyntaxError: invalid syntax
```

```
>>> 'number'.isidentifier()
True
>>> 'number1'.isidentifier()
True
>>> 'number_2'.isidentifier()
True
>>> 'number-3'.isidentifier()
False
>>> '5number'.isidentifier()
False
```

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- islower() / isupper()

```
>>> 'world war'.islower()
True
>>> 'world wAr'.islower() # 'A'False
False
>>> 'world war 2'.islower()
True
```

```
>>> '1234'.islower() # 소문자가 없습니다.
False
>>> '1234a'.islower()
True
```

```
>>> x = 'PYTHON'
>>> x.isupper()
True
>>> y = 'pyTHon'
>>> y.isupper()
False
```

```
>>> s = 'PYTHON PROGRAMMING'
>>> s.isupper()
True
>>> t = 'PYTHON VERSION 3 !!'
>>> t.isupper()
True
```

## 8. 문자열 메소드

### ◆ is로 시작하는 메소드들

- isspace()

```
>>> x = ' '  
>>> x.isspace()  
True  
>>> y = 't'  
>>> y.isspace()  
False
```

```
>>> y = ' a ' ← 스페이스 a가 같이 있음.  
>>> y.isspace()  
False
```

**참고** isdigit( )와 비슷한 일을 하는 메소드로 isdecimal( )과 isnumeric( )이 있어요.

## 9. 정리

---

- ◆ 파이썬에서 문자열을 저장하고 처리하는 일은 아주 중요함.
- ◆ 문자열이 어떻게 저장되는지와 인덱스 개념을 알아야 함.
- ◆ 문자열 인덱싱, 슬라이싱을 알아야 함.
- ◆ 문자열 메소드를 사용할 수 있어야 함.