제7강 Regular Expressions

학습 목차

- 정규식이란?
- 기본 사용법
- PyCharm 상에서 활용
- 다양한 정규식 표현

Regular Expressions

- **텍스트의 패턴을 나타내는 규칙**

- **전화번호 010-333-4444 의 패턴 규칙은?**

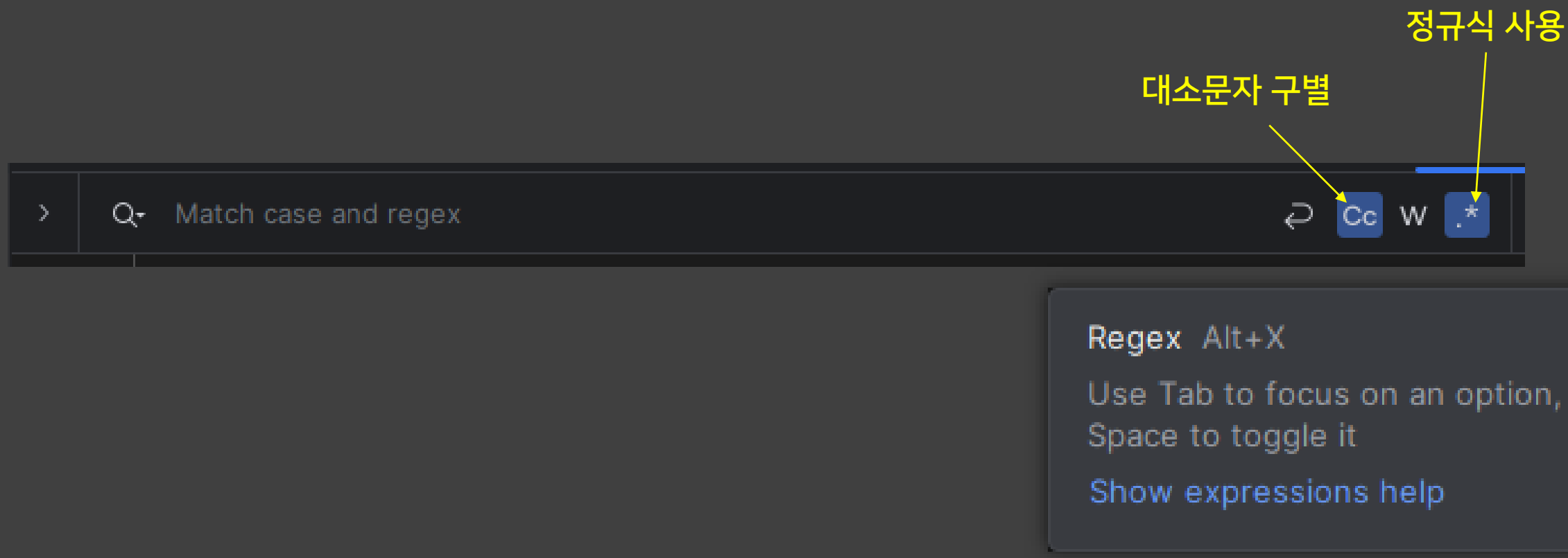
정규식 언제 쓰는가?

- 데이터 검증 : 이메일 주소, 전화번호, 비밀번호 형식 검증
- 웹 스크래핑 : 웹 페이지에서 특정한 패턴을 갖는 정보 추출
- 워드프로세서: 특정 문자열 패턴을 찾거나 치환
- 파일 시스템 검색: 특정 패턴을 갖는 파일들을 검색.
- …

기본 사용 – 문자열 검색

```
>>> import re
>>> text = 'Hello, world and world'
>>> p = re.compile('Hello')
>>> mo = p.search(text)
>>> mo
<re.Match object; span=(0, 5), match='Hello'>
>>> mo.group()
'Hello'
>>> mo.span()
(0, 5)
>>> p = re.compile('hello')
>>> mo = p.search(text)
>>> mo
>>> mo
>>> p = re.compile('world')
>>> p.search(text)
<re.Match object; span=(7, 12), match='world'>
```

Pycharm 정규식 검색 Ctrl + F



A | B – 파이프, 둘 다 매칭

- 아빠|엄마

- 수|우|미|양|가

여러 개 매칭, 클래스 지정 [ ]

- **[Ww]orld**

- **숫자**
  - [0|1|2|3|4|5|6|7|8|9]
  - [0123456789]
  - [0-9]
  - \d

- **문자**
  - [a-z]
  - [aeiouAEIOU]
  - \D # **숫자 아닌 모든 문자**

16진수는?

주요 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

모든 한글 문자

- [\uAC00-\uD7A3]

초성, 중성, 종성은?

{x, y} 반복

- \d\d\d
- \d{3}
- \d{3,4}
- \d{3,}

+ 한 개 이상

- 0x[0-9A-Fa-f]+ # 16 진수
- [a-zA-Z]+at
- [\uAC00-\uD7A3]+치

? 없거나 또는 하나 있는 요소에 대한 매칭

- \d{4}-?\d{4}

그룹화 괄호 사용

- 괄호를 통해 그룹화시켜서 매칭되는 일부분을 지정할 수 있음.

```
>>> p = re.compile(r'(\d{4})-(\d{4})')
>>> mo = p.search(test_string)
>>> mo.group()
'1234-5678'
>>> mo.groups()
('1234', '5678')
>>> mo.group(1)
'1234'
>>> mo.group(2)
'5678'
>>> mo.group(0)
'1234-5678'
>>> station_number, user_number = mo.groups()
>>> station_number
'1234'
>>> user_number
'5678'
```

```
>>> p = re.compile(r'((\d{4})-(\d{4}))')
>>> mo = p.search(test_string)
>>> mo.group()
'1234-5678'
>>> mo.groups()
('1234-5678', '1234', '5678')
>>> mo.group(0)
'1234-5678'
>>> mo.group(1)
'1234-5678'
>>> mo.group(2)
'1234'
>>> mo.group(3)
'5678'
```

find_all – 매칭되는 모든 문자열을 검색

```
>>> p = re.compile(r'아빠|엄마')
>>> p.findall(test_string)
['아빠', '엄마']
>>> p = re.compile(r'[a-zA-Z]+at')
>>> p.findall(test_string)
['cat', 'hat', 'sat', 'flat', 'mat']
>>> p = re.compile(r'([a-zA-Z]+)(at)')
>>> p.findall(test_string)
[('c', 'at'), ('h', 'at'), ('s', 'at'), ('fl', 'at'), ('m', 'at')]
```

Greedy vs Non-Greedy

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

```
>>> test_string = 'HaHaHaHaHa'
>>> p = re.compile(r'(Ha){3,5}')
>>> p.search(test_string)
<re.Match object; span=(0, 10), match='HaHaHaHaHa'>
>>> p = re.compile(r'(Ha){3,5}?')
>>> p.search(test_string)
<re.Match object; span=(0, 6), match='HaHaHa'>
```

^ - 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')
```

.* 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()
```

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)
```