# Python (3)

## 자료형 - 문자열

# 02-2 문자열 자료형

문자열(string)이란 문자, 단어 등으로 구성된 문자들의 집합을 말한다. 예를 들면 다음과 같다.

```
"Life is too short, You need Python"
"a"
"123"
```

위의 모든 예문이 큰따옴표(")로 둘러싸여 있다. "123"은 숫자가 아니라 따옴표로 둘러싸여 있어 문자열이라고 한다.

## ○ 문자열은 어떻게 만들고 사용할까?

앞에서는 문자열을 만들 때 큰따옴표만 사용했지만, 이 밖에도 문자열을 만드는 방법은 3가지가 더 있다.
즉, 파이썬에서 문자열을 만드는 방법은 총 4가지이다.

### 1. 큰따옴표로 양쪽 둘러싸기

```
"Hello World"
```

**2. 작은따옴표로 양쪽 둘러싸기**

```
'Python is fun'
```

**3. 큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기**

```
"""Life is too short, You need python"""
```

**4. 작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기**

```
'''Life is too short, You need python'''
```

## ○ 문자열 안에 작은따옴표나 큰따옴표를 포함시키고 싶을 때

문자열을 만들어 주는 것은 작은따옴표(')와 큰따옴표(")이다.
그런데 문자열 안에도 작은따옴표와 큰따옴표가 들어 있어야 할 경우가 있다.
이때는 좀 더 특별한 방법이 필요하다. 예제를 하나씩 살펴보면서 원리를 익혀 보자.

### 1. 문자열에 작은따옴표 포함하기

```
Python's favorite food is perl
```

위와 같은 문자열을 **food** 변수에 저장하고 싶다고 가정해 보자. 문자열 중 **Python's**에 작은따옴표(')가 포함되어 있다.

이 경우에는 문자열을 큰따옴표로 둘러싸야 한다.

```
food = "Python's favorite food is perl"
print(food)
--------------------
Python's favorite food is perl
```

이번에는 다음과 같이 문자열을 큰따옴표가 아닌 작은따옴표로 둘러싼 후 다시 실행해 보자.

'Python'이 문자열로 인식되어 구문 오류(SyntaxError)가 발생할 것이다.

## 2. 문자열에 큰따옴표 포함하기

아래와 같이 큰따옴표가 포함된 문자열이라면 문자열을 작은따옴표로 둘러싸면 된다.

```
say = '"Python is very easy." he says.'
print(say)
--------------------
"Python is very easy." he says.
```

## 3. 역슬래시를 사용해서 작은따옴표와 큰따옴표를 문자열에 포함하기

```
food = 'Python\'s favorite food is perl'
say = "\"Python is very easy.\" he says."
print(food)
print(say)
--------------------
Python's favorite food is perl
"Python is very easy." he says.
```

작은따옴표나 큰따옴표를 문자열에 포함시키는 또 다른 방법은 역슬래시(\)를 사용하는 것이다.
즉, 역슬래시를 작은따옴표나 큰따옴표 앞에 삽입하면 역슬래시 뒤의 작은따옴표나 큰따옴표는 문자열을 둘러싸는 기호의 의미가 아니라 따옴표 그 자체를 뜻하게 된다.

어떤 방법을 사용해서 문자열 안에 작은따옴표(')와 큰따옴표(")를 포함시킬 것인지는 각자가 선택하여 사용하면 된다.

## ○ 여러 줄인 문자열을 변수에 대입하고 싶을 때

문자열이 항상 한 줄짜리만 있는 것은 아니다. 다음과 같은 여러 줄의 문자열을 변수에 대입하려면 어떻게 해야 할까?

```
Life is too short
You need python
```

### 1. 줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기

```
multiline = "Life is too short\nYou need python"
print(multiline)
```

```
--------------------
Life is too short
You need python
```

위 예처럼 줄바꿈 문자인 \n을 삽입하는 방법이 있지만, 읽기가 불편하고 줄이 길어지는 단점이 있다.

## 2. 연속된 작은따옴표 3개 또는 큰따옴표 3개 사용하기

1번 방법의 단점을 극복하기 위해 파이썬에서는 다음과 같이 작은따옴표 3개(''') 또는 큰따옴표 3개(""")를 사용한다.

아래는 작은따옴표 3개를 사용한 경우이다.

```
multiline = '''
Life is too short
You need python
'''
print(multiline)
--------------------

Life is too short
You need python
```

아래는 큰따옴표 3개를 사용한 경우이다

```
multiline = """
Life is too short
You need python
"""
print(multiline)
--------------------

Life is too short
You need python
```

두 경우 모두 결과는 동일하다.
위 예에서도 확인할 수 있듯이 문자열이 여러 줄인 경우, 이스케이프 코드를 쓰는 것보다 따옴표 3개를 사용하는 것이 훨씬 깔끔하다.

여기서 한가지 주의할 것은, 위 두가지 예를 자세히 살펴보면 앞, 뒤로 새로운 줄이 하나씩 생긴 것을 볼 수 있다.

## ◆ 이스케이프 코드란?

문자열 예제에서 여러 줄의 문장을 처리할 때 역슬래시 문자와 소문자 n을 조합한 \n 이스케이프 코드를 사용했다.
이스케이프(escape) 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 '문자 조합'을 말한다.
주로 출력물을 보기 좋게 정렬하는 용도로 사용한다. 몇 가지 이스케이프 코드를 정리하면 다음과 같다.

| 코드 | 설명 |
|---|---|
| \n | 문자열 안에서 줄을 바꿀 때 사용 |
| \t | 문자열 사이에 탭 간격을 줄 때 사용 |
| \\ | \를 그대로 표현할 때 사용 |
| \' | 작은따옴표(')를 그대로 표현할 때 사용 |
| \" | 큰따옴표(")를 그대로 표현할 때 사용 |
| \r | 캐리지 리턴(줄 바꿈 문자, 커서를 현재 줄의 가장 앞으로 이동) |
| \f | 폼 피드(줄 바꿈 문자, 커서를 현재 줄의 다음 줄로 이동) |
| \a | 벨 소리(출력할 때 PC 스피커에서 '삑' 소리가 난다) |
| \b | 백 스페이스 |
| \000 | Null 문자 |

이 중에서 활용 빈도가 높은 것은 \n, \t, \\, \', \"이다. 나머지는 프로그램에서 잘 사용하지 않는다.

# ○ 문자열 연산하기

## ◆ 문자열 더해서 연결하기

```
head = "Python"
tail = " is fun!"
print(head + tail)
--------------------
Python is fun!
```

위 소스 코드에서 세 번째 줄을 살펴보자.
"Python"이라는 문자열 값을 가진 head 변수와 " is fun!"이라는 문자열을 가진 tail 변수를 더한 것이다.

## ◆ 문자열 곱하기

```
a = "python"
print(a * 2)
--------------------
pythonpython
```

위 소스 코드에서 *의 의미는 우리가 일반적으로 사용하는 숫자 곱하기의 의미와는 다르다.
위 소스 코드에서 a * 2라는 문장은 a를 2번 반복하라는 뜻이다.

### ◆ 문자열 곱하기를 응용하기

```
print("=" * 50)
print("My Program")
print("=" * 50)
--------------------
==================================================
My Program
==================================================
```

## ○ 문자열 길이 구하기

문자열의 길이는 다음과 같이 len 함수를 사용하면 구할 수 있다.
len 함수는 print 함수처럼 파이썬의 기본 내장 함수로, 별다른 설정 없이 바로 사용할 수 있다.

문자열의 길이에는 공백 문자도 포함된다.

```
a = "Life is too short"
print(len(a))
--------------------
17
```

## ○ 문자열 인덱싱과 슬라이싱

인덱싱(indexing)이란 무엇인가를 '가리킨다', 슬라이싱(slicing)은 무엇인가를 '잘라 낸다'라는 의미이다.
이런 의미를 생각하면서 다음 내용을 살펴보자.

### ◆ 문자열 인덱싱

```
a = "Life is too short, You need Python"
```

위 코드에서 변수 a에 저장한 문자열의 각 문자마다 번호를 매겨 보면 다음과 같다.

| L | i | f | e |  | i | s |  | t | o | o |  | s | h | o | r | t | , |  | Y | o | u |  | n | e | e | d |  | P | y | t | h | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

위 문자열에서 L은 첫 번째 자리를 뜻하는 숫자 0, i는 1 이런 식으로 계속 번호를 붙인 것이다. 즉, 중간에 있는 short의 s는 12가 된다.

이제 다음 예를 실행해 보자.

```
a = "Life is too short, You need Python"
print(a[3])
--------------------
e
```

a[3]이 뜻하는 것은 a라는 문자열의 네 번째 문자 e를 말한다.
"파이썬은 0부터 숫자를 센다."

따라서 파이썬은 위 문자열을 다음과 같이 바라보고 있다.

a[0]:'L', a[1]:'i', a[2]:'f', a[3]:'e', a[4]:' ', ...

위 예에서 볼 수 있듯이 a[번호]는 문자열 안의 특정한 값을 뽑아 내는 역할을 한다. 이러한 작업을 '인덱싱'이라고 한다.

## ◆ 문자열 인덱싱 활용하기

인덱싱의 예를 몇 가지 더 살펴보자.

```
a = "Life is too short, You need Python"
print(a[0])
print(a[12])
print(a[-1])
--------------------
L
s
n
```

앞의 a[0]과 a[12]는 쉽게 이해할 수 있는데, 마지막의 a[-1]이 뜻하는 것은 뭘까?
이것은 문자열을 뒤에서부터 읽기 위해 -(빼기) 기호를 붙인 것이다.
즉, a[-1]은 뒤에서부터 세어 첫 번째가 되는 문자를 뜻한다. 뒤에서부터 첫 번째 문자는 가장 마지막 문자 'n'이다.

## ◆ 문자열 슬라이싱

그렇다면 "Life is too short, You need Python" 문자열에서 단순히 한 문자만을 뽑아 내는 것이 아니라 ‘Life’ 또는 ‘You’와 같은 단어를 뽑아 내는 방법에 대해 얄아보자

문자열 a가 아래와 같다고 하자.

```
a = "Life is too short, You need Python"
```

```
b = a[0] + a[1] + a[2] + a[3]
print(b)
--------------------
Life
```

위와 같이 단순하게 접근할 수도 있지만 파이썬에서는 더 좋은 방법을 제공한다.
바로 슬라이싱(slicing) 기법이다. 위 예는 슬라이싱 기법으로 다음과 같이 간단하게 처리할 수 있다.

문자열 슬라이싱을 적용하는 문법은 다음과 같다

```
문자열[시작_인덱스:끝_인덱스]
```

아래의 예를 살펴보자.

```
print(a[0:4])
```

```
--------------------
Life
```

a[0:4]는 a 문자열, 즉 "Life is too short, You need Python" 문자열에서 시작 인덱스 0부터 끝 인덱스 4까지의 문자를 뽑아 낸다는 뜻이다.

그런데 위 결과에서 보듯, 시작 인덱스는 포함되지만, 끝 인덱스는 포함되지 않는 것을 볼 수 있다.
다시 말하면, (끝_인덱스 - 1) 위치의 문자까지만 잘라낸다는 것을 알 수 있다. 이것은 매우 중요하므로 꼭 기억해 두어야 한다.

## ◆ 문자열을 슬라이싱하는 방법

슬라이싱의 예를 좀 더 살펴보자.

```
print(a[0:5])
--------------------
Life                  # 'Life '
```

위 예는 a[0] + a[1] + a[2] + a[3] + a[4]와 동일하다.
a[4]는 공백 문자이기 때문에 'Life'가 아닌 'Life '가 출력된다.
공백 문자 역시 L, i, f, e 와 같은 문자와 동일하게 취급되는 것을 잊지 말자. 'Life'와 'Life '는 완전히 다른 문자열이다.

또한, 슬라이싱할 때 항상 시작 번호가 0일 필요는 없다.

```
print(a[0:2])
print(a[5:7])
print(a[12:17])
--------------------
Li
is
short
```

만약 아래와 같이 a[시작_인덱스:끝_인덱스]에서 끝_인덱스를 생략하면 시작 번호부터 그 문자열의 끝까지 추출한다.

```
print(a[19:])
--------------------
You need Python
```

a[시작_인덱스:끝_인덱스]에서 시작_인덱스를 생략하면 문자열의 처음부터 끝_인덱스까지 추출한다.

```
print(a[:17])
--------------------
```

```
Life is too short
```

a[시작_인덱스:끝_인덱스]에서 시작_인덱스와 끝_인덱스를 모두 생략하면 문자열의 처음부터 끝까지 추출한다.

```
print(a[:])
--------------------
Life is too short, You need Python
```

또한 아래와 같이 슬라이싱에서도 인덱싱과 마찬가지로 음수(-)를 사용할 수 있다.

```
print(a[19:-7])
--------------------
You need
```

a[19:-7]은 a[19]에서 a[-8]까지를 의미한다. 이때에도 a[-7]은 포함하지 않는다.

## ◆ 슬라이싱으로 문자열 나누기

다음은 자주 사용하는 슬라이싱 기법 중 하나이다.

```
a = "20230331Rainy"
date = a[:8]
weather = a[8:]

print(date)
print(weather)
--------------------
20230331
Rainy
```

위 예는 문자열 a를 두 부분으로 나누는 기법이다.
숫자 8을 기준으로 문자열 a를 양쪽으로 한 번씩 슬라이싱했다.
a[:8]은 a[8]을 포함하지 않고 a[8:]은 a[8]을 포함하기 때문에 8을 기준으로 해서 두 부분으로 나눌 수 있는 것이다.
위 예에서는 "20230331Rainy" 문자열을 날짜를 나타내는 부분인 '20230331'과 날씨를 나타내는 부분인 'Rainy'로 나누는 방법을 보여 준다.

"20230331Rainy"를 연도인 2023, 월과 일을 나타내는 0331, 날씨를 나타내는 Rainy까지 세 부분으로 나누는 방법은 다음과 같다.

```
a = "20230331Rainy"
year = a[:4]
day = a[4:8]
weather = a[8:]
```

```
print(year)
print(day)
print(weather)
--------------------
2023
0331
Rainy
```

지금까지 인덱싱과 슬라이싱에 대해서 살펴보았다. 인덱싱과 슬라이싱은 프로그래밍할 때 자주 사용하는 기법이므로 꼭 반복해서 연습해 두자.

## ◆ Pithon 문자열을 Python으로 바꾸려면?

Pithon 문자열을 Python으로 바꾸려면 어떻게 해야 할까? 제일 먼저 떠오르는 생각은 다음과 같을 것이다.

```
a = "Pithon"
print(a[1])

a[1] = 'y'
print(a)
```

즉, a 변수에 "Pithon" 문자열을 대입하고 a[1]의 값이 i이므로 a[1]을 y로 바꾸어 준다는 생각이다.
하지만 결과는 오류가 발생한다.

문자열의 개별 문자는 바꿀 수 없기 때문이다. (그래서 문자열을 '변경 불가능한(immutable) 자료형'이라고도 부른다).

하지만 앞에서 배운 슬라이싱 기법을 사용하면 Pithon 문자열을 사용해 Python 문자열을 만들 수 있다. 다음 예를 살펴보자.

```
a = "Pithon"
print(a[:1])
print(a[2:])

print(a[:1] + 'y' + a[2:])
--------------------
P
thon
Python
```

위 예제와 같이 슬라이싱을 사용하면 "Pithon" 문자열을 'P' 부분과 'thon' 부분으로 나눌 수 있고, 그 사이에 'y' 문자를 추가하면 'Python'이라는 새로운 문자열을 만들 수 있다.

## ❍ 문자열 포매팅이란?

문자열에서 또 하나 알아야 할 것으로는 '문자열 포매팅(string formatting)'이 있다.
문자열 포매팅을 공부하기 전에 다음과 같은 문자열을 출력하는 프로그램을 작성했다고 가정해 보자.

```
"현재 온도는 18도입니다."
```

시간이 지나서 20도가 되면 다음 문장을 출력한다.

```
"현재 온도는 20도입니다"
```

두 문자열은 모두 같은데 20이라는 숫자와 18이라는 숫자만 다르다.

이렇게 문자열 안의 특정한 값을 바꿔야 할 경우가 있을 때 이것을 가능하게 해 주는 것이 바로 문자열 포매팅이다.

쉽게 말해 문자열 포매팅이란 문자열 안에 어떤 값을 삽입하는 방법이다.

다음 예를 직접 실행해 보면서 그 사용법을 알아보자.

# ○ 문자열 포매팅 따라 하기

## 1. 숫자 바로 대입

```
str = "I eat %d apples." % 3
print(str)
--------------------
I eat 3 apples.
```

결괏값을 보면 알겠지만, 위 예제는 문자열 안에 정수 3을 삽입하는 방법을 보여 준다.
문자열 안의 숫자를 넣고 싶은 자리에 %d 문자를 넣어 주고 삽입할 숫자 3은 가장 뒤에 있는 % 문자 다음에 써 넣었다.
여기에서 %d는 '문자열 포맷 코드'라고 부른다.

## 2. 문자열 바로 대입

문자열 안에 꼭 숫자만 넣으라는 법은 없다. 이번에는 숫자 대신 문자열을 넣어 보자.

```
str = "I eat %s apples." % "five"
print(str)
--------------------
I eat five apples.
```

문자열 안에 또 다른 문자열을 삽입하기 위해 앞에서 사용한 문자열 포맷 코드 %d가 아닌 %s를 사용했다.
문자열을 대입할 때는 반드시 큰따옴표나 작은따옴표를 사용해야 한다.

## 3. 숫자 값을 나타내는 변수로 대입

```
number = 3
print("I eat %d apples." % number)
--------------------
I eat 3 apples.
```

1번처럼 숫자를 바로 대입하든, 위 예제처럼 숫자 값을 나타내는 변수를 대입하든 결과는 같다.

## 4. 2개 이상의 값 넣기

그렇다면 문자열 안에 1개가 아닌 여러 개의 값을 넣고 싶을 때는 어떻게 해야 할까?

```
number = 10
day = "three"
str = "I ate %d apples. so I was sick for %s days." % (number, day)
print(str)
```

```
--------------------
I ate 10 apples. so I was sick for three days.
```

2개 이상의 값을 넣으려면 마지막 % 다음 괄호 안에 쉼표(,)로 구분하여 각각의 값을 넣어 주면 된다.

## ◆ 문자열 포맷 코드

위 문자열 포매팅 예제에서는 대입해 넣는 자료형으로 정수와 문자열을 사용했지만, 이 밖에도 다양한 것을 대입할 수 있다.
문자열 포맷 코드의 종류는 다음과 같다.

| 코드 | 설명 |
|---|---|
| %s | 문자열(String) |
| %c | 문자 1개(character) |
| %d | 정수(Integer) |
| %f | 부동소수(floating-point) |
| %o | 8진수 |
| %x | 16진수 |
| %% | Literal % (문자 % 자체) |

여기에서 재미있는 것은 %s 포맷 코드인데, 이 코드에는 어떤 형태의 값이든 변환해 넣을 수 있다.
무슨 말인지 예를 통해 확인해 보자.

```
print("I have %s apples" % 3)
print("rate is %s" % 3.234)
--------------------
I have 3 apples
rate is 3.234
```

3을 문자열 안에 삽입하려면 %d를 사용해야 하고 3.234를 삽입하려면 %f를 사용해야 한다.
하지만 %s를 사용하면 %s는 자동으로 % 뒤에 있는 3이나 3.234와 같은 값을 문자열로 바꾸어 대입하기 때문에 이런 것을 생각하지 않아도 된다.

## ◆ 포매팅 연산자 %d와 %를 같이 쓸 때는 %%를 쓴다

아래의 코드를 보자.

```
print("Error is %d%." % 98)
```

결괏값으로 당연히 "Error is 98%."가 출력될 것이라고 예상하겠지만, 파이썬은 '형식이 불완전하다'라는 오류 메시지를 보여 준다.

```
ValueError: incomplete format
```

그 이유는 '문자열 포맷 코드인 %d와 %가 같은 문자열 안에 존재하는 경우, %를 나타내려면 반드시 %%를 써야 한다'라는 법칙이 있기 때문이다.
하지만 문자열 안에 %d와 같은 포매팅 연산자가 없으면 %는 홀로 쓰여도 상관없다.
따라서 위 예를 제대로 실행하려면 다음과 같이 작성해야 한다.

```
print("Error is %d%%." % 98)
--------------------
Error is 98%.
```

## ○ 포맷 코드와 숫자 함께 사용하기

앞에서 살펴보았듯이 %d, %s 등과 같은 포맷 코드는 문자열 안에 어떤 값을 삽입할 때 사용한다.
하지만 포맷 코드를 숫자와 함께 사용하면 더 유용하다. 다음 예를 따라해 보자.

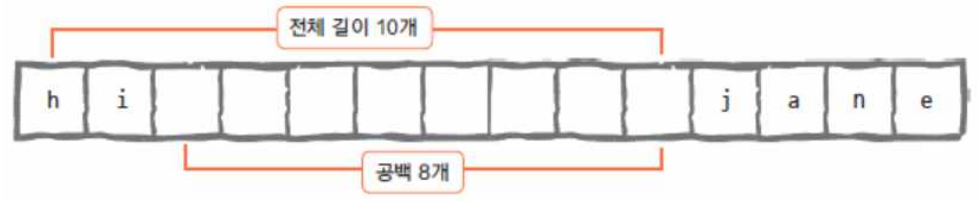
### 1. 정렬과 공백

```
print("%10s" % "hi")
--------------------
        hi
```

**%10s**는 전체 길이가 **10**개인 문자열 공간에서 대입되는 값을 오른쪽으로 정렬하고 그 앞의 나머지는 공백으로 남겨 두라는 의미이다.


그렇다면 반대쪽인 왼쪽 정렬은 **%-10s**가 될 것이다.

```
print("%-10sjane" % 'hi')
--------------------
hi       jane.
```



hi를 왼쪽으로 정렬하고 나머지는 공백으로 채웠다는 것을 알 수 있다.

## 2. 소수점 표현하기

```
print("%0.4f" % 3.42134234)
--------------------
3.4213
```

3.42134234를 소수점 네 번째 자리까지만 나타내고 싶은 경우에는 위와 같이 작성한다.
%0.4f에서 '.'는 소수점 포인트, 그 뒤의 숫자 4는 소수점 뒤에 나올 숫자의 개수를 말한다.
소수점 포인트 앞의 숫자는 문자열의 전체 길이를 의미하는데, %0.4f에서 사용한 숫자 0은 길이에 상관하지 않겠다는 의미이다.

%0.4f는 0을 생략하여 %.4f처럼 사용하기도 한다.

다음 예를 살펴보자.

```
print("%10.4f" % 3.42134234)
--------------------
    3.4213
```

위는 숫자 3.42134234를 소수점 네 번째 자리까지만 표시하고 전체 길이가 10개인 문자열 공간에서 오른쪽으로 정렬하는 예를 보여

준다.


## ○ format 함수를 사용한 포매팅

문자열의 format 함수를 사용하면 좀 더 발전된 스타일로 문자열 포맷을 지정할 수 있다.
앞에서 살펴본 문자열 포매팅 예제를 format 함수를 사용해서 바꾸면 다음과 같다.

### ◆ 숫자 바로 대입하기

```
print("I eat {0} apples".format(3))
--------------------
I eat 3 apples
```

"I eat {0} apples" 문자열 중 {0} 부분이 숫자 3으로 바뀌었다.

## ◆ 문자열 바로 대입하기

```
print("I eat {0} apples".format("five"))
--------------------
I eat five apples
```

문자열의 **{0}** 항목이 **‘five’**라는 문자열로 바뀌었다.

## ◆ 숫자 값을 가진 변수로 대입하기

```
number = 3
print("I eat {0} apples".format(number))
--------------------
I eat 3 apples
```

문자열의 **{0}** 항목이 **number** 변수의 값인 **3**으로 바뀌었다.

## ◆ 2개 이상의 값 넣기

```
number = 10
day = "three"
print("I ate {0} apples. so I was sick for {1} days.".format(number, day))
--------------------
I ate 10 apples. so I was sick for three days.
```

2개 이상의 값을 넣을 경우, 문자열의 {0}, {1}과 같은 인덱스 항목이 format 함수의 입력값으로 순서에 맞게 바뀐다.
위 예에서 {0}은 format 함수의 첫 번째 입력값인 number, {1}은 format 함수의 두 번째 입력값인 day로 바뀐다.

## ◆ 이름으로 넣기

```
print("I ate {number} apples. so I was sick for {day} days.".format(number=10, day=3))
--------------------
I ate 10 apples. so I was sick for 3 days.
```

{0}, {1}과 같은 인덱스 항목 대신 더 편리한 {name} 형태를 사용하는 방법도 있다.
{name} 형태를 사용할 경우, format 함수에는 반드시 name=value와 같은 형태의 입력값이 있어야 한다.
위 예는 문자열의 {number}, {day}가 format 함수의 입력값인 number=10, day=3 값으로 각각 바뀌는 것을 보여 주고 있다.

## ◆ 인덱스와 이름을 혼용해서 넣기

```
print("I ate {0} apples. so I was sick for {day} days.".format(10, day=3))
--------------------
I ate 10 apples. so I was sick for 3 days.
```

인덱스 항목과 **name=value** 형태를 혼용하는 것도 가능하다.

## ◆ 왼쪽 정렬

```
print("{0:<10}".format("hi"))
--------------------
hi
```

**:<10** 표현식을 사용하면 치환되는 문자열을 왼쪽으로 정렬하고 문자열의 총 자릿수를 **10**으로 맞출 수 있다.

## ◆ 오른쪽 정렬

```
print("{0:>10}".format("hi"))
--------------------
       hi
```

오른쪽 정렬은 :< 대신 :>을 사용하면 된다. 화살표의 방향을 생각하면 어느 쪽으로 정렬되는지 바로 알 수 있을 것이다.

## ◆ 가운데 정렬

```
print("{0:^10}".format("hi"))
--------------------
    hi
```

:^를 사용하면 가운데 정렬도 가능하다.

## ◆ 공백 채우기

```
print("{0:=^10}".format("hi"))
print("{0:!<10}".format("hi"))
--------------------
====hi====
hi!!!!!!!!
```

정렬할 때 공백 문자 대신 지정한 문자 값으로 채워 넣을 수도 있다.

채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 한다.
위 예에서 첫 번째 예제는 가운데(^)로 정렬하고 빈 공간을 =로 채웠고, 두 번째 예제는 왼쪽(<)으로 정렬하고 빈 공간을 !로 채웠다.

## ◆ 소수점 표현하기

```
y = 3.42134234
print("{0:0.4f}".format(y))
--------------------
3.4213
```

위는 format 함수를 사용해 소수점을 4자리까지만 표현하는 방법을 보여 준다.
앞에서 살펴보았던 표현식 0.4f를 그대로 사용한 것을 알 수 있다.

```
y = 3.42134234
print("{0:10.4f}".format(y))
--------------------
    3.4213
```

위와 같이 자릿수를 10으로 맞출 수도 있다. 이 또한 앞에서 살펴본 10.4f의 표현식을 그대로 사용한 것을 알 수 있다.

### ◆ { 또는 } 문자 표현하기

```
print("{{ and }}".format())
--------------------
{ and }
```

format 함수를 사용해 문자열을 포매팅할 경우, {}와 같은 중괄호 문자를 포매팅 문자가 아닌 문자 그대로 사용하고 싶은 경우에는 위 예의 {{}}처럼 2개를 연속해서 사용하면 된다.

## ○ f 문자열 포매팅  손찬혁

파이썬 3.6 버전부터는 f 문자열 포매팅 기능을 사용할 수 있다. 파이썬 3.6 미만 버전에서는 사용할 수 없는 기능이므로 주의해야 한다.

다음과 같이 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능을 사용할 수 있다.

```
name = '홍길동'
age = 18
print(f'나의 이름은 {name}입니다. 나이는 {age}입니다.')
```

```
--------------------
나의 이름은 홍길동입니다. 나이는 18입니다.
```

f 문자열 포매팅은 위와 같이 name, age와 같은 변숫값을 생성한 후에 그 값을 참조할 수 있다.
또한 f 문자열 포매팅은 표현식을 지원하기 때문에 다음과 같은 것도 가능하다.

여기서 표현식이란, 중괄호 안의 변수를 계산식과 함께 사용하는 것을 말한다.

```
age = 20
print(f'나는 내년이면 {age + 1}살이 된다.')
--------------------
나는 내년이면 21살이 된다.
```

딕셔너리는 f 문자열 포매팅에서 다음과 같이 사용할 수 있다.

```
d = {'name':'홍길동', 'age':20}
print(f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.')
--------------------
나의 이름은 홍길동입니다. 나이는 20입니다.
```

딕셔너리는 Key와 Value라는 것을 한 쌍으로 가지는 자료형이다. 향후 자세히 알아볼 것이다.

정렬은 다음과 같이 할 수 있다.

```
print(f'{"hi":<10}')   # 왼쪽 정렬
print(f'{"hi":>10}')   # 오른쪽 정렬
print(f'{"hi":^10}')   # 가운데 정렬
--------------------
hi
        hi
    hi
```

공백 채우기는 다음과 같이 할 수 있다.

```
print(f'{"hi":=^10}')  # 가운데 정렬하고 '=' 문자로 공백 채우기
print(f'{"hi":!<10}')  # 왼쪽 정렬하고 '!' 문자로 공백 채우기
--------------------
====hi====
hi!!!!!!!!
```

소수점은 다음과 같이 표현할 수 있다.

```
y = 3.42134234
print(f'{y:0.4f}')     # 소수점 4자리까지만 표현
print(f'{y:10.4f}')    # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤
--------------------
3.4213
    3.4213
```

f 문자열에서 {}를 문자 그대로 표시하려면 다음과 같이 2개를 동시에 사용해야 한다.

```
print(f'{{ and }}')
--------------------
{ and }
```

지금까지는 문자열을 가지고 할 수 있는 기본적인 것에 대해 알아보았다. 이제부터는 문자열을 좀 더 자유자재로 다루기 위해 공부해야 할 것을 설명한다.

## ○ 문자열 관련 함수들

문자열 자료형은 자체적으로 함수를 가지고 있다.
이들 함수를 다른 말로 '문자열 내장 함수'라고 한다.
이 내장 함수를 사용하려면 문자열 변수 이름 뒤에 '.'를 붙인 후 함수 이름을 써 주면 된다.
이제 문자열의 내장 함수에 대해서 알아보자.

### ◆ 문자 개수 세기 - count

```
a = "hobby"
print(a.count('b'))
--------------------
2
```

count 함수로 문자열 중 문자 b의 개수를 리턴했다.

### ◆ 위치 알려 주기 1 - find

```
a = "Python is the best choice"
print(a.find('b'))
```

```
--------------------
14
```

find 함수로 문자열 중 문자 b가 처음으로 나온 위치를 반환했다.
만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환한다.

파이썬은 숫자를 0부터 세기 때문에 b의 위치는 15가 아닌 14가 된다.

## ◆ 위치 알려 주기 2 – index

```
a = "Life is too short"
print(a.index('t'))
--------------------
8
```

index 함수로 문자열 중 문자 t가 맨 처음으로 나온 위치를 반환했다.

```
print(a.index('k'))
```

만약 찾는 문자나 문자열이 존재하지 않는다면 오류가 발생한다.
앞의 find 함수와 다른 점은 문자열 안에 존재하지 않는 문자를 찾으면 오류가 발생한다는 것이다.

## ◆ 문자열 삽입 – join

```
print(",".join('abcd'))
--------------------
a,b,c,d
```

**join** 함수로 **abcd** 문자열의 각각의 문자 사이에 **‘,’**를 삽입했다.

**join** 함수는 문자열뿐만 아니라 앞으로 배울 리스트나 튜플도 입력으로 사용할 수 있다(리스트와 튜플은 뒤에서 배울 내용이므로 여기에서는 잠시 눈으로만 살펴보자). **join** 함수의 입력으로 리스트를 사용하는 예는 다음과 같다.

```
print(",".join(['a', 'b', 'c', 'd']))
--------------------
a,b,c,d
```

## ◆ 소문자를 대문자로 바꾸기 – upper

```
a = "hi"
```

```
print(a.upper())
--------------------
HI
```

upper 함수는 소문자를 대문자로 바꾸어 준다. 만약 문자열이 이미 대문자라면 아무런 변화도 일어나지 않을 것이다.

## ◆ 대문자를 소문자로 바꾸기 - lower

```
a = "HI"
print(a.lower())
--------------------
hi
```

lower 함수는 대문자를 소문자로 바꾸어 준다.

## ◆ 왼쪽 공백 지우기 - lstrip

```
a = " hi "
print(a.lstrip())
--------------------
hi
```

**lstrip** 함수는 문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백들을 모두 지운다. **lstrip**에서 **l**은 **left**를 의미한다.

## ◆ 오른쪽 공백 지우기 – rstrip

```
a = " hi "
print(a.rstrip())
--------------------
 hi
```

**rstrip** 함수는 문자열 중 가장 오른쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다. **rstrip**에서 **r**은 **right**를 의미한다.

## ◆ 양쪽 공백 지우기 – strip

```
a = " hi "
print(a.strip())
--------------------
hi
```

**strip** 함수는 문자열 양쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다.

## ◆ 문자열 바꾸기 – replace

```
a = "Life is too short"
print(a.replace("Life", "Your leg"))
--------------------
Your leg is too short
```

replace 함수는 replace(바뀔_문자열, 바꿀_문자열)처럼 사용해서 문자열 안의 특정한 값을 다른 값으로 치환해 준다.

## ◆ 문자열 나누기 – split

```
a = "Life is too short"
print(a.split())

b = "a:b:c:d"
print(b.split(':'))
--------------------
['Life', 'is', 'too', 'short']
['a', 'b', 'c', 'd']
```

split 함수는 a.split()처럼 괄호 안에 아무 값도 넣어 주지 않으면 공백([Space]], [Tab], [Enter])을 기준으로 문자열을 나누

어 준다.

만약 **b.split(':')**처럼 괄호 안에 특정 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 준다.

이렇게 나눈 값은 리스트에 하나씩 들어간다.
**['Life', 'is', 'too', 'short']**나 **['a', 'b', 'c', 'd']**가 리스트인데, 향후 알아볼 것이다.

앞에서 소개한 문자열 관련 함수는 문자열 처리에서 사용 빈도가 매우 높고 유용하다.
이 밖에도 몇 가지가 더 있지만 자주 사용하지는 않는다.

## ◆ 착각하기 쉬운 문자열 함수

소문자를 대문자로 바꾸어 주는 다음의 예를 보자.

```
a = "hi"
print(a.upper())
```

이와 같이 실행한 후에 **a** 변수의 값은 **'HI'**로 변했을까? 아니면 **'hi'** 값을 유지할까? 다음과 같이 확인해 보자.

```
print(a)
--------------------
```

```
hi
```

a.upper()를 수행하더라도 a 변수의 값은 변하지 않았다.
왜냐하면 a.upper()를 실행하면 upper 함수는 a 변수의 값 자체를 변경하는 것이 아니라 대문자로 바꾼 값을 반환하기 때문이다.

문자열은 이전에도 잠깐 언급했지만 자체의 값을 변경할 수 없는 immutable 자료형이다.
따라서 a 값을 'HI' 로 바꾸고 싶다면 다음과 같이 대입문을 사용해야 한다.

```
a = a.upper()
print(a)
--------------------
HI
```

upper 뿐만 아니라 lower, join, lstrip, rstrip, strip, replace, split 함수는 모두 이와 같은 규칙이 적용되어 문자열 자체의 값이 변경되는 것이 아니라 변경된 값을 반환한다는 사실에 주의하자.