제7강 Regular Expressions

학습 목차

- 정규식이란?
- 기본 사용법
- 각종 매칭 문법
- 문자 클래스

Regular Expressions

- 텍스트의 패턴을 나타내는 규칙 정의

- 전화번호 010-333-4444

```
\d\d\d-\d\d\d\d-\d\d\d\d
\d{3}-\d{3}-\d{4}
```

- 왜 쓰는가? 다양한 조건의 패턴을 쉽게 검색, 필요에 따라 교체

기본 사용 – 문자열 검색

```
>> import re
>> hello_re = re.compile(r'hello’)
>> hello_re.search('hello world')
<re.Match object; span=(0, 5), match='hello’>
>> print(hello_re.search('test'))
None
```

기본 사용 – 숫자 검색

```
phone_re = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
mo = phone_re.search('My number is 010-555-4242.')
print('Phone number found: ' + mo.group())
```

```
Phone number found: 010-555-4242
```

그룹화 괄호 사용

```
>>> phone_re = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phone_re.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242’
>>> mo.groups()
('415', '555-4242')
>>> area_code, main_number = mo.groups()
>>> print(area_code)
415
>>> print(main_number)
555-4242
```

pipe ¦ - 여러개 그룹을 동시에 매칭

```
idol_re = re.compile(r'로제|제니')
mo = idol_re.search('로제와 제니')
mo.group()
>> '로제'
mo = idol_re.search('제니와 로제')
mo.group()
>> '제니'
idol_re = re.compile(r'블랙핑크 (지수|로제|제니)')
mo = idol_re.search('나의 우상은 블랙핑크 지수이다.')
mo.group()
>> '블랙핑크 지수'
mo = idol_re.search('나의 우상은 블랙핑크 로제이다.')
mo.group()
>> '블랙핑크 로제'
mo.group(1)
>> '로제'
```

? – 없거나 또는 하나 있는 요소에 대한 매칭

```
>>> bat_re = re.compile(r'Bat(wo)?man')
>>> mo1 = bat_re.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = bat_re.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman’.
```

```
phone_re = re.compile(r'(\d\d\d-)?(\d\d\d\d-\d\d\d\d)')
mo = phone_re.search('my number is 010-8041-0114.')
mo.group()
mo = phone_re.search('my number is 8041-0114.')
mo.group()
```

* - 없거나 또는 여러 번 반복되는 요소에 대한 매칭

+ - 한번 또는 여러 번 반복되는 요소에 대한 매칭

```
>>> bat_re = re.compile(r'Bat(wo)*man')
>>> mo1 = bat_re.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = bat_re.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
>>> mo3 = bat_re.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'

>>> bat_re = re.compile(r'Bat(wo)+man')
>>> mo1 = bat_re.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'
>>> mo2 = bat_re.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'
```

{x,y} - 특정 횟수만큼 반복되는 요소에 대한 매칭

```
>>> ha_re = re.compile(r'(Ha){3}')
>>> mo1 = ha_re.search('HaHaHa')
>>> mo1.group()
'HaHaHa'
>>> mo2 = ha_re.search('Ha')
>>> mo2 == None
True
```

```
(Ha){3,5}
((Ha)(Ha)(Ha))¦((Ha)(Ha)(Ha)(Ha))¦((Ha)(Ha)(Ha)(Ha)(Ha))
```

Greedy vs Non-Greedy

- Greedy : 가장 긴 것에 매칭 (default)
- Non-Greedy : 먼저 발견된 것에 매칭 (물음표 추가)

```
>>> greedy_re = re.compile(r'(Ha){3,5}')
>>> mo1 = greedy_re.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'
>>> nongreedy_re = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedy_re.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

findall() – 매칭되는 모든 문자열을 찾음.

```
>>> phone_re = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups
>>> phone_re.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000 ' ]
```

```
>>> phone_re = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups
>>> phone_re.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

Character Class

문자	뜻
\d	0부터 9까지 숫자
\D	숫자가 아닌 모든 문자
\w	글자, 숫자, 그리고 _
\W	\w 이 아닌 모든 문자
\s	공백, 탭, 개행문자(₩n) – white space
\S	\s 이 아닌 모든 문자

[] - 문자 클래스 사용자 지정

(a|b|c|d|e|f|g)
[abcdefg]
[a-g]
[aeiouAEIOU] # 모음
[^aeiouAEIOU] # 자음
[a-zA-Z0-9]

^ - 문자열의 시작을 매칭, $ - 문자열의 마지막을 매칭

```
p = re.compile(r'^hello')
p.search('hello world')
p.search('She say hello')

p = re.compile(r'\d$')
p.search('you number is 42')
p.search('you number is 42 and my number is ...')

p = re.compile(r'^\d+$')
p.search('1234')
p.search('number is 1234')
```

. new line 을 제외한 모든 문자와 매칭

```
p = re.compile('.at')
p.findall('The cat in the hat sat on the flat mat.')

p = re.compile(r'.치')
p.findall('참치 꽁치 쥐치 가물치')
```

.* 모든 문자열에 매칭

```
p = re.compile(r'성:(.*)이름:(.*)')
mo = p.search('성: 이 이름: 대현')
mo.group()
mo = p.search('성:      이름:      ')
mo.group()
mo.groups()
mo = p.search('성:이름:')
mo.group()
mo.groups()
```

.* 에 대한 Greedy VS Non-Greedy

```
p = re.compile(r'<.*>')
p.search('<이대현> 님 입장하셨습니다.>')

p = re.compile(r'<.*?>')
p.search('<이대현> 님 입장하셨습니다.>')
```

new line /n 매칭 – DOTALL

```
text = '''This
is
multiple
lines
'"


p = re.compile('.*')
p.search(text)
p = re.compile('.*', re.DOTALL)
p.search(text)
```

대소문자 무시 - I

```
p = re.compile(r'hello', re.I)
p.search('HELLO WORLD').group()
p.search('Hello World').group()
```

sub() – 문자열 교체

```
p = re.compile(r'<(\D)\D+>')
p.search('제1회 복권 당첨자는 <이대현>입니다.')
p.sub('***', '제1회 복권 당첨자는 <이대현>입니다.')
p.sub(r'\1**', '제1회 복권 당첨자는 <이대현>입니다.')
```

\1 \2 \3 : group

복잡한 정규식의 쉬운 표시 방법 - VERBOSE

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)\d{4}
(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    \d{3}                         # first 3 digits
    (\s|-|\.)                     # separator
    \d{4}                         # last 4 digits
    (\s*(ext|x|ext.)\s*\d{2,5})?  # extension
    )''', re.VERBOSE)
```

옵션의 동시 선택

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL | re.VERBOSE)
```

정리

?

*

+

{n,m}

^spam

spam$

.

\d, \w, \s

[abc]

[^abc]