

4-3. 텍스트 데이터를 문자열로 저장한다는 것 (2) 문자열 다루기

이제부터 텍스트 데이터를 다루기 위한 사전 작업으로 파이썬 문자열(string)을 살펴 볼텐데요. 자주 쓰이는 메소드들을 예제와 함께 차근 차근 알아가봅시다.

이스케이프 문자

```
s = 'I don't like Python!'
```

위와 같은 상황에서 텍스트를 파이썬 문자열로 만들려면 어떻게 해야 할까요? 물론 큰따옴표로 둘러싼 문자열로 만들면 되지만, 반드시 작은따옴표로 둘러싸여야만 하는 상황이라면 어떻게 해야 할까요?

문자열에 작은따옴표 (') 나 큰따옴표 (") , 줄 바꿈 등의 특수문자를 포함하고 싶은데, 그 문자가 특수문자가 아닌 것처럼 처리하고 싶을 때 이스케이프 문자를 사용할 수 있습니다. 이스케이프 문자는 `\[특정 문자]` 형태로, 직접 입력할 수 없는 일부 문자를 문자열에 포함시킬 수 있는 특수 문자인데요. 이 이스케이프 문자를 사용해서 한번 시도해봅시다.

이스케이프 문자	출력
\ '	홀따옴표 '
\ "	겹따옴표 "
\ t	탭
\ n	줄바꿈
\\	백슬래시 \

[Input]

#- 줄 바꿈

```
print("사회적\n거리두기")
print('-----')
```

#- 탭(tab)

```
print("사회적\t거리두기")
print('-----')
```

#- 작은따옴표 포함

```
print('오늘부터 \'사회적 거리두기\')
```

실행 ►

[Output]



원시 문자열

그렇다면, 이스케이프 문자를 무시하고 싶을 때는 어떻게 해야 할까요? 이때 사용하는 것이 **원시 문자열**(raw string) 입니다. 문자열을 시작하는 따옴표 앞에 **r** 을 붙이면 이스케이프 문자가 적용되지 않은 있는 그대로의 원시 문자열을 나타낼 수 있습니다.

설명이 조금 난해하죠? 아래 예시 코드에서 출력된 두 문장을 비교해 보면 금방 이해하실 수 있을 거예요. 👍

[Input]

#- 예제 코드2

```
print('Please don\'t touch it')  
print(r'Please don\'t touch it')
```



실행 ▶

[Output]

startswith, endswith

startswith

생산직은 **OB** 로, 사무직은 **OW** 로 시작되는 직원 ID가 저장된 데이터베이스가 있다고 해봅시다. 생산직에 속한 직원 ID만을 추려보고 싶을 때 **startswith** 를 사용하면 되는데요. 아래 코드로 한번 확인해 보겠습니다.

[Input]

```
EmployeeID = ['OB94382', 'OW34723', 'OB32308', 'OB83461',  
              'OB74830', 'OW37402', 'OW11235', 'OB82345']  
Production_Employee = [P for P in EmployeeID if P.startswith('OB')] # 'OB'로 시작하는 직원 ID를 다 찾아봅니다
```







Production_Employee


[Output]


endswith


실행 ▶


 linux.jpg


 markus-spisk...-unsplash.jpg


 matteo.jpeg


 metaflow.png


 NVIDIA.jpeg


 onedrive-illo3.jpg


 painting.jpg


 paolo-nicolell...-unsplash.jpg


 photo.jpeg


 photo.jpg


 Pytorch.png


 RF.jpg


 sckitlearn.jpg


 smartphone.png


 TDD-vs-BDD.jpg

 TDG-10-Oct-2.jpg

 TDG.png

 test.jpg

 tomasz-wozn...-unsplash.jpg

 web.jpg

위 폴더에 그림 파일들이 저장되어 있습니다. `.png` , `.jpg` , `.jpeg` 등 다양하죠?

여기서 `.png` 파일만 얻고 싶다면 어떻게 해야 할까요? `endswith()` 를 통해 해당 파일(들)을 찾을 수 있습니다. 저는 아래와 같이 실행해 보았습니다.

```
>>> import os
>>> photo = os.listdir('/Users/jooyoungson/Documents/photo')
>>> png = [png for png in photo if png.endswith('.png')]
>>> print(png)
metaflow.png
Pytorch.png
smartphone.png
TDG.png
```

여러분들도 각자의 사진 파일을 찾아 봅시다.

[Input]

```
import os
image_dir_path = os.getenv("HOME") + "/Pictures"
#- 각자의 사진이 보관된 디렉토리를 골라 주세요.
photo = os.listdir(image_dir_path)
png = [png for png in photo if png.endswith('.png')]
print(png)
```



실행 ▶

[Output]

공백 문자 처리 : trimming

컴퓨터의 스페이스바(space bar)처럼 일상에서 자주 쓰고 접하는 띄어쓰기! 지금부터 살펴 볼 **공백 문자**는 이 띄어쓰기로 표기되는 항목인데요. 문자열 작업에서는 다듬기 즉, **trimming**이라고 불립니다. 프로그래밍에서 인식하는 공백 문자는 어떤 것들이 있을까요? 한번 살펴봅시다.

여러가지 공백 문자

- 스페이스(space) : 한 칸 띄어쓰기
- 탭(tab) \t : 네 칸 띄어쓰기. 경우에 따라 두 칸 띄어쓰기로 표기되기도 합니다.
- 줄 바꿈(new line) : 줄 바꿈
- 라인 피드 (line feed) \n : 줄 바꿈을 엄밀히 말하면 라인 피드라고 합니다.
- 개행 복귀 (carriage return) \r : 커서를 맨 앞으로 이동시키는 것, 즉 커서를 원위치로 복귀(return)한다는 뜻입니다.

아래 코드를 참고해 공백 문자의 차이를 확인해 보세요.

[Input]

```
print("사회적 거리두기")
print('-----')
print("사회적\t거리두기")
print('-----')
print("사회적\n거리두기")
print('-----')
print("사회적22\r거리두기")
```



실행 ▶

[Output]

공백 문자 제거하기

파이썬에서는 `strip()` 메서드를 사용해 공백 문자를 처리합니다.

[Input]

```
#txt = "   공백 문자를 제거해 보아요.   "
txt = "   Strip white spaces.   "
print('{0} □ □ {1}'.format(txt, '2'))
print('-----')

#- 양쪽 공백 제거 : strip()
print('{0}'.format(txt.strip()))
print('-----')

#- 왼쪽 공백 제거 : lstrip()
print('{0}'.format(txt.lstrip()))
print('-----')

#- 오른쪽 공백 제거 : rstrip()
print('{0}'.format(txt.rstrip()))
```



실행 ▶

[Output]

대소문자 관련

알파벳으로 구성된 문자열에서 대소 문자를 쉽게 변경할 수 있는 방법이 있는데요. 한번 알아보까요?

- `upper()` : 모든 문자를 대문자로 변환합니다.

- `lower()` : 모든 문자를 소문자로 변환합니다.
- `capitalize()` : 첫 글자만 대문자로 변환합니다.

직접 실행해 보면서 어떻게 바뀌는지 확인해 봅시다.

[Input]

```
#- 모든 문자를 대문자로 변환 : upper()
txt = "I fell into AIFFEEL"
txt.upper()
print(txt)
```



실행 ▶

[Output]

[Input]

```
#- 모든 문자를 소문자로 변환 : lower()
txt.lower()
```



실행 ▶

[Output]

[Input]

```
#- 첫 글자만 대문자로 변환 : capitalize()
txt.capitalize()
```



실행 ▶

[Output]

isX

다음 소개해 드릴 메소드는 isX 형태의 메소드들인데요. 문자열의 구성에 따라 불린(boolean)의 값을 반환(return)해줍니다.

- `isupper()` : 문자열이 모두 **대문자**로만 되어 있으면 True, 그렇지 않으면 False를 반환
- `islower()` : 문자열이 모두 **소문자**로만 되어 있으면 True, 그렇지 않으면 False를 반환
- `istitle()` : 문자열의 **첫 글자만 대문자**로 되어 있으면 True, 그렇지 않으면 False를 반환
- `isalpha()` : 문자열이 모두 **알파벳 문자**로만 되어 있으면 True, 그렇지 않으면 False를 반환
- `isalnum()` : 문자열이 모두 **알파벳 문자**와 **숫자**로만 되어 있으면 True, 그렇지 않으면 False를 반환
- `isdecimal()` : 문자열이 모두 **숫자**로만 되어 있으면 True, 그렇지 않으면 False를 반환

[Input]

```
print("aiffel".isupper())  
print("aiffel".islower())  
print("PYTHON".istitle())  
print("python101".isalpha())  
print("python101".isalnum())  
print("101".isdecimal())
```



실행 ▶

[Output]

join()과 split()

`join()` 은 인자로 tuple, list, string 등 반복 가능한(iterable) 객체를 받는 메소드입니다. 이 `join()` 은 각각의 원소를 모아 하나의 문자열로 합쳐 주는데요. 일단 한번 실행해 봅시다.

[Input]

```
#- join()
stages = ['fundamentals', 'exploration', 'goingdeeper']
"□".join(stages)
```



실행 ▶

[Output]

`split()` 은 반대로 하나의 문자열을 구분자를 기준으로 나누어 줍니다. 명시적으로 구분자를 지정하지 않으면 쉼표 (,) 를 기준으로 나누어 줍니다.

자세히 보니 `split()` 은 리스트를 반환하고 `join()` 은 문자열을 반환한다는 차이가 보이네요. 기억해두면 좋겠죠?

[Input]

```
#- split()
'fundamentals,exploration,goingdeeper'.split(',')
```



실행 ▶

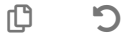
[Output]

replace()

- `replace()` : `replace(s1, s2)` 형태로 문자열 내 문자열 `s1` 을 `s2` 로 바꿉니다.

[Input]

```
sent = 'I can do it!'
sent.replace('I', 'You')
```



실행 ▶

[Output]

불변(immutable)의 문자열

mutable은 값이 변한다는 뜻이고, immutable은 반대로 값이 변하지 않는다는 의미이죠. 개념을 잠시 살펴 볼까요?

- 가변객체(mutable object)
 - 객체를 생성한 후 객체의 값을 수정할 수 있습니다.
 - 변수는 값이 수정된 같은 객체를 가리키게 됩니다.
 - e.g. `list` , `set` , `dict`
- 불변객체(immutable object)
 - 객체를 생성한 후 객체의 값을 수정할 수 없습니다.
 - 변수는 해당 값을 가진 다른 객체를 가리키게 됩니다.
 - e.g. `int` , `float` , `complex` , `bool` , `string` , `tuple` , `frozen set`

개념이 다소 난해해 한번에 이해하기 어렵지요? 위 개념은 구구절절한 설명 대신 직관적인 예시들이 포함된 참고 자료를 소개하고 넘어가도록 하겠습니다.
[하나](#), [둘](#)을 살펴 보시고 주변 동료들과 함께 토의해보세요!

잠깐, 쉬어가요. ☕

'I fell into AIFFE~~L~~'이라는 문장 속 알파벳을 전부 대문자로 바꾸고 싶습니다. 딱 떠오르는 것은 방금 배운 `upper()` 메소드를 사용하면 될 것 같은데요.

아래 두 개의 코드 블록을 비교해 보며, 지금까지 배운 메소드들을 왜 사용하는지 그리고 어떻게 사용해야 하는지를 각자 생각해 보는 시간을 가져 봅시다.

`id(sent)` 를 확인해보는 것도 도움이 될 것입니다.

[Input]

```
sent = 'I fell into AIFFEL'
print(sent)
print(id(sent))
sent.upper()
print(sent)
print(id(sent))
```



실행 ▶

[Output]

[Input]

```
sent = 'I fell into AIFFEL'
print(sent)
print(id(sent))
sent = sent.upper()
print(sent)
print(id(sent))
```





[Output]

실행 ▶