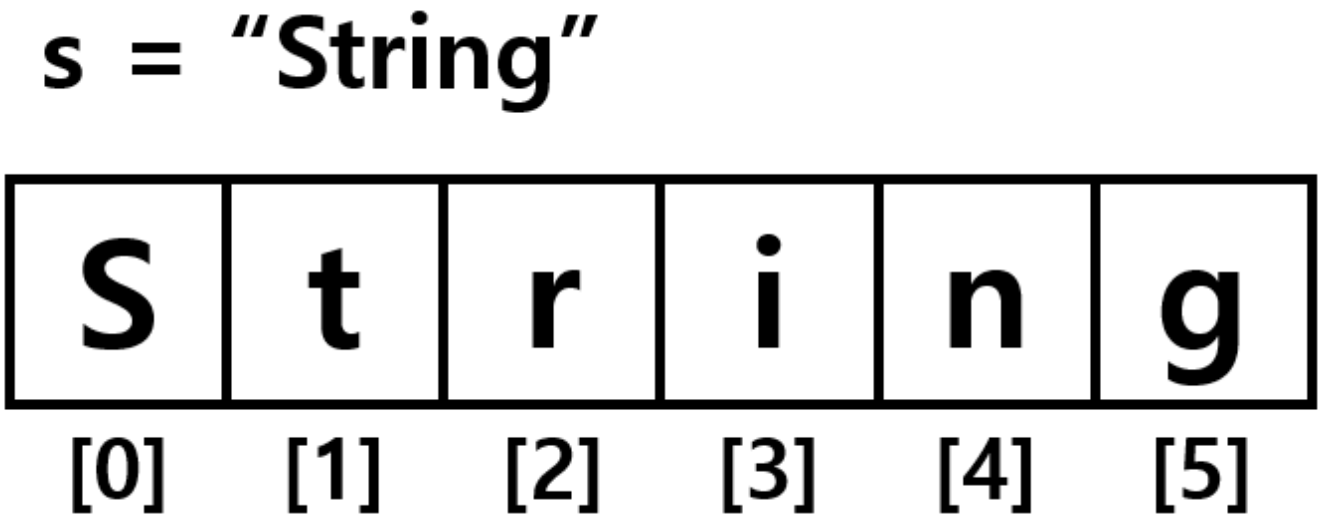
문자열(String)

문자열 정의

- 문자, 단어 등으로 구성된 문자들의 집합
- 시퀀스 자료형(sequence data type)




```
s = "String"
print(s)
print(s[0])
print(s[1])
print(s[2])
print(s[3])
print(s[4])
print(s[5])
```

```
String
S
t
r
i
n
g
```

```
s = "문자열"
print(s)
print(s[0])
print(s[1])
print(s[2])
```

```
문자열
문
자
열
```

문자열 생성

- 파이썬에서 문자열은 작은따옴표(') 또는 큰 따옴표(")로 표현

```
print('Hello')
print("파이썬은 재미있다.")
print('''파이썬은 심플하다.''')
print("""파이썬은 문자열 처리가 뛰어나다""")
```

```
Hello
파이썬은 재미있다.
파이썬은 심플하다.
파이썬은 문자열 처리가 뛰어나다
```

따옴표가 있는 문자열 생성

- 문자열에 작은따옴표가 있을 경우, 큰따옴표로 둘러싸서 표현
- 문자열에 큰따옴표가 있을 경우, 작은따옴표로 둘러싸서 표현
- 이스케이프 코드 \를 이용하여 작은따옴표(')와 큰따옴표(")를 문자열에 포함

```
string = "Python's built-in string classes"
print(string)
string = '모두가 "파이썬을 매우 쉽다."라고 말한다.'
print(string)
string = 'Python₩'s built-in string classes'
print(string)
string = "모두가 ₩"파이썬을 매우 쉽다.₩"라고 말한다."
print(string)
```

```
Python's built-in string classes
모두가 "파이썬을 매우 쉽다."라고 말한다.
Python's built-in string classes
모두가 "파이썬을 매우 쉽다."라고 말한다.
```

이스케이프 문자(Escape Character)

- 특수 문자
- 문자 앞에 백슬래시(\)를 사용
- 문자열에서 특수 문자를 표현하는데 사용

이스케이프 문자	이름
\	역슬래시(backslash)
\'	작은 따옴표(single quote)

이스케이프 문자	이름
\"	큰 따옴표(double quote)
\a	알람(alarm)
\b	백스페이스(backspace)
\f	폼 피드(form feed)
\n	라인피드(linefeed)
\r	캐리지 리턴(carriage return)
\t	수평 탭(tab)
\v	수직 탭(tab)
\ooo	8진수 문자(octal value)
\xhh	16진수 문자(hexadecimal value)

여러 줄이 있는 문자열 생성

- 이스케이프 문자(\n)를 이용하여 여러 줄이 있는 문자열 생성 가능

```
text = "동해물과 백두산이 마르고 닳도록₩n하느님이 보우하사 우리나라 만세₩n무궁화 삼천리 화려강산₩n
print(text)
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한 사람, 대한으로 길이 보전하세.
```

- 작은 따옴표 3개 또는 큰 따옴표 3개를 이용하여 여러 줄이 있는 문자열 생성

```
text = """동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산
대한 사람, 대한으로 길이 보전하세."""
print(text)
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산
대한 사람, 대한으로 길이 보전하세.
```

문자열 연산(String Operators)

문자열 더하기

- 연산자를 사용하여 문자열 연결

```
s1 = "동해물과 "
```

```
s2 = "백두산이"
print(s1 + s2)
```

```
동해물과 백두산이
```

문자열 곱하기

- * 연산자를 사용하여 문자열 반복

```
s = "String "
print(s * 3)
```

```
String String String
```

문자열 길이(length)

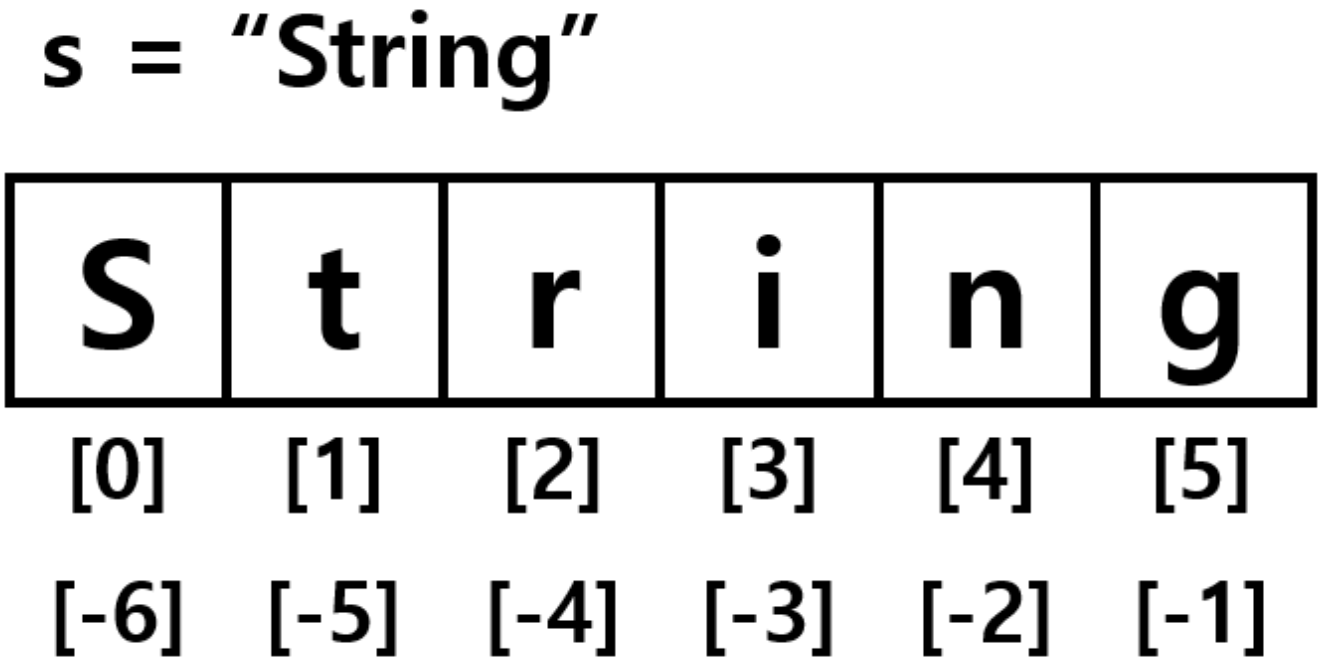
- 문자열 길이를 구하는 `len()` 함수

```
s = "String "
print(len(s))
print(len(s * 3))
```

```
7
21
```

문자열 인덱싱(indexing)

- 문자열은 리스트처럼 문자 하나하나가 상대적인 주소(offset)를 가짐
- 이 주소를 사용해 할당된 값을 가져오는 인덱싱을 사용



```
s = "String"
print(s)
print(s[-6])
print(s[-5])
print(s[-4])
print(s[-3])
print(s[-2])
print(s[-1])
```

```
String
S
t
r
i
n
g
```

문자열 슬라이싱(slicing)

- 문자열의 주소를 이용하여 문자열을 조각(부분)을 추출

```
s = "SuanLab - Suan Computer Laboratory"
print(s)
print(s[0:7])
print(s[:7])
print(s[10:])
print(s[-10:])
print(s[-19:-11])
```

```
SuanLab - Suan Computer Laboratory
SuanLab
SuanLab
Suan Computer Laboratory
Laboratory
Computer
```

문자열 메소드(String Methods)

- 파이썬은 문자열 처리를 아주 쉽게 처리할 수 있는 많은 메소드들을 제공
- 문자열을 다루는 주요 메소드들을 표로 정리

함수	설명
`capitalize()`	첫 문자를 대문자로하고, 나머지 문자를 소문자로 하는 문자열 반환
`casefold()`	모든 대소문자 구분을 제거
`count(sub, [, start[, end]])`	[start, end] 범위에서 부분 문자열 sub의 중복되지 않은 수를 반환
`find(sub [, start [, end]])`	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 인덱스를 반환. sub가 발견되지 않는
`rfind(sub [, start [, end]])`	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 큰 인덱스를 반환. sub가 발견되지 않

함수	설명
`index(sub [, start [, end]])`	find()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
`rindex(sub [, start [, end]])`	rfind()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
`isalnum()`	문자열의 모든 문자가 영숫자로 1개 이상 있으면 True, 아니면 False 반환
`isalpha()`	문자열의 모든 문자가 영문자로 1개 이상 있으면 True, 아니면 False 반환
`isdecimal()`	문자열의 모든 문자가 10진수 문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
`isdigit()`	문자열의 모든 문자가 숫자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
`isnumeric()`	문자열의 모든 문자가 수치형이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
`isidentifier()`	문자열이 유효한 식별자인 경우 True 반환
`isspace()`	문자열 내에 공백 문자가 있고, 문자가 1개 이상 있으면 True, 그렇지 않으면 False
`istitle()`	문자열이 제목이 있는 문자열에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
`islower()`	문자열의 모든 문자가 소문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
`isupper()`	문자열의 문자가 모두 대문자에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
`join(iterable)`	iterable에 있는 문자열에 연결된 문자열을 반환
`center(width [, fillchar])`	길이 너비만큼 중앙정렬된 문자열 반환
`ljust(width [, fillchar])`	너미만큼의 문자열에서 왼쪽 정렬된 문자열을 반환
`rjust(width [, fillchar])`	너미만큼의 문자열에서 오른쪽 정렬된 문자열을 반환
`lower()`	모든 대소문자가 소문자로 변환된 문자열을 반환
`upper()`	문자열에서 모든 문자를 대문자로 변환한 문자열을 반환
`title()`	문자열에서 첫 글자만 대문자이고 나머지는 소문자인 문자열 반환
`swapcase()`	문자열에서 소문자를 대문자로 대문자를 소문자로 변환한 문자열 반환
`strip([chars])`	문자열 양쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
`lstrip([chars])`	문자열 왼쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
`rstrip([chars])`	문자열 오른쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
`partition(sep)`	문자열에서 첫번째 sep를 기준으로 분할하여 3개의 튜플을 반환
`rpartition(sep)`	문자열에서 마지막 sep를 기준으로 분할하여 3개의 튜플을 반환
`replace(old, new[,count])`	문자열의 모든 old를 new로 교체한 문자열을 반환
`split(sep=None, maxsplit=1)`	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
`rsplit(sep=None, maxsplit=1)`	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
`splitlines([keepends])`	문자열에서 라인 단위로 구분하여 리스트를 반환
`startswith(prefix [, start[, end]])`	[start, end] 범위에서 지정한 prefix로 시작하면 True, 아니면 False 반환
`endswith(suffix [, start[, end]])`	[start, end] 범위에서 지정한 suffix로 끝나면 True, 아니면 False 반환
`zfill(width)`	너비 만큼의 문자열에서 비어있는 부분에 '0'이 채워진 문자열 반환

capitalize() / casefold()

- 문자열 "string"에 대해 `capitalize()`를 실행하면, 첫 문자가 대문자인 "String" 문자열로 변환
- 문자열 "String"에 대해 `casefold()`를 실행하면, 모든 문자가 소문자인 "string" 문자열로 변환

```
s = "string"
print(s)
s = s.capitalize()
print(s)
s = s.casefold()
print(s)
```

```
string
String
string
```

count()

- 문자열 "string string"에 포함된 문자 's'의 갯수를 반환하도록 `count()`를 실행
- 문자 's'가 총 2개 존재하므로 2를 반환
- 부분 문자열 "str"이 몇개인지 반환하는 것도 가능하고, 부분 문자열 "str"이 총 2개 존재하므로 2를 반환

```
s = "string string"
print(s)
print(s.count('s'))
print(s.count("str"))
```

```
string string
2
2
```

find() / rfind()

- 문자열 "string string"에서 문자 's'의 해당 위치를 `find()`를 통해 반환
- 문자열 "string string"에서 부분 문자열 "ing"의 해당 위치를 `find()`를 통해 반환
- 문자열 "string string"에서 문자 's'를 해당 위치를 `rfind()`를 통해 오른쪽부터 탐색하여 가장 큰 인덱스를 반환
- 문자열 "string string"에서 부분 문자열 "ring"를 해당 위치를 `rfind()`를 통해 오른쪽부터 탐색하여 가장 큰 인덱스를 반환

```
s = "string string"
print(s.find('s'))
print(s.find("ing"))
print(s.rfind('s'))
print(s.rfind("ring"))
```

```
0
3
7
9
```

index() / rindex()

- 문자열 "string string"에서 문자 's'의 해당 위치를 `index()`를 통해 반환
- 문자열 "string string"에서 부분 문자열 "ing"의 해당 위치를 `index()`를 통해 반환
- 문자열 "string string"에서 문자 's'를 해당 위치를 `rindex()`를 통해 오른쪽부터 탐색하여 가장 큰 인덱스를 반환

- 문자열 "string string"에서 부분 문자열 "ring"를 해당 위치를 `rindex()`를 통해 오른쪽부터 탐색하여 가장 큰 인덱스를 반환
- 문자열 "string string"에서 문자 'z'를 해당 위치를 `rindex()` 함수를 통해 오른쪽부터 탐색하지만 찾을 수 없어서 'ValueError' 발생

```
s = "string string"
print(s.index('s'))
print(s.index("ing"))
print(s.rindex('s'))
print(s.rindex("ring"))
#print(s.rindex('z'))
```

```
0
3
7
9
```

isalnum()

- `isalnum()`은 문자열에 알파벳이나 숫자가 1개 이상 있으면 True 반환
- 문자열 "string"은 알파벳으로 구성 되었으므로 True 반환
- 문자열 "한글"은 알파벳으로 구성 되어있으므로 True 반환
- 문자열 "!@#"은 특수기호로 구성 되어있으므로 False 반환
- 문자열 "123"은 숫자들로 구성 되어있으므로 True 반환

```
print("string".isalnum())
print("한글".isalnum())
print("!@#".isalnum())
print("123".isalnum())
```

```
True
True
False
True
```

isalpha()

- `isalpha()`는 문자열에 알파벳이 1개 이상 있으면 True 반환
- 문자열 "string"은 알파벳으로 구성 되었으므로 True 반환
- 문자열 "한글"은 알파벳으로 구성 되어있으므로 True 반환
- 문자열 "!@#"은 특수기호로 구성 되어있으므로 False 반환
- 문자열 "123"은 숫자들로 구성 되어있으므로 Flase 반환

```
print("string".isalpha())
print("한글".isalpha())
print("!@#".isalpha())
print("123".isalpha())
```

```
True
True
False
False
```

isdecimal()

- `isdecimal()`는 문자열의 모든 문자가 10진수 문자이면 True 반환
- 문자열 "123"은 모두 10진수 문자열이므로 True 반환
- 문자열 "1.23"은 실수형 문자열이므로 False 반환

```
print("123".isdecimal())
print("1.23".isdecimal())
```

```
True
False
```

isdigit()

- `isdigit()`는 문자열의 모든 문자가 숫자일 때 True 반환
- 문자열 "123"은 모든 문자가 숫자에 해당하므로 True 반환
- 문자열 "1.23"은 문자 중 숫자에 해당안되는 . 이 존재하여 False

```
print("123".isdigit())
print("1.23".isdigit())
```

```
True
False
```

isnumeric()

- `isnumeric()`는 문자열의 모든 문자가 수치형일 때 True 반환
- 문자열 "123"은 모든 문자가 숫자에 해당하므로 True 반환
- 문자열 "1.23"은 문자 중 수치형에 해당안되는 . 이 존재하여 False

```
print("123".isnumeric())
print("1.23".isnumeric())
```

```
True
False
```

isidentifier()

- `isidentifier()`는 문자열이 파이썬에서 사용하는 식별자인 경우 True 반환
- 문자열 "123"은 식별자로 사용되지 않는 문자열이라서 False 반환

- 문자열 "True"는 식별자로 사용되기 때문에 True 반환
- 문자열 "print"는 식별자로 사용되기 때문에 True 반환

```
print("123".isidentifier())
print("True".isidentifier())
print("print".isidentifier())
```

```
False
True
True
```

isspace()

- 문자열 " "은 문자열 내에 공백 문자가 있어서 True 반환
- 문자열 " 1 "은 문자열 내에 공백 아닌 문자가 있어서 False 반환

```
print(" ".isspace())
print(" 1 ".isspace())
```

```
True
False
```

istitle()

- 문자열 "String"은 문자열 내에 제목과 같이 첫 글자가 대문자여서 True 반환
- 문자열 "STRING"은 모든 문자가 대문자이기 때문에 False 반환

```
print("String".istitle())
print("STRING".istitle())
```

```
True
False
```

islower()

- 문자열 "string"은 모든 문자가 소문자여서 True 반환
- 문자열 "String"은 첫 문자가 대문자여서 False 반환

```
print("string".islower())
print("String".islower())
```

```
True
False
```

isupper() 함수

- 문자열 "STRING"은 모든 문자가 대문자여서 True 반환

- 문자열 "String"은 첫 문자만 대문자여서 False 반환

```
print("STRING".isupper())
print("String".isupper())
```

```
True
False
```

join()

- 문자열 "String"에 대해서 `join()`으로 공백 문자 ' '를 문자 사이마다 추가한 문자열 "S t r i n g"을 반환
- 문자열 "String"에 대해서 `join()` 으로 공백 문자 '_'를 문자 사이마다 추가한 문자열 "S_t_r_i_n_g"을 반환
- 문자열 "String"에 대해서 `join()` 으로 공백 문자 '|'를 문자 사이마다 추가한 문자열 "S|t|r|i|n|g"을 반환

```
s = "String"
print(' '.join(s))
print('_'.join(s))
print('|'.join(s))
```

```
S t r i n g
S_t_r_i_n_g
S|t|r|i|n|g
```

center() / ljust() / rjust()

- 문자열 "String"에 대해서 `center()`를 사용하여 너비 10에 해당하는 문자열 길이에 가운데 정렬한 문자열 ' String '로 변환
- 문자열 "String"에 대해서 `ljust()`를 사용하여 너비 10에 해당하는 문자열 길이에 왼쪽 정렬한 문자열 'String '로 변환
- 문자열 "String"에 대해서 `rjust()`를 사용하여 너비 10에 해당하는 문자열 길이에 오른쪽 정렬한 문자열 ' String'로 변환

```
print("'" + "String".center(10) + "'")
print("'" + "String".ljust(10) + "'")
print("'" + "String".rjust(10) + "'")
```

```
'  String  '
'String    '
'    String'
```

lower() / upper() / title() / swapcase()

- 문자열 "String"을 `lower()`를 이용하여 모든 문자가 소문자로 변환된 "string" 반환
- 문자열 "String"을 `upper()`를 이용하여 모든 문자가 대문자로 변환된 "STRING" 반환

- 문자열 "string"을 `title()`를 이용하여 첫 문자가 대문자로 변환된 "String" 반환
- 문자열 "String"을 `swapcase()`를 이용하여 대문자는 소문자로 변환되고 소문자는 대문자로 변환된 "sTRING" 반환

```
print("String".lower())
print("String".upper())
print("string".title())
print("String".swapcase())
```

```
string
STRING
String
sTRING
```

strip() / lstrip() / rstrip()

- 양쪽 공백이 포함된 문자열 " String "을 `strip()`를 통해 공백 제거
- 양쪽 공백이 포함된 문자열 " String "을 `lstrip()`를 통해 왼쪽 공백 제거
- 양쪽 공백이 포함된 문자열 " String "을 `rstrip()`를 통해 오른쪽 공백 제거

```
print("  String  ".strip())
print("  String  ".lstrip())
print("  String  ".rstrip())
```

```
String
String
  String
```

partition() / rpartition()

- 문자열 "String"을 `partition()`를 이용하여 문자 't' 기준으로 분할
- 문자열 "String String"을 `rpartition()`를 이용하여 마지막 문자 'S'를 기준으로 분할

```
print("String".partition('t'))
print("String String".rpartition('S'))
```

```
('S', 't', 'ring')
('String ', 'S', 'tring')
```

replace()

- 문자열 "String"을 `replace()`를 이용해 문자열 "Str"을 문자 'R'로 교체

```
print("String".replace("Str", 'R'))
```

```
Ring
```

split() / rsplit() / splitlines()

- 문자열 "1 2 3"을 `split()`의 기본 구분자를 이용하여 '1', '2', '3' 으로 구분
- 문자열 "1_2_3"을 `split()`으로 구분자 '_'를 이용하여 '1', '2', '3' 으로 구분
- 문자열 "1 2 3"을 `rsplit()`으로 구분자 '_'를 이용하여 오른쪽부터 1번만 구분하여 '1_2', '3' 으로 구분
- 문자열 "123\n123\n123\n"을 `splitlines()` 함수를 이용하여 라인 단위로 구분하여 '123, '123, '123 반환

```
print("1 2 3".split())
print("1_2_3".split('_'))
print("1_2_3".rsplit('_', 1))
print("123₩n123₩n123₩n".splitlines())
```

```
['1', '2', '3']
['1', '2', '3']
['1_2', '3']
['123', '123', '123']
```

startswith() / endswith()

- 문자열 "String"에서 시작 문자가 'S'인지 `startswith()`를 통해 확인
- 문자열 "String"에서 마지막 문자가 'g'인지 `endswith()`를 통해 확인

```
print("String".startswith('S'))
print("String".endswith('g'))
```

```
True
True
```

zfill()

- 문자열 "123"에서 너비 8만큼으로 늘리고 비어있는 부분에 '0'이 채워진 문자열 "00000123" 반환

```
print("123".zfill(8))
```

```
00000123
```

문자열 서식(String Format)

문자열 포맷팅(String Formatting)

- 문자열 내에서 서식에 맞추어 특정 값을 삽입 또는 변경
- 기호 % 뒤에 있는 값이 문자열 내의 서식에 순서대로 매핑

```
print("나는 사과가 %d개 있다." % 2)
print("나는 사과가 %s개 있다." % "두")
print("나는 %s가 %d개 있다." % ("사과", 2))
```

```
나는 사과가 2개 있다.
나는 사과가 두개 있다.
나는 사과가 2개 있다.
```

문자열 포맷 코드(String Format Code)

- 문자열 포멧팅에서 사용할 수 있는 다양한 포맷 코드

코드	설명
%s	문자열(string)
%c	문자(character)
%d	정수(integer)
%f	부동소수(floating-point)
%o	8진수(octal)
%x	16진수(hexadecimal)
%%	문자 '%'

정렬, 공백, 소수점 포맷

- 포맷 문자 앞에 숫자는 길이를 의미
- -는 왼쪽 정렬을 의미
- 소수점 '.' 뒤에 숫자는 소숫점 이하 개수를 의미

```
print("%8s" % "Hello")
print("%-8sPython" % "Hello")
print("%0.2f" % 3.1415926535)
print("%8.2f" % 3.1415926535)
print("%-8.2f" % 3.1415926535)
```

```
   Hello
Hello   Python
3.14
    3.14
3.14
```

문자열 format 메소드

- 문자열 `format()`의 인덱스를 이용한 고급 포맷팅
- 인덱스 번호에 따라 `format()` 함수의 각 값들이 매핑

```
print("나는 사과가 {0}개 있다.".format(2))
print("나는 사과가 {0}개 있다.".format("두"))
print("나는 {0}가 {1}개 있다".format("사과", 2))
```

```
나는 사과가 2개 있다.
나는 사과가 두개 있다.
나는 사과가 2개 있다
```

- 문자열 `format()` 메소드의 인덱스와 변수 이름을 이용한 포맷팅
- 변수 이름 s와 n을 지정하여 "사과"와 숫자 2를 매핑

```
print("나는 사과가 {n}개 있다.".format(n=2))
print("나는 {s}가 {n}개 있다".format(s="사과", n=2))
```

```
나는 사과가 2개 있다.
나는 사과가 2개 있다
```

- 문자열 `format()` 함수의 정렬, 공백, 소수점

포맷	설명
:8	길이 8
:<	왼쪽 정렬
:>	오른쪽 정렬
:^	가운데 정렬
:~	'~'로 공백 채우기 (정렬 문자 <, >, ^ 앞에 넣은 문자로 공백 채우기)
:0.2f	소수점을 2자리까지 표현
:8.4f	길이 8, 소수점 2자리
{{	'{' 문자 표현
}}	'}' 문자 표현

```
print("{0:8}".format("Hello"))
print("{0:<8}".format("Hello"))
print("{0:>8}".format("Hello"))
print("{0:^8}".format("Hello"))
print("{0:~^8}".format("Hello"))
print("{0:-^8}".format("Hello"))
print("{0:0.2f}".format(3.1415926535))
print("{0:8.4f}".format(3.1415926535))
print("{{중괄호}}".format())
```

```
Hello
Hello
   Hello
 Hello
~Hello~~
-Hello--
3.14
  3.1416
{중괄호}
```

f 문자열 포맷팅

- 문자열 포맷팅을 위해 새롭게 나온 기능으로 문자열 앞에 f를 붙여서 구분
- 변수 s와 n에 대해서 문자열에 포함된 이름으로 매핑하여 사용 가능

```
s = "사과"
n = 2
print(f"나는 사과가 {n}개 있다.")
print(f"나는 {s}가 {n}개 있다.")
```

```
나는 사과가 2개 있다.
나는 사과가 2개 있다.
```

- 기존의 문자열 포맷을 문자열 다음 기호 : 뒤에 표현하여 그대로 사용 가능

```
print(f'{"Hello":8}')
print(f'{"Hello":<8}')
print(f'{"Hello":>8}')
print(f'{"Hello":^8}')
print(f'{"Hello":~^8}')
print(f'{"Hello":-^8}')
print(f'{3.1415926535:0.2f}')
print(f'{3.1415926535:8.4f}')
print(f'{{중괄호}}')
```

```
Hello
Hello
   Hello
 Hello
~Hello~~
-Hello--
3.14
  3.1416
{중괄호}
```