[문자열(string)]

- 문자열
- 문자열 관련 함수

자료구조

- 프로그램에서 자료를 저장하는 구조들이 여러 종류 있음
- 이것을 자료 구조(data structure)라고 함
- 자료 구조는 서로 관련이 있는 자료들을 효율적으로 처리하고 저장하는 데 사용
- 문자열, 리스트, 튜플, 사전, 집합 등의 특성과 사용법에 대해 공부

문자열

- 문자열(string)은 'abc', "hello"처럼 연속적으로 이어진 글자들의 집합을 작은따옴표(' ') 또는 큰따옴표(" ")로 묶은 것
- 문자열은 한줄인 경우에는 큰따옴표 또는 작은따옴표로 둘러싸며
- 여러 줄로 계속되는 경우에는 연속적인 큰따옴표(또는 작은따옴표) 3개로 앞, 뒤를 둘러싸면 됨
- 큰따옴표 를 이용해 문자열을 만들더라도 print()로 출력하면 작은따옴표를 사용해 출력

문자열 예

```
>>> str1 = "Python"
>>> str2 = 'language'
>>> str3 = """
multi-line
string
example
"""
>>> str3
'\nmulti-line\nstring\nexample\n'
```

Escape 코드

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

문자열의 index(첨자)

- 문자열에 인덱스(첨자)를 붙일 수 있는 데, 첨자는 대괄호 속에 표시
- 가장 첫 글자의 첨자는 0이고 하나씩 증가한다.
- 다음 예의 문자열은 길이가 6이므로 첨자는 0부터 5까지 있다.

```
>>> word = 'Python'
>>> word[0]      # 인덱스(첨자) 0의 문자
'P'
>>> word[5]
'n'
```

문자열 연산 - 결합과 반복

- 덧셈과 곱셈 연산을 사용해 문자열을 결합시키고 반복할 수 있다.
- 더하기

```
>>> s1 = 'abc'
>>> s2 = 'def'
>>> s3 = s1 + s2
>>> s3
'abcdef'
```

- 곱하기

```
>>> s1 = 'abc'
>>> s1 * 3
'abcabcabc'
```

in, not in 연산자 사용 비교

- 연산자 : in과 not in

  `s in str` : 문자열 str이 문자열 s를 포함하고 있으면 True를 반환.

  그렇지 않으면 False를 반환(not in은 반대로 동작)

```
>>> str = 'Python programming is easy!'
>>> 'is' in str
True
>>> 'gn' not in str
True
```

관계연산자 사용해 문자열 비교 분석

- 관계연산자를 사용해 문자열이 완전히 일치하는 지 파악

```
>>> 'apple' == 'banana'
False
>>> 'apple' != 'banana'
True
```

- 사전 순서로 앞, 뒤 관계를 파악

```
>>> 'apple' < 'banana'
True
>>> 'apple' >= 'banana'
False
```

문자열 길이 구하기

- 문자열의 길이는 파이썬의 기본 내장 함수 len()을 이용해 구할 수 있음
- 문자열의 길이에는 공백 문자도 포함

```
>>> a = 'python is easy'
>>> len(a)
14
>>> b = ''
>>> len(b)
0
```

음의 index(첨자)

- 첨자는 문자열 오른쪽의 맨 끝에서부터 음수로 시작할 수도 있다. -0은 0과 같으므로 음의 첨자는 -1부터 시작.
- 다음 그림을 보면 문자열의 첨자를 더 잘 이해할 수 있다.
  - 숫자의 첫 줄은 문자열의 첨자 0–6이며,
  - 두번째 줄은 같게 대응되는 음의 첨자

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
  0   1   2   3   4   5
 -6  -5  -4  -3  -2  -1
```

인덱싱과 슬라이싱

- 문자열의 첨자 하나를 사용하는 것을 인덱싱(indexing)이라고 한다.
- 인덱싱 외에 첨자 두 개를 사용하는 슬라이싱(slicing)도 가능
- 인덱싱이 첨자 하나만 사용하기 때문에 한 글자만 취할 수 있는 데 비해, 슬라이싱은 문자열의 일부를 얻을 수 있다.
- 슬라이싱은 첫 첨자(생략하면 기본값은 0)과 두번째 첨자(생략하면 기본값은 문자열의 길이)보다 하나 작은 위치까지의 문자열을 얻는 것이다.

여러 슬라이싱 예

```
>>> word = 'Python'
>>> word[0:2] # 첨자 0부터 1(=2-1)까지
'Py'
>>> word[2:5] # 첨자 2부터 4(=5-1)까지
'tho'
>>> word[:2] # 첨자 0부터 1까지의 문자
'Py'
>>> word[4:] # 첨자 4부터 끝까지
'on'
>>> word[-2:]
'on'
```

슬라이싱

- 슬라이싱에서 첨자의 처음은 포함되고 마지막은 포함되지 않기 때문에 word[:i] + word[i:]는 항상 word와 같다.
- `L[start:end:step]` 모양이 range() 객체와 유사?

```
>>> word[:]
'Python'
>>> word[::]
'Python'
>>> word[::-1] # step -1이면 끝에서부터 거꾸로
'nohtyP'
>>> word[::2] # step 2
'Pto'
>>> word[::-2] # step -2이면 끝에서부터 2만큼씩
'nhy'
```

문자열 포맷팅 - print() 함수와 주로 사용 가능

- 수와 문자열 대입

```
>>> print('%d apples' %2)       # 정수
'2 apples'
>>> print('%s donuts' %'four') # 문자열
'four donuts'
>>> print('%f' %f)              # 실수
1.200000
>>> n = 2
>>> print('%d eggs' %n)         # 숫자값 변수
'2 eggs'
>>> m = 3
>>> print('%d eggs. %d apples' %(n, m)) # 2개 이상의 값
'2 eggs. 3 apples'
```

print() 함수에서 사용 가능한 포맷과 사용 예

서식	설명	사용 예
%d	10진수 정수로 출력	n=5; print("%d" %n)
%f	실수로 출력	a=1.2; print("%f" %a)
%s	문자열 출력	b = "ok"; print("%s" %b)
%c	한 글자 출력	k = 'q'; print("%c" %k)
%x	16진수 정수로 출력	h = 15; print("%x" %h)
%o	8진수 정수로 출력	m = 10; print("%o" %m)
%%	% 문자 자체	print('error is %d%%' %10)

정렬과 공백

```
>>> '%8s' %'python'    # 오른쪽 정렬. 나머지 공백
'  python'
>>> '%-8s' %'python'   # 왼쪽 정렬. 나머지 공백
'python  '
>>> '%-7s vs C' %'python'
'python  vs C'
```

소수점 이하의 표현

```
>>> import math
>>> math.pi
3.141592653589793
>>> print('%.10f' %math.pi)
3.1415926536
>>> print('%8.4f' %math.pi)
  3.1416
>>> print('%8.4f' %-math.pi)
 -3.1416
>>> print('%f' %math.pi)
3.141593
```

format() 함수 사용

```
>>> v1=1.2; v2=2; v3 = v1*v2
>>> print('v1={0}, v2={1}, v3={2}'.format(v1,v2,v3))
v1=1.2, v2=2, v3=2.4
>>> print('{2}는 {0}과 {1}의 곱이다'.format(v1,v2,v3))
2.4는 1.2과 2의 곱이다
```

- 위에서 {0}, {1}, {2}는 0, 1, 2번째 인수가 각각 v1, v2, v3임을 의미

```
>>> print('{C}는 {A}과 {B}의 곱이다'.format(A=v1,B=v2,C=v3))
2.4는 1.2과 2의 곱이다
```

- 인수 대신 위에서 처럼 {A}와 같이 변수명으로도 사용 가능

f-string 포맷팅 기능 - 버전 3.6부터 가능

- 아래와 같이 사용 가능

```
>>> a = 3
>>> b = 5
>>> print(f'sum of {a} and {b} is {a+b}')
sum of 3 and 5 is 8
>>> name = 'Tom'
>>> print(f'His name is {name}.')
His name is Tom.
>>> v1=1.2; v2=2; v3 = v1*v2
print(f'v1={v1} v2={v2} v3={v3}')
v1=1.2 v2=2 v3=2.4
```

- 다음과 같이 단위가 큰 수에 콤마를 자동 삽입해 출력 가능

```
>>> print(f'{100000000:,}원')
100,000,000원
```

문자열 함수와 메소드

- 파이썬은 객체지향 프로그래밍 언어이기 때문에 사용되는 많은 것들이 객체(object)이다(뒤의 클래스에서 자세히 배움)
- 객체는 관련되는 변수와 함수를 묶은 것인데, 문자열도 객체임
- 객체에 어떤 연산이나 동작을 하고 싶을 때 객체의 이름 바로 뒤에 점(.)을 붙인 후에 함수의 이름을 적는 데 이 함수를 메소드 함수 또는 메소드(method)라고 함
- 이후의 내용 중 함수와 메소드를 명확히 구분하지 않고 그냥 함수라 사용하기도 함
- 파이썬에는 쉽게 프로그램에서 활용할 수 있는 다양한 문자열 처리함수와 메소드가 있음

문자열 처리함수와 메소드

- 문자열 처리함수와 메소드의 이름을 알고 싶으면 파이썬 셸에서 다음과 같이 입력 확인해 보자.

```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__getnewargs__', '__getstate__', '__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', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',
 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
 'zfill']
```

문자열 구성 파악 함수

- 각 문자열이 어떻게 구성되어 있는 지 쉽게 알아볼 수 있는 메소드
- isdigit(), isalpha(), isalnum(), isupper(), islower(), isspace() 등
- 이름을 보면 쉽게 짐작 가능할 수 있듯이 각각 숫자, 대소문자 알파벳, 알파벳과 숫자, 대문자, 소문자, 공백 문자를 파악할 때 사용
- 이 메소드들은 연산의 결과를 True 또는 False로 반환
- 문자열 메소드를 사용하는 방법은 문자열 변수 다음에 `.문자열메소드()` 를 입력
- `s.isdigit()` 은 문자열 s가 숫자로만 구성되어 있으면 True를 반환. 그렇지 않으면 False를 반환.
- 이외에도 isdecimal(), isnumeric() 메소드가 있음

문자열 구성 파악 함수

```
>>> '123'.isdigit() # 문자열에 숫자만 들어있나?
True
>>> 'abcABC'.isalpha() #문자열이 대소문자만 인가?
True
>>> 'Ab122'.isalnum() # 문자열이 문자+숫자 혼합인가?
True
>>> 'AB'.isupper() # 문자열이 대문자인가?
True
>>> 'ab'.islower() # 문자열이 소문자인가?
True
>>> ' '.isspace() # 문자열이 공백인가?
True
>>> a = '5²'
>>> a.isdecimal()
False
>>> a.isdigit()
True
>>> a.isnumeric()
True
```

대소문자 변환 함수

- upper()는 대문자로, lower()는 소문자로, swapcase()는 대소문자를 상호 변환
- title()는 각 단어의 첫 글자를 대문자로 변환, capitalize( )는 첫 글자만 대문자로 변환
- 사용 예는 다음과 같다. 직접 실행해 결과를 확인

```
>>> str = 'Python programming is easy!'
>>> str.upper() # 대문자로 변환
'PYTHON PROGRAMMING IS EASY!'
>>> str.lower() # 소문자로 변환
'python programming is easy!'
>>> str.swapcase() # 대소문자 상호 변환
'pYTHON PROGRAMMING IS EASY!'
>>> str.title() # 단어의 첫 글자를 대문자로 변환
'Python Programming Is Easy!'
```

문자열 찾기 함수

함수 : count(), find(), rfind(), index(), rindex(), startswith(), endswith() 함수가 있다.

- count() 는 해당 문자열에서 특정 문자열이 포함된 개수를 반환하는 함수
- find(), rfind()는 찾는 문자열의 처음 또는 마지막 위치를 반환. 없으면 -1을 반환

  사용 예

```
>>> str = 'Python programming is easy!'
>>> str.count('i') # 문자열에서 'i'의 수
2
>>> str.find('on') # 문자열에서 'on'의 처음 위치. 없으면 -1 반환
4
>>> str.rfind('sy') # 'sy'가 나오는 가장 나중 위치(또는 오른쪽부터 처음 위치)
24
```

문자열 찾기 함수(2)

- index(), rindex() : 찾는 문자열의 처음 또는 마지막 위치를 반환. 없으면 ValueError가 발생
- startswith(s) 함수와 endswith(s) 함수는 각각 해당 문자열 s로 시작 또는 끝나는 지의 여부를 True 또는 False로 반환
- 사용 예

```
>>> str = 'Python programming is easy!'
>>> str.index('on') # 문자열에서 'on'의 처음 위치. 없으면 ValueError
4
>>> str.rindex('n')
16
>>> 'Python'.startswith('Py')
True
>>> 'Python'.endswith('of')
False
```

문자열 공백 삭제, 변경 함수

- 문자열의 앞 뒤 공백이나 특정 문자를 삭제 또는 변경할 때 사용.
- strip()은 문자열 양쪽 끝을 자른다. rstrip(), lstrip()은 왼쪽끝, 오른쪽끝을 자른다. replace()는 문자열의 특정 부분을 변경할 때 사용한다.
- 사용 예

```
>>> str = ' hello '
>>> str.strip() # 양쪽에서 공백 제거
'hello'
>>> str # 원래 문자열은 변하지 않는다
' hello '
>>> str.rstrip() # 오른쪽 공백 제거
' hello'
>>> str.lstrip() # 왼쪽 공백 제거
'hello '
>>> str.replace('ll', 'RR')
' heRRo '
```

문자열 분리, 결합 함수

- split()은 문자열을 분리해 리스트를 반환
- splitlines()는 문자열에서 줄바꿈 기호를 제거한 리스트를 반환
- join()은 문자열 리스트를 전달받아 결합 문자를 중간에 넣어서 문자열을 결합

```
>>> s = 'a b' # a와 b 사이에 공백이 있다
>>> s.split() # s.split(None)와 같음
['a', 'b']
>>> str = 'abc'
>>> str.split('b') # 문자열을 ( )속의 문자로 분리해 리스트로 반환
['a', 'c']
>>> 'ab\ncd'.splitlines() #문자열에서 줄바꿈 기호를 제거한 리스트를 반환
['ab', 'cd']
>>> '*'.join('hello') # *와 문자열의 각 문자를 하나씩 결합한 문자열 생성
'h*e*l*l*o'
>>> '*'.join(['one','two'])
'one*two'
>>> '*'.join('python')
'p*y*t*h*o*n'
```

split() 함수의 이용 예

- 한 라인에서 실수와 정수를 입력받을 수 있다.

```
x, n = input('실수와 정수를 입력: ').split()
x = float(x)
n = int(n)
print(x**n)
```

- split()과 map() 함수를 사용한 수의 입력 방법

```
x, y, z = map(int, input('3개의 정수값을 입력: ').split())
print(x+y+z)
```

문자열 정렬, 채우기 함수

- center(), ljust(), rjust(), zfill() 함수가 있다.
- center(number)는 number만큼 자리를 잡은 후 문자열을 가운데 배치
- ljust() 함수는 왼쪽에 붙여서 출력하고, rjust() 함수는 오른쪽에 붙여서 출력

```
>>> str = 'abc'
>>> str.center(10)
'   abc    '
>>> str.center(10, '-')
'---abc----'
>>> 'he'.rjust(5) # 5자리를 오른쪽으로 정렬
'   he'
>>> 'he'.ljust(5) # 5자리를 왼쪽으로 정렬
'he   '
>>> 'he'.center(5) # 5자리를 가운데로 정렬
' he '
>>> '12'.zfill(5) # 문자열 '12'의 왼쪽을 0으로 채워 5자리를 만듬
'00012'
```

문자열 연습

다음 각 프로그램을 작성해 보자.

- 임의의 문자열에서 대문자가 나오는 횟수 세기 _
- 임의의 문자열에서 특정 문자의 집합에 속하는 문자(예: 모음)가 몇 번 나오는 지 세기 _
- 임의의 문자열에서 모든 대문자의 위치(인덱스)를 출력 _
- 여러 단어의 두문자로 이루어진 단어(acronym)를 만들어 대문자로 출력 : _
  (예: united nations --> UN, read the fine manual --> RTFM)
  - 입력된 여러 단어를 단어별로 잘라내 단어의 첫 글자를 대문자로 만든 후 계속 추가해 감
  - 힌트 _

- 힌트

```
s = input('Type words: ')
a = _______                    # acronym 담을 곳을 빈 문자열로 시작
for w in s._____________:      # s를 단어별로 잘라냄
    a += _____________         # 단어의 첫글자를 대문자로 만든 후 a에 추가
print(a)

# 다른 방법
b = ______
for w in s._________________: # s를 모두 대문자로 바꾼 후 단어별로 잘라냄
    b += _________             # 단어의 첫글자를 b에 추가
print(b)
```