6. 파이썬 자료형 - 문자열

Made By. 김규일

- 문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 나타낸다. 예를 들어 아래와 같은 것이 문자열이다.

```
"Hello World"
"a"
"123"
```

- 위 문자열을 잘 보면 모두 큰 따음표로 둘러싸여 있다. 따라서 숫자형 123을 쓰더라고 큰 따음표안에 작성하면 정수형 123이 아닌 문자열 123이 되는 것이다.

문자열 만드는 방식

- 큰따음표로 양쪽 감싸기

```
"Hello World"
```

- 작은따음표로 양쪽 감싸기

```
'Hello World'
```

- 큰따음표 3개를 연속으로 써서 양쪽 감싸기

```
"""Hello World"""
```

- 작은따음표 3개를 연속으로 써서 양쪽 감싸기

```
'''Hello World'''
```

- 파이썬에서 문자열 만드는 방법을 4개로 나뉜 이유는 아래에서 확인할 수 있다.

문자열 안에 작은따음표, 큰따음표를 사용하고 싶을 때

- 파이썬에서 큰따음표와 작은따음표를 나누는 이유는 문자열에서 큰따음표와 작은따음표를 사용하기 위해서이다.
- 문자열 안에서 작은따음표를 사용하고 싶다면 문자열을 큰따음표를 이용하여 만들면 된다. 큰따음표를 이용해서 만들면 문자열 안에 포함되는 작은따음표는 자동으로 문자 취급이 된다. 반대의 상황도 똑같다.

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

- 위의 방법 외에 백슬래시('/')를 이용하는 방법이 존재한다. 작은따음표나 큰따음표 앞에 백슬래시를 포함시켜 문자열로 취급하여 사용할 수 있다. 이때는 문자열을 감쌀 때 작은따음표, 큰따음표 구분 없이 사용 가능하다.

```
>>> food = 'Python\'s favorite food is perl'
>>> say = "\"Python is very easy.\" he says."
```

문자열을 여러 줄로 만들고 싶을 때

1. 줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기

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

- 하지만 위와 같은 방식의 단점은 코드를 읽기 불편하고 줄이 길어진다는 단점이 있다.

2. 연속된 작은따음표나 큰따음표를 3개 이용하기

```
>>> multiline='''
... Life is too short
... You need python
... '''
```

```
>>> multiline="""
... Life is too short
... You need python
... """

>>> print(multiline)
Life is too short
You need python
```

- 위와 같이 2가지의 방식을 이용해서 여러 줄을 하나의 문자열에 포함시킬 수 있다.

이스케이프 코드

- 이스케이프 코드는 다양한 역할을 하는 문자 조합이다. 코드 출력을 보기 좋게 정렬하는 용도로 많이 사용한다.

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

문자열 연산하기

- 문자열 더하기
- 문자열을 단순히 더하기로 이용해서 연산하게 되면 두 문자열을 이어주는 역할을 한다.
- 만약 1 + 2가 3이 아닌 12가 나왔다면 1, 2는 문자열로 표현되어 있는 것이다.

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

- 문자열 곱하기
- 문자열에서는 곱하라는 의미가 아닌 반복하라는 의미이다. 예를 들어 a*2가 된다면 a를 2번 반복하라는 의미이다.

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

- 문자열 곱하기 응용

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

# 결과는 아래와 같다.
==================================================
My Program
==================================================
```

문자열 길이 구하기

- 문자열의 길이는 len 함수를 이용하여 쉽게 구할 수 있다.

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

문자열 인덱싱과 슬라이싱

- 인덱싱(Indexing)이란 무엇인가를 가리킨다는 의미이고 슬라이싱(Slicing)은 무엇인가를 잘라낸다는 의미

```
"Hello World"
```

- 위와 같은 문자열이 있을 때 각각의 인덱스 번호는 맨 앞부터 차례대로 0이 할당된다.
- 즉 H = 0, e = 1, l = 2, l = 3, o = 4, 공백 = 5, W = 6, o = 7, r = 8, l = 9, d = 10이 된다.

```
>>> a = "Hello World"
>>> a[3]
'l'
```

- 위처럼 [] 기호와 인덱스 번호를 통해 원하는 위치의 문자를 바로 꺼낼 수 있다.
- 여기서 중요한 점은 인덱스는 항상 0부터 시작한다는 점이다.
- 인덱스를 마지막 기준으로 본다면 -1부터 시작한다.
- 즉, 여기서 a[-1] = ‘d’ 가 된다. a[-2] = l, 이런식으로 나타난다.
- a[-0] = a[0]과 같다.

- 문자열 슬라이싱은 위의 인덱싱을 활용하여 할 수 있다.
- 슬라이싱은 클론 “:” 문자를 활용하여 사용한다.

```
>>> a = "Life is too short, You need Python"
>>> a[0:3]
'Lif'
```

- 클론을 통해 왼쪽에는 시작점의 인덱스를 오른쪽에는 끝나는 값의 인덱스를 넣으면 위와 같이 슬라이싱을 할 수 있다.
- 여기서 중요한 점은 클론의 오른쪽에 써있는 인덱스 번호까지 출력이 아닌 그 전까지 출력이다.
- 쉽게 생각하면 슬라이싱은 왼쪽값 $<= a <$ 오른쪽값 으로 이해하면 쉽다.
- 시작번호와 끝번호는 생략 가능하다.

- 시작번호를 생략하면 처음부터, 끝번호를 생략하면 마지막까지, 둘 다 생략하면 처음부터 끝까지이다.
- 또한 인덱싱에서 배운 -1 등의 인덱스 번호를 통해 슬라이싱 또한 가능하다.

문자열 슬라이싱 활용

```
>>> a = "20010331Rainy"
>>> date = a[:8]
>>> weather = a[8:]
>>> date
'20010331'
>>> weather
'Rainy'
```

```
>>> a = "20010331Rainy"
>>> year = a[:4]
>>> day = a[4:8]
>>> weather = a[8:]
>>> year
'2001'
>>> day
'0331'
>>> weather
'Rainy'
```

문자열 포매팅

- 문자열 포매팅은 문자열을 출력할 때 조금 더 쉽게 할 수 있는 방법들이다.
- 문자열 안에 변수를 삽입할 때 사용한다.

  1) 숫자 바로 대입

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

  숫자를 넣고 싶은 자리에 %d 연산자를 통해 매핑할 수 있다.

  %d는 문자열 포맷 코드로 여러가지가 있다.

2) 문자열 바로 대입

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

%s 연산을 통해 문자열을 바로 대입할 수 있다.

3) 변수를 대입

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

해당 자료형에 맞게 변수를 대입할 수 있다.

4) 여러 개의 변수 또는 값 넣기

```
>>> 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.'
```

여러 개의 포맷 코드를 활용하고 괄호를 통해서 여러 개의 값을 대입할 수 있다.

문자열 포맷 코드

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

포맷 코드 활용

1. 정렬과 공백

```
>>> "%10s" % "hi"
'        hi'
```

%10s는 전체길이가 10인 문자열안에 포맷 코드 s에 값을 대입하는 것이다.

따라서 위와 같이 공백으로 정렬이 된다.

반대의 경우는 -를 활용하면 된다.

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

2. 소수점 표현하기

```
>>> "%0.4f" % 3.42134234
'3.4213'
```

0.4 등 포맷 코드 앞에 소수를 붙여 자리수를 표현할 수 있다.

또한 공백과 소수점 표현을 합쳐 다음과 같은 표현도 가능하다.

```
>>> "%10.4f" % 3.42134234
'    3.4213'
```

format 함수를 이용한 포맷팅

- 포맷 코드가 아닌 foramt 함수를 이용하여 포매팅 할 수 있다.

```
>>> 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.'
```

- 원하는 자리를 포맷 코드 대신 {}로 써주고 문자열의 종료 이후 .format() 함수를 이용하여 포매팅 할 수 있다.
- 이때 {} 안에 숫자는 생략 가능하고 foramt() 안의 변수가 문자열 안의 {} 안에 차례대로 대입된다.
- 또한 {}에 이름을 붙여 대입할 수 있다.

```
>>> "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.'
```

- 왼쪽 정렬

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

- 오른쪽 정렬

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

- 가운데 정렬

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

- 공백 채우기

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

- 소수점 표현

```
>>> y = 3.42134234
>>> "{0:0.4f}".format(y)
```

```
'3.4213'
```

- {,} 문자 사용하기

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

f 문자열 포매팅

- foramt 함수가 아닌 f 예약어를 통해 문자열 포매팅을 할 수 있다.

```
>>> age = 30
>>> f'나는 내년이면 {age+1}살이 된다.'
'나는 내년이면 31살이 된다.'


## 딕셔너리 포매팅
>>> d = {'name':'홍길동', 'age':30}
>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'

## 정렬하기
>>> f'{"hi":<10}'  # 왼쪽 정렬
'hi        '
>>> f'{"hi":>10}'  # 오른쪽 정렬
'        hi'
>>> f'{"hi":^10}'  # 가운데 정렬
'    hi    '

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

## 소수점 표현하기
>>> 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'

## { } 기호 사용하기
>>> f'{{ and }}'
'{ and }'
```

문자열 관련 함수들

- 문자열 개수 세기(count)

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

- 위치 알려주기(find)

```
>>> a = "Python is the best choice"
>>> a.find('b')
14
>>> a.find('k')
-1

## 문자열이 처음 나타난 자리를 반환, 찾지 못하면 -1 반환
```

- 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

## 문자열이 처음 나타난 자리를 반환, 찾지 못하면 오류를 발생
```

- 문자열 삽입(join)

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

## abcd 문자열 사이에 , 삽입
```

- 소문자를 대문자로 바꾸기(upper)

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

```
'HI'
```

- 대문자를 소문자로 바꾸기(lower)

```
>>> a = "HI"
>>> a.lower()
'hi'
```

- 왼쪽 공백 지우기(lstrip)

```
>>> a = " hi "
>>> a.lstrip()
'hi '
```

- 오른쪽 공백 지우기(rstrip)

```
>>> a= " hi "
>>> a.rstrip()
' hi'
```

- 양쪽 공백 지우기(strip)

```
>>> a = " hi "
>>> a.strip()
'hi'
```

- 문자열 바꾸기(replace)

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'

## replace(바뀌게 될 문자열, 바꿀 문자열)
```

- 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']

## 함수 안의 값을 기준으로 나눈다. 만약 함수 안이 공백인 경우 스페이스, 공백, 탭을 기준으로 나눈다.
## 나눠진 문자열들은 리스트안에 삽입이 된다.
```