07-3 정규 표현식의 세계로2

▼ 전방탐색

▼ 긍정(positive) 전방 탐색 (?=..)

...에 해당하는 정규식과 매치되어야함. 조건이 통과되어도 문자열 소비X

```
p = re.compile('.+(?=:)')
m = p.sesarch("http://google.com")
print(m.group())
'http'

/* " : "에 해당하는 문자열이 정규식의 엔진에 의해 소비되지 않아 검색 결과에서는 : 이
제거된 후 돌려줌 */
```

▼ 부정(negative) 전방 탐색 (?!..)

..에 해당하는 정규식과 매치되지 않아야함. 조건이 통과되어도 문자열 소비X

제외하는 경우일 때 부정형 전방탐색을 쓰면 훨씬 깔끔해짐

▼ 문자열 바꾸기(sub)

sub메서드 → 정규식과 매치되는 부분을 다른 문자로 바꿀 수 있음

형식: ~.sub(첫번째 매개변수, 두번째 매개변수)

```
p = re.compile('(blue|white|red)')
p.sub('color', 'blue socks and red shoes') #color: 바꿀 문자열, blue socks ~: 대상 문자열
'color socks and color shoes'

-> 딱 한 번만 바꾸고 싶은 경우, 즉 바꾸기 횟수를 제한하고 싶은 경우: count
p.sub('color', 'blue socks and red shoes', count = 1)
'color socks and red shoes'
```

▼ sub 메서드를 사용할 때 참조 구문 사용하기(수정)

▼ sub 메서드의 매개변수로 함수 넣기

sub의 첫번째 매개변수로 함수가 사용될 경우, 해당 함수의 첫번째 매개변수에는 정규식과 매치된 match 객체가 입력됨

▼ Greedy vs Non-Greedy

Greedy: 탐욕스러운

정규식에서 Greedy 하다는 것은 매치할 수 있는 최대한의 문자열을 모두 소비한다는 것

ex code)

```
s = '<html><head><title>Title</title>'
len(s)
'32'
print(re.match('<.*>',s).span())
'(0,32)'
print(re.match('<.*>',s).group())
'<html><head><title>Title</title>'

# <.*> 정규식의 결과로 <html>을 돌려줄 것이라고 예상했지만 * 메타문자의 Greedy함 때문에
# 최대한의 문자열을 모두 소비함
```

Greedy함을 제한하고 <html>문자열까지만 소비하도록 하는 방법?

→ non-greedy 문자인 ? 을 사용

```
print(re.match('<.*?>',s).group())
'<html>'
```

➕ 추가)

▼ subn 메서드

sub 메서드와 기능은 똑같지만 결과를 돌려줄 때 튜플로 돌려줌

튜플의 첫번째 요소는 변경된 문자열이고, 두번째 요소는 바꾸기가 발생한 횟수