07-3 정규 표현식의 세계로1

문자열 소비가 없는(zero-width assertions) 메타문자

▼ |

|메타문자는 == or

A|B == A or B

▼ ^

^메타문자: 문자열의 맨처음과 일치함을 의미

re.MULTILINE과 함께 사용할 경우에는 여러 줄의 문자열일 때ㅐ 각 줄의 처음과 일치하게 됨

▼ $

$메타문자는 ^메타문자와 반대의 경우→ 문자열의 가장 끝과 매치함을 의미

▼ \A

\A: 문자열의 처음과 매치

^메타문자와 동일하지만 re.MULTILINE과 사용할 경우 다르게 해석됨

- re.MULTILINE과 사용할 경우

-^메타문자: 각 라인에 있는 문자열의 처음과 매치

-\A: 줄 상관없이 전체 문자열의 처음하고만 매치

▼ \Z

문자열의 끝과 매치, $메타문자와 동일하지만 re.MULTILINE과 사용할 경우 달라짐

- re.MULTILINE과 사용할 경우

-$메타문자: 각 라인에 있는 문자열의 마지막과 매치

-\Z: 줄 상관없이 전체 문자열의 마지막과 매치

▼ \b

\b: 단어 구분자(Word boundary)

보통 단어는 whitespace에 의해 구분됨

```
p = re.compile(r'\bclass\b')
print(p.search('no class at all'))
'<re.Match object; span=(3,8), match='class'>'

#여기서 \bclass\b는 앞 뒤로 whitespace로 구분된 class라는 단어와 매치됨을 의미
```

⚠️ \b 메타문자를 사용할 때 주의할 점

→ \b는 파이썬 리터럴 규칙에 의하면 백스페이스를 의미하므로 백스페이스가 아닌 단어 구분자임을 나타내기 위해서는 r'\bclass\b'처럼 Raw string임을 알려주는 기호 r을 반드시 붙여야함

▼ \B

\b메타문자와 반대의 경우 → 즉 whitespace로 구분된 단어가 아닌 경우에만 매치

```
p = re.compile(r'\Bclass\B')
print(p.search('no class at all'))
'None'

print(p.search('the declassified algorithm'))
'<re.Match object; span=(6,11), match='class'>'

print(p.search('one subclass is'))
'None'
#이렇게 한쪽이라도 공백이 포함되어 있으면 매치되지 않고 None을 돌려줌
```

▼ 그루핑(Grouping)

문자열이 계속해서 반복되는지 조사하는 정규식을 작성하고 싶을 때 or 매치된 문자열 중에서 특정 부분의 문자열만 뽑아낼 때 사용

그룹을 만들어 주는 메타문자: ( )

▼ 반복되는 문자열 찾을 때 예시

```
p = re.compile('(ABC)+')
m = p.search('ABCABCABC OK?')
print(m)
'<re.Match object; span=(0,9), match='ABCABCABC'>'
print(m.group(0))
'ABCABCABC'
```

▼ 특정 부분의 문자열 뽑아낼 때 예시

```
#이름과 전화번호 형태의 문자열을 찾는 정규식에서 이름 부분만 뽑아내려 할 때
p = re.compile(r'(\w+)\s+\d+[-]\d+[-]\d+')
m = p.search('park 010-1234-5678')
print(m.group(1)) #첫번째 그룹
'park'

p = re.compile(r'(\w+)\s+(\d+[-]\d+[-]\d+)')
m = p.search('park 010-1234-5678')
print(m.group(2)) #두번째 그룹
'010-1234-5678'

/* 이름에 해당하는 \w+ 부분을 그루핑해서 (\w+)로 만들면 match객체의 group(인덱스)
메서드를 사용하여 그루핑된 부분의 문자열만 뽑아낼 수 있음 */
```

- group 메서드의 인덱스의 의미 표

group	설명
group(0)	매치된 전체 문자열
group(1)	첫번째 그룹에 해당하는 문자열
group(2)	두번째 그룹에 해당하는 문자열
group(3)	n번째 그룹에 해당하는 문자열

r’(\w+)\s((\d+)[-]\d+[-]\d+))’ 처럼 중첩되게 그룹을 사용할 수 있음

그룹이 중첩되어 있는 경우, 바깥→ 안쪽으로 들어갈 수록 인덱스 증가함

▼ 그루핑된 문자열 재참조(\1)

한 번 그루핑한 문자열 재참조(Backreferences)할 수 있음

```
p = re.compile(r'(\b\w+)\s+\1')
p.search('Paris in the the spring').group()
'the the'

/* 정규식 (\b\w+)\s+\1 은 '(그룹) + " " + 그룹과 동일한 단어'와 매치됨을 의미
즉 2개의 동일한 단어를 연속적으로 사용해야만 매치됨
이것을 가능하게 해준 것이 재참조 메타문자 \1 (\1은 정규식의 그룹 중 첫번째 그룹을 가리킴) */
```

▼ 그루핑된 문자열에 이름 붙이기

정규식 안에 그룹이 매우 많아지면 혼란스러워짐 → 그룹을 인덱스가 아닌 이름(Named Groups)으로 참조

```
(?P<name>\w+)\s+((\d+[-]\d+[-]\d+)
# 이름과 전화번호를 추출하는 정규식 중 (\w+)그룹에 name이라는 이름을 붙임
# 여기 사용한 (?...) 표현은 정규 표현식의 확장 구문
```

➕ 추가로 그룹이름을 사용하면 정규식 안에서 재참조도 가능함

재참조할 때에는 (?P=그룹이름) 이라는 확장 구문 사용