**02-2 문자열 자료형**

* [문자열이란?](https://wikidocs.net/13#_1)
* [문자열은 어떻게 만들고 사용할까?](https://wikidocs.net/13#_2)
  + [문자열 안에 작은따옴표나 큰따옴표를 포함시키고 싶을 때](https://wikidocs.net/13#_3)
  + [여러 줄인 문자열을 변수에 대입하고 싶을 때](https://wikidocs.net/13#_4)
* [문자열 연산하기](https://wikidocs.net/13#_5)
  + [문자열 더해서 연결하기(Concatenation)](https://wikidocs.net/13#concatenation)
  + [문자열 곱하기](https://wikidocs.net/13#_6)
  + [문자열 곱하기 응용](https://wikidocs.net/13#_7)
  + [문자열 길이 구하기](https://wikidocs.net/13#_8)
* [문자열 인덱싱과 슬라이싱](https://wikidocs.net/13#_9)
  + [문자열 인덱싱이란?](https://wikidocs.net/13#_10)
  + [문자열 인덱싱 활용하기](https://wikidocs.net/13#_11)
  + [문자열 슬라이싱이란?](https://wikidocs.net/13#_12)
  + [문자열을 슬라이싱하는 방법](https://wikidocs.net/13#_13)
  + [슬라이싱으로 문자열 나누기](https://wikidocs.net/13#_14)
* [문자열 포매팅](https://wikidocs.net/13#_15)
  + [문자열 포매팅 따라 하기](https://wikidocs.net/13#_16)
  + [문자열 포맷 코드](https://wikidocs.net/13#_17)
  + [포맷 코드와 숫자 함께 사용하기](https://wikidocs.net/13#_18)
  + [format 함수를 사용한 포매팅](https://wikidocs.net/13#format)
  + [f 문자열 포매팅](https://wikidocs.net/13#f)
* [문자열 관련 함수들](https://wikidocs.net/13#_19)
  + [문자 개수 세기(count)](https://wikidocs.net/13#count)
  + [위치 알려주기1(find)](https://wikidocs.net/13#1find)
  + [위치 알려주기2(index)](https://wikidocs.net/13#2index)
  + [문자열 삽입(join)](https://wikidocs.net/13#join)
  + [소문자를 대문자로 바꾸기(upper)](https://wikidocs.net/13#upper)
  + [대문자를 소문자로 바꾸기(lower)](https://wikidocs.net/13#lower)
  + [왼쪽 공백 지우기(lstrip)](https://wikidocs.net/13#lstrip)
  + [오른쪽 공백 지우기(rstrip)](https://wikidocs.net/13#rstrip)
  + [양쪽 공백 지우기(strip)](https://wikidocs.net/13#strip)
  + [문자열 바꾸기(replace)](https://wikidocs.net/13#replace)
  + [문자열 나누기(split)](https://wikidocs.net/13#split)

**문자열이란?**

문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 의미한다. 예를 들어 다음과 같은 것들이 문자열이다.

"Life is too short, You need Python"

"a"

"123"

위 문자열 예문을 보면 모두 큰따옴표(" ")로 둘러싸여 있다. "123은 숫자인데 왜 문자열이지?"라는 의문이 드는 독자도 있을 것이다. 따옴표로 둘러싸여 있으면 모두 문자열이라고 보면 된다.

**문자열은 어떻게 만들고 사용할까?**

위 예에서는 문자열을 만들 때 큰따옴표(" ")만을 사용했지만 이 외에도 문자열을 만드는 방법은 3가지가 더 있다. 파이썬에서 문자열을 만드는 방법은 총 4가지이다.

**1. 큰따옴표(")로 양쪽 둘러싸기**

"Hello World"

**2. 작은따옴표(')로 양쪽 둘러싸기**

'Python is fun'

**3. 큰따옴표 3개를 연속(""")으로 써서 양쪽 둘러싸기**

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

**4. 작은따옴표 3개를 연속(''')으로 써서 양쪽 둘러싸기**

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

단순함이 자랑인 파이썬이 문자열을 만드는 방법은 왜 4가지나 가지게 되었을까? 그 이유에 대해 알아보자.

**문자열 안에 작은따옴표나 큰따옴표를 포함시키고 싶을 때**

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

**1. 문자열에 작은따옴표 (') 포함시키기**

Python's favorite food is perl

위와 같은 문자열을 food 변수에 저장하고 싶다고 가정하자. 문자열 중 Python's에 작은따옴표(')가 포함되어 있다.

이럴 때는 다음과 같이 문자열을 큰따옴표(")로 둘러싸야 한다. 큰따옴표 안에 들어 있는 작은따옴표는 문자열을 나타내기 위한 기호로 인식되지 않는다.

>>> food = "Python's favorite food is perl"

프롬프트에 food를 입력해서 결과를 확인하자. 변수에 저장된 문자열이 그대로 출력되는 것을 볼 수 있다.

>>> food

"Python's favorite food is perl"

시험 삼아 다음과 같이 큰따옴표(")가 아닌 작은따옴표(')로 문자열을 둘러싼 후 다시 실행해 보자. 'Python'이 문자열로 인식되어 구문 오류(SyntaxError)가 발생할 것이다.

>>> food = 'Python's favorite food is perl'

File "<stdin>", line 1

food = 'Python's favorite food is perl'

^

SyntaxError: invalid syntax

**2. 문자열에 큰따옴표 (") 포함시키기**

"Python is very easy." he says.

위와 같이 큰따옴표(")가 포함된 문자열이라면 어떻게 해야 큰따옴표가 제대로 표현될까?

다음과 같이 문자열을 작은따옴표(')로 둘러싸면 된다.

>>> say = '"Python is very easy." he says.'

이렇게 작은따옴표(') 안에 사용된 큰따옴표(")는 문자열을 만드는 기호로 인식되지 않는다.

**3. 백슬래시(\)를 사용해서 작은따옴표(')와 큰따옴표(")를 문자열에 포함시키기**

>>> food = 'Python\'s favorite food is perl'

>>> say = "\"Python is very easy.\" he says."

작은따옴표(')나 큰따옴표(")를 문자열에 포함시키는 또 다른 방법은 백슬래시(\)를 사용하는 것이다. 즉 백슬래시(\)를 작은따옴표(')나 큰따옴표(") 앞에 삽입하면 백슬래시(\) 뒤의 작은따옴표(')나 큰따옴표(")는 문자열을 둘러싸는 기호의 의미가 아니라 문자 ('), (") 그 자체를 뜻하게 된다.

어떤 방법을 사용해서 문자열 안에 작은따옴표(')와 큰따옴표(")를 포함시킬지는 각자의 선택이다. 대화형 인터프리터를 실행한 후 위 예문을 꼭 직접 작성해 보자.

**여러 줄인 문자열을 변수에 대입하고 싶을 때**

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

Life is too short

You need python

**1. 줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기**

>>> multiline = "Life is too short\nYou need python"

위 예처럼 줄바꿈 문자 \n을 삽입하는 방법이 있지만 읽기에 불편하고 줄이 길어지는 단점이 있다.

**2. 연속된 작은따옴표 3개(''') 또는 큰따옴표 3개(""") 사용하기**

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

>>> multiline='''

... Life is too short

... You need python

... '''

*작은따옴표 3개를 사용한 경우*

>>> multiline="""

... Life is too short

... You need python

... """

*큰따옴표 3개를 사용한 경우*

print(multiline)을 입력해서 어떻게 출력되는지 확인해 보자.

>>> 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 따옴표를 연속해서 쓰는 것이 훨씬 깔끔하다.

**[이스케이프 코드란?]**

문자열 예제에서 여러 줄의 문장을 처리할 때 백슬래시 문자와 소문자 n을 조합한 \n 이스케이프 코드를 사용했다. 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"이다. 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 |
| \\ | 문자 \를 그대로 표현할 때 사용 |
| \' | 작은따옴표(')를 그대로 표현할 때 사용 |
| \" | 큰따옴표(")를 그대로 표현할 때 사용 |
| \r | 캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동) |
| \f | 폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동) |
| \a | 벨 소리(출력할 때 PC 스피커에서 '삑' 소리가 난다) |
| \b | 백 스페이스 |
| \000 | 널 문자 |

이중에서 활용빈도가 높은 것은 \n, \t, \\, \', \"이다. 나머지는 프로그램에서 잘 사용하지 않는다.

**문자열 연산하기**

파이썬에서는 문자열을 더하거나 곱할 수 있다. 다른 언어에서는 쉽게 찾아볼 수 없는 재미있는 기능으로, 우리 생각을 그대로 반영해 주는 파이썬만의 장점이라고 할 수 있다. 문자열을 더하거나 곱하는 방법에 대해 알아보자.

**문자열 더해서 연결하기(Concatenation)**

>>> head = "Python"

>>> tail = " is fun!"

>>> head + tail

'Python is fun!'

위 소스 코드에서 세 번째 줄을 보자. 복잡하게 생각하지 말고 눈에 보이는 대로 생각해 보자. "Python"이라는 head 변수와 " is fun!"이라는 tail 변수를 더한 것이다. 결과는 'Python is fun!'이다. 즉 head와 tail 변수가 +에 의해 합쳐진 것이다.

직접 실행해 보고 결괏값이 제시한 것과 똑같이 나오는지 확인해 보자.

**문자열 곱하기**

>>> a = "python"

>>> a \* 2

'pythonpython'

위 소스 코드에서 \*의 의미는 우리가 일반적으로 사용하는 숫자 곱하기의 의미와는 다르다. 위 소스 코드에서 a \* 2 문장은 a를 두 번 반복하라는 뜻이다. 즉 \*는 문자열의 반복을 뜻하는 의미로 사용되었다. 굳이 코드의 의미를 설명할 필요가 없을 정도로 직관적이다.

**문자열 곱하기 응용**

문자열 곱하기를 좀 더 응용해 보자. 다음 소스를 IDLE 에디터를 열고 작성해 보자.

# multistring.py

print("=" \* 50)

print("My Program")

print("=" \* 50)

입력한 소스는 C:\doit 디렉터리에 파일 이름 multistring.py로 저장하자.

이제 프로그램을 실행해 보자. [윈도우 + R(실행) → cmd 입력 → Enter]를 눌러 명령 프롬프트 창을 열고 다음을 따라 해 보자. 결괏값이 다음과 같이 나타날 것이다.

C:\Users>cd C:\doit

C:\doit>python multistring.py

==================================================

My Program

==================================================

이런 식의 표현은 앞으로 자주 사용하게 될 것이다. 프로그램을 만들어 실행시켰을 때 출력되는 화면 제일 위쪽에 프로그램 제목을 이와 같이 표시하면 보기 좋지 않겠는가?

**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.

>>> a = "Life is too short"

>>> len(a)

17

**문자열 인덱싱과 슬라이싱**

인덱싱(Indexing)이란 무엇인가를 "가리킨다"는 의미이고, 슬라이싱(Slicing)은 무엇인가를 "잘라낸다"는 의미이다. 이런 의미를 생각하면서 다음 내용을 살펴보자.

**문자열 인덱싱이란?**

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

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

Life **is** too short, You need Python

0 1 2 3

0123456789012345678901234567890123

"Life is too short, You need Python" 문자열에서 L은 첫 번째 자리를 뜻하는 숫자 0, 바로 다음인 i는 1 이런 식으로 계속 번호를 붙인 것이다. 중간에 있는 short의 s는 12가 된다.

이제 다음 예를 실행해 보자.

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

>>> a[3]

'e'

a[3]이 뜻하는 것은 a라는 문자열의 네 번째 문자 e를 말한다. 프로그래밍을 처음 접하는 독자라면 a[3]에서 숫자 3이 왜 네 번째 문자를 뜻하는지 의아할 수도 있다. 사실 이 부분이 헷갈릴 수 있는 부분인데, 이렇게 생각하면 쉽게 알 수 있을 것이다.

**"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]:' ', ...

0부터 숫자를 센다는 것이 처음에는 익숙하지 않겠지만 계속 사용하다 보면 자연스러워질 것이다. 위 예에서 볼 수 있듯이 a[번호]는 문자열 안의 특정한 값을 뽑아내는 역할을 한다. 이러한 작업을 인덱싱이라고 한다.

**문자열 인덱싱 활용하기**

인덱싱 예를 몇 가지 더 보자.

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

>>> a[0]

'L'

>>> a[12]

's'

>>> a[-1]

'n'

앞의 a[0]과 a[12]는 쉽게 이해할 수 있는데 마지막의 a[-1]이 뜻하는 것은 뭘까? 눈치 빠른 독자는 이미 알아챘겠지만 문자열을 뒤에서부터 읽기 위해 마이너스(-) 기호를 붙이는 것이다. 즉 a[-1]은 뒤에서부터 세어 첫 번째가 되는 문자를 말한다. a의 값은 "Life is too short, You need Python" 문자열이므로 뒤에서부터 첫 번째 문자는 가장 마지막 문자 'n'이다.

뒤에서부터 첫 번째 문자를 표시할 때도 0부터 세어 "a[-0]이라고 해야 하지 않을까?"라는 의문이 들 수도 있겠지만 잘 생각해 보자. 0과 -0은 똑같은 것이기 때문에 a[-0]은 a[0]과 똑같은 값을 보여 준다.

>>> a[-0]

'L'

계속해서 몇 가지 예를 더 보자.

>>> a[-2]

'o'

>>> a[-5]

'y'

위 첫 번째 예는 뒤에서부터 두 번째 문자를 가리키는 것이고, 두 번째 예는 뒤에서부터 다섯 번째 문자를 가리키는 것이다.

**문자열 슬라이싱이란?**

그렇다면 "Life is too short, You need Python" 문자열에서 단순히 한 문자만을 뽑아내는 것이 아니라 'Life' 또는 'You' 같은 단어를 뽑아내는 방법은 없을까?

다음과 같이 하면 된다.

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

>>> b = a[0] + a[1] + a[2] + a[3]

>>> b

'Life'

위 방법처럼 단순하게 접근할 수도 있지만 파이썬에서는 더 좋은 방법을 제공한다. 바로 슬라이싱(Slicing) 기법이다.

※ 인덱싱 기법과 슬라이싱 기법은 뒤에서 배울 자료형인 리스트나 튜플에서도 사용할 수 있다.

위 예는 슬라이싱 기법으로 다음과 같이 간단하게 처리할 수 있다.

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

>>> a[0:4]

'Life'

a[0:4]가 뜻하는 것은 a 문자열, 즉 "Life is too short, You need Python" 문장에서 자리 번호 0부터 4까지의 문자를 뽑아낸다는 뜻이다. 하지만 다음과 같은 의문이 생길 것이다. a[0]은 L, a[1]은 i, a[2]는 f, a[3]은 e니까 a[0:3]으로도 Life라는 단어를 뽑아낼 수 있지 않을까? 다음 예로 확인해 보자.

>>> a[0:3]

'Lif'

이렇게 되는 이유는 간단하다. 슬라이싱 기법으로 a[시작 번호:끝 번호]를 지정할 때 끝 번호에 해당하는 것은 포함하지 않기 때문이다. a[0:3]을 수식으로 나타내면 다음과 같다.

0 <= a < 3

이 수식을 만족하는 것은 a[0], a[1], a[2]이다. 따라서 a[0:3]은 'Lif'이고 a[0:4]는 'Life'가 되는 것이다. 이 부분이 문자열 연산에서 가장 혼동하기 쉬운 부분이니 장 마지막의 연습 문제를 많이 풀어 보면서 몸에 익히기 바란다.

**문자열을 슬라이싱하는 방법**

슬라이싱의 예를 조금 더 보자.

>>> a[0:5]

'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 같은 문자와 동일하게 취급되는 것을 잊지 말자. 'Life'와 'Life '는 완전히 다른 문자열이다.

슬라이싱할 때 항상 시작 번호가 0일 필요는 없다.

>>> a[0:2]

'Li'

>>> a[5:7]

'is'

>>> a[12:17]

'short'

a[시작 번호:끝 번호]에서 끝 번호 부분을 생략하면 시작 번호부터 그 문자열의 끝까지 뽑아낸다.

>>> a[19:]

'You need Python'

a[시작 번호:끝 번호]에서 시작 번호를 생략하면 문자열의 처음부터 끝 번호까지 뽑아낸다.

>>> a[:17]

'Life is too short'

a[시작 번호:끝 번호]에서 시작 번호와 끝 번호를 생략하면 문자열의 처음부터 끝까지를 뽑아낸다.

>>> a[:]

'Life is too short, You need Python'

슬라이싱에서도 인덱싱과 마찬가지로 마이너스(-) 기호를 사용할 수 있다.

>>> a[19:-7]

'You need'

위 소스 코드에서 a[19:-7]이 뜻하는 것은 a[19]에서부터 a[-8]까지를 말한다. 이 역시 a[-7]은 포함하지 않는다.

**슬라이싱으로 문자열 나누기**

다음은 자주 사용하게 되는 슬라이싱 기법 중 하나이다.

>>> a = "20010331Rainy"

>>> date = a[:8]

>>> weather = a[8:]

>>> date

'20010331'

>>> weather

'Rainy'

위 예는 문자열 a를 두 부분으로 나누는 기법이다. 숫자 8을 기준으로 문자열 a를 양쪽으로 한 번씩 슬라이싱했다. a[:8]은 a[8]이 포함되지 않고, a[8:]은 a[8]을 포함하기 때문에 8을 기준으로 해서 두 부분으로 나눌 수 있는 것이다. 위 예에서는 "20010331Rainy" 문자열을 날짜를 나타내는 부분인 '20010331'과 날씨를 나타내는 부분인 'Rainy'로 나누는 방법을 보여 준다.

위 문자열 "20010331Rainy"를 연도 2001, 월과 일을 나타내는 0331, 날씨를 나타내는 Rainy의 세 부분으로 나누려면 다음과 같이 할 수 있다.

>>> a = "20010331Rainy"

>>> year = a[:4]

>>> day = a[4:8]

>>> weather = a[8:]

>>> year

'2001'

>>> day

'0331'

>>> weather

'Rainy'

위 예는 숫자 4와 8로 "20010331Rainy" 문자열을 세 부분으로 나누는 방법을 보여 준다.

지금까지 인덱싱과 슬라이싱에 대해서 살펴보았다. 인덱싱과 슬라이싱은 프로그래밍을 할때 매우 자주 사용하는 기법이니 꼭 반복해서 연습해 두자.

**["Pithon"이라는 문자열을 "Python"으로 바꾸려면?]**

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

>>> a = "Pithon"

>>> a[1]

'i'

>>> a[1] = 'y'

즉 a 변수에 "Pithon" 문자열을 대입하고 a[1]의 값이 i니까 a[1]을 y로 바꾸어 준다는 생각이다. 하지만 결과는 어떻게 나올까?

당연히 오류가 발생한다. 왜냐하면 문자열의 요솟값은 바꿀 수 있는 값이 아니기 때문이다(문자열 자료형은 그 요솟값을 변경할 수 없다. 그래서 immutable한 자료형이라고도 부른다).

하지만 앞에서 살펴본 슬라이싱 기법을 사용하면 Pithon 문자열을 사용해 Python 문자열을 만들 수 있다.

다음 예를 보자.

>>> a = "Pithon"

>>> a[:1]

'P'

>>> a[2:]

'thon'

>>> a[:1] + 'y' + a[2:]

'Python'

위 예에서 볼 수 있듯이 슬라이싱을 사용하면 "Pithon" 문자열을 'P' 부분과 'thon' 부분으로 나눌 수 있기 때문에 그 사이에 'y' 문자를 추가하여 'Python'이라는 새로운 문자열을 만들 수 있다.

**문자열 포매팅**

문자열에서 또 하나 알아야 할 것으로는 문자열 포매팅(Formatting)이 있다. 이것을 공부하기 전에 다음과 같은 문자열을 출력하는 프로그램을 작성했다고 가정해 보자.

**"현재 온도는 18도입니다."**

시간이 지나서 20도가 되면 다음 문장을 출력한다.

**"현재 온도는 20도입니다"**

위 두 문자열은 모두 같은데 20이라는 숫자와 18이라는 숫자만 다르다. 이렇게 문자열 안의 특정한 값을 바꿔야 할 경우가 있을 때 이것을 가능하게 해주는 것이 바로 문자열 포매팅 기법이다.

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**

>>> "I eat %d apples." % 3

'I eat 3 apples.'

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

**2. 문자열 바로 대입**

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

>>> "I eat %s apples." % "five"

'I eat five apples.'

위 예제에서는 문자열 안에 또 다른 문자열을 삽입하기 위해 앞에서 사용한 문자열 포맷 코드 %d가 아닌 %s를 썼다. 어쩌면 눈치 빠른 독자는 벌써 유추하였을 것이다. 숫자를 넣기 위해서는 %d를 써야 하고, 문자열을 넣기 위해서는 %s를 써야 한다는 사실을 말이다.

※ 문자열을 대입할 때는 앞에서 배운 것처럼 큰따옴표나 작은따옴표를 반드시 써주어야 한다.

**3. 숫자 값을 나타내는 변수로 대입**

>>> number = 3

>>> "I eat %d apples." % number

'I eat 3 apples.'

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

**4. 2개 이상의 값 넣기**

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

>>> number = 10

>>> day = "three"

>>> "I ate %d apples. so I was sick for %s days." % (number, day)

'I ate 10 apples. so I was sick for three days.'

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

**문자열 포맷 코드**

문자열 포매팅 예제에서는 대입해 넣는 자료형으로 정수와 문자열을 사용했으나 이 외에도 다양한 것을 대입할 수 있다. 문자열 포맷 코드로는 다음과 같은 것이 있다.

| **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 코드는 어떤 형태의 값이든 변환해 넣을 수 있다. 무슨 말인지 예를 통해 확인해 보자.

>>> "I have %s apples" % 3

'I have 3 apples'

>>> "rate is %s" % 3.234

'rate is 3.234'

3을 문자열 안에 삽입하려면 %d를 사용하고, 3.234를 삽입하려면 %f를 사용해야 한다. 하지만 %s를 사용하면 이런 것을 생각하지 않아도 된다. 왜냐하면 %s는 자동으로 % 뒤에 있는 값을 문자열로 바꾸기 때문이다.

**[포매팅 연산자 %d와 %를 같이 쓸 때는 %%를 쓴다]**

>>> "Error is %d%." % 98

위 예문의 결괏값으로 당연히 "Error is 98%."가 출력될 것이라고 예상하겠지만 파이썬은 값이 올바르지 않다는 값 오류(Value Error) 메시지를 보여 준다.

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: incomplete format

이유는 문자열 포맷 코드인 %d와 %가 같은 문자열 안에 존재하는 경우, %를 나타내려면 반드시 %%로 써야 하는 법칙이 있기 때문이다. 이 점은 꼭 기억해 두어야 한다. 하지만 문자열 안에 %d 같은 포매팅 연산자가 없으면 %는 홀로 쓰여도 상관이 없다.

따라서 위 예를 제대로 실행하려면 다음과 같이 해야 한다.

>>> "Error is %d%%." % 98

'Error is 98%.'

**포맷 코드와 숫자 함께 사용하기**

위에서 보았듯이 %d, %s 등의 포맷 코드는 문자열 안에 어떤 값을 삽입하기 위해 사용한다. 하지만 포맷 코드를 숫자와 함께 사용하면 더 유용하게 사용할 수 있다. 다음 예를 보고 따라해 보자.

**1. 정렬과 공백**

>>> "%10s" % "hi"

' hi'

앞의 예문에서 %10s는 전체 길이가 10개인 문자열 공간에서 대입되는 값을 오른쪽으로 정렬하고 그 앞의 나머지는 공백으로 남겨 두라는 의미이다.

그렇다면 반대쪽인 왼쪽 정렬은 %-10s가 될 것이다. 확인해 보자.

>>> "%-10sjane." % 'hi'

'hi jane.'

hi를 왼쪽으로 정렬하고 나머지는 공백으로 채웠음을 볼 수 있다.

**2. 소수점 표현하기**

>>> "%0.4f" % 3.42134234

'3.4213'

3.42134234를 소수점 네 번째 자리까지만 나타내고 싶은 경우에는 위와 같이 사용한다. 즉 여기서 '.'의 의미는 소수점 포인트를 말하고 그 뒤의 숫자 4는 소수점 뒤에 나올 숫자의 개수를 말한다. 다음 예를 살펴보자.

>>> "%10.4f" % 3.42134234

' 3.4213'

위 예는 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**

>>> "I eat {0} apples".format(3)

'I eat 3 apples'

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

**문자열 바로 대입하기**

>>> "I eat {0} apples".format("five")

'I eat five apples'

문자열의 {0} 항목이 five라는 문자열로 바뀌었다.

**숫자 값을 가진 변수로 대입하기**

>>> number = 3

>>> "I eat {0} apples".format(number)

'I eat 3 apples'

문자열의 {0} 항목이 number 변수 값인 3으로 바뀌었다.

**2개 이상의 값 넣기**

>>> number = 10

>>> day = "three"

>>> "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. 즉 위 예에서 {0}은 format 함수의 첫 번째 입력값인 number로 바뀌고 {1}은 format 함수의 두 번째 입력값인 day로 바뀐다.

**이름으로 넣기**

>>> "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.'

위 예에서 볼 수 있듯이 {0}, {1}과 같은 인덱스 항목 대신 더 편리한 {name} 형태를 사용하는 방법도 있다. {name} 형태를 사용할 경우 format 함수에는 반드시 name=value 와 같은 형태의 입력값이 있어야만 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**

>>> "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.'

위와 같이 인덱스 항목과 name=value 형태를 혼용하는 것도 가능하다.

**왼쪽 정렬**

>>> "{0:<10}".format("hi")

'hi '

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

**오른쪽 정렬**

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

' hi'

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

**가운데 정렬**

>>> "{0:^10}".format("hi")

' hi '

:^ 기호를 사용하면 가운데 정렬도 가능하다.

**공백 채우기**

>>> "{0:=^10}".format("hi")

'====hi===='

>>> "{0:!<10}".format("hi")

'hi!!!!!!!!'

정렬할 때 공백 문자 대신에 지정한 문자 값으로 채워 넣는 것도 가능하다. 채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 한다. 위 예에서 첫 번째 예제는 가운데(^)로 정렬하고 빈 공간을 = 문자로 채웠고, 두 번째 예제는 왼쪽(<)으로 정렬하고 빈 공간을 ! 문자로 채웠다.

**소수점 표현하기**

>>> y = 3.42134234

>>> "{0:0.4f}".format(y)

'3.4213'

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

>>> "{0:10.4f}".format(y)

' 3.4213'

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

**{ 또는 } 문자 표현하기**

>>> "{{ and }}".format()

'{ and }'

format 함수를 사용해 문자열 포매팅을 할 경우 { }와 같은 중괄호(brace) 문자를 포매팅 문자가 아닌 문자 그대로 사용하고 싶은 경우에는 위 예의 {{ }}처럼 2개를 연속해서 사용하면 된다.

**f 문자열 포매팅**

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 = 30

>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'

'나의 이름은 홍길동입니다. 나이는 30입니다.'

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

※ 표현식이란 문자열 안에서 변수와 +, -와 같은 수식을 함께 사용하는 것을 말한다.

>>> age = 30

>>> f'나는 내년이면 {age+1}살이 된다.'

'나는 내년이면 31살이 된다.'

딕셔너리는 f 문자열 포매팅에서 다음과 같이 사용할 수 있다.

※ 딕셔너리는 Key와 Value라는 것을 한 쌍으로 갖는 자료형이다. 02-5에서 자세히 알아본다.

>>> d = {'name':'홍길동', 'age':30}

>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'

'나의 이름은 홍길동입니다. 나이는 30입니다.'

정렬은 다음과 같이 할 수 있다.

>>> f'{"hi":<10}' # 왼쪽 정렬

'hi '

>>> f'{"hi":>10}' # 오른쪽 정렬

' hi'

>>> f'{"hi":^10}' # 가운데 정렬

' hi '

공백 채우기는 다음과 같이 할 수 있다.

>>> f'{"hi":=^10}' # 가운데 정렬하고 '=' 문자로 공백 채우기

'====hi===='

>>> f'{"hi":!<10}' # 왼쪽 정렬하고 '!' 문자로 공백 채우기

'hi!!!!!!!!'

소수점은 다음과 같이 표현할 수 있다.

>>> y = 3.42134234

>>> f'{y:0.4f}' # 소수점 4자리까지만 표현

'3.4213'

>>> f'{y:10.4f}' # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤

' 3.4213'

f 문자열에서 { } 문자를 표시하려면 다음과 같이 두 개를 동시에 사용해야 한다.

>>> f'{{ and }}'

'{ and }'

지금까지는 문자열을 가지고 할 수 있는 기본적인 것에 대해 알아보았다. 이제부터는 문자열을 좀 더 자유자재로 다루기 위해 공부해야 할 것을 설명할 것이다. 지쳤다면 잠시 책을 접고 휴식을 취하자.

**문자열 관련 함수들**

문자열 자료형은 자체적으로 함수를 가지고 있다. 이들 함수를 다른 말로 문자열 내장 함수라 한다. 이 내장 함수를 사용하려면 문자열 변수 이름 뒤에 ‘.’를 붙인 다음에 함수 이름을 써주면 된다. 이제 문자열의 내장 함수에 대해서 알아보자.

**문자 개수 세기(count)**

>>> a = "hobby"

>>> a.count('b')

2

문자열 중 문자 b의 개수를 돌려준다.

**위치 알려주기1(find)**

>>> a = "Python is the best choice"

>>> a.find('b')

14

>>> a.find('k')

-1

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

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

**위치 알려주기2(index)**

>>> a = "Life is too short"

>>> a.index('t')

8

>>> a.index('k')

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

ValueError: substring **not** found

문자열 중 문자 t가 맨 처음으로 나온 위치를 반환한다. 만약 찾는 문자나 문자열이 존재하지 않는다면 오류를 발생시킨다. 앞의 find 함수와 다른 점은 문자열 안에 존재하지 않는 문자를 찾으면 오류가 발생한다는 점이다.

**문자열 삽입(join)**

>>> ",".join('abcd')

'a,b,c,d'

abcd 문자열의 각각의 문자 사이에 ','를 삽입한다.

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

>>> ",".join(['a', 'b', 'c', 'd'])

'a,b,c,d'

**소문자를 대문자로 바꾸기(upper)**

>>> a = "hi"

>>> a.upper()

'HI'

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

**대문자를 소문자로 바꾸기(lower)**

>>> a = "HI"

>>> a.lower()

'hi'

lower 함수는 대문자를 소문자로 바꾸어 준다.

**왼쪽 공백 지우기(lstrip)**

>>> a = " hi "

>>> a.lstrip()

'hi '

문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백들을 모두 지운다. lstrip에서 l은 left를 의미한다.

**오른쪽 공백 지우기(rstrip)**

>>> a= " hi "

>>> a.rstrip()

' hi'

문자열 중 가장 오른쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다. rstrip에서 r는 right를 의미한다.

**양쪽 공백 지우기(strip)**

>>> a = " hi "

>>> a.strip()

'hi'

문자열 양쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다.

**문자열 바꾸기(replace)**

>>> a = "Life is too short"

>>> a.replace("Life", "Your leg")

'Your leg is too short'

replace(바뀌게 될 문자열, 바꿀 문자열)처럼 사용해서 문자열 안의 특정한 값을 다른 값으로 치환해 준다.

**문자열 나누기(split)**

>>> a = "Life is too short"

>>> a.split()

['Life', 'is', 'too', 'short']

>>> b = "a:b:c:d"

>>> b.split(':')

['a', 'b', 'c', 'd']

split 함수는 a.split()처럼 괄호 안에 아무 값도 넣어 주지 않으면 공백(스페이스, 탭, 엔터 등)을 기준으로 문자열을 나누어 준다. 만약 b.split(':')처럼 괄호 안에 특정 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 준다. 이렇게 나눈 값은 리스트에 하나씩 들어가게 된다. ['Life', 'is', 'too', 'short']나 ['a', 'b', 'c', 'd']가 리스트인데 02-3에서 자세히 알아볼 것이니 여기에서는 너무 신경 쓰지 않아도 된다.

위에서 소개한 문자열 관련 함수는 문자열 처리에서 사용 빈도가 매우 높고 유용하다. 이 외에도 몇 가지가 더 있지만 자주 사용되지는 않는다.