파이썬 정규표현식

Regular Expression

정규표현식?

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어
- 많은 텍스트 편집기와 프로그래밍 언어에서 문자열의 검색과 치환을 위해 널리 사용

I watch three climb before it's my turn. It's a tough one. The guy before me tries twice. He falls twice. After the last one, he comes down. He's finished for the day. It's my turn. My buddy says "good luck!" to me. I noticed a bit of a problem. There's an outcrop on this one. It's about halfway up the wall. It's not a

노란 부분은 아래 정규식을 사용하여 매칭한 것

```
(?:\.) {2,}(?=[A-Z])
```

파이썬 정규표현식
: 사용 방법

- 모듈 임포트
  - `import re`
- 패턴 컴파일
  - 예) `p = re.compile(r'정규표현식')`
- 패턴 객체가 가지고 있는 메서드로 작업을 수행

파이썬 정규표현식
: 사용 방법

- 첫 번째 방법: 패턴 컴파일 후 매칭

```
>>> import re
>>> p = re.compile(r'[a-z]+') # 패턴 컴파일
>>> p.match("python") # 패턴 객체의 메서드 수행
<_sre.SRE_Match object; span=(0, 6), match='python'>
```

- 두 번째 방법: 축약형

```
>>> source = "Life is too short, You need python"
>>> re.match(r"[a-z]+", source)
>>> re.match(r"[A-Za-z]+", source) # 컴파일과 매치를 한번에 한다
<_sre.SRE_Match object; span=(0, 4), match='Life'>
```

파이썬 정규표현식
: 메타 문자

```
. ^ $ + ? { } [ ] \ | ( )
```

- 모듈 임포트
  - `import re`
- 패턴 컴파일
  - 예) `p = re.compile(r'정규표현식')`
- 패턴 객체가 가지고 있는 메서드로 작업을 수행

메타 문자	설명
[ ]	문자 클래스
.	\n을 제외한 모든 문자와 매치 (점 하나는 글자 하나)
*	0회 이상 반복 (없어도 상관 없음)
+	1회 이상 반복 (무조건 한번 이상 등장해야 함)
{m, n}	m회 이상 n회 이하
|	or 조건식을 의미
^	문자열의 시작 의미
$	문자열의 끝을 의미
?	0회 이상 1회 이하
\	이스케이프, 또는 메타 문자를 일반 문자로 인식하게 한다 (예: \\)
( )	그룹핑, 추출할 패턴을 지정한다.

파이썬 정규표현식
: 메타 문자 [], dot(.)

- [] 안에는 무엇이든 들어갈 수 있다
  - 단, 대소문자를 구분한다
- 내용은 or 로 연결된다
  - 하이픈(-)으로 연결할 수 있다
  - [a-z]는 a부터 z까지를 의미
- [abc] 라면 'a, b, c' 중에서 한 개의 문자와 매칭된다

- 도트 하나는 문자 하나를 의미

- 문자는 숫자(0-9)나 특수문자 (!@#$%^& 등)를 포함한다

파이썬 정규표현식
: 축약 문자 클래스

보통 알파벳 표현을 위해 [a-zA-Z]를 사용
숫자 표현을 위해 [0-9] 사용
\b는 Boundary를 의미

자주 쓰는 조합을 한 글자로 줄여 제공

숫자 (0, 1, 2, …)	\d	\w	\S	\D
문자 (a, b, c, …)				
특수문자 (!, %, $, …)		\W		
화이트 스페이스 (\n, \t, \b)	\s			

파이썬 정규표현식
: 축약 문자 클래스와 원래 표현식

원래 표현식	축약 표현	설명	사용처
[0-9]	\d	숫자를 찾는다	숫자
[^0-9]	\D	숫자가 아닌 것을 찾는다	텍스트 + 특수문자 + 화이트스페이스
[\t\n\r\f\v]	\s	whitespace 문자인 것을 찾는다	스페이스, TAB, 개행(new line)
[^ \t\n\r\f\v]	\S	whitespace 문자가 아닌 것을 찾는다	텍스트 + 특수문자 + 숫자
[a-zA-Z0-9]	\w	문자+숫자인 것을 찾는다. (특수문자는 제외. 단, 언더스코어 포함)	텍스트 + 숫자
[^a-zA-Z0-9]	\W	문자+숫자가 아닌 것을 찾는다.	특수문자 + 공백

파이썬 정규표현식
: 반복 - *, +

- * : 바로 앞 문자가 0번 이상 반복
- + : 바로 앞 문자가 1번 이상 반복(1번은 반드시 출현해야 함)

```
>>> import re
>>> source = "Life is too short, you need Python"
>>> re.findall('\w+', source)
['Life', 'is', 'too', 'short', 'you', 'need', 'Python']
```

파이썬 정규표현식
: 반복 - {}

- 바로 앞 문자의 반복 횟수를 지정한다

표현식	설명
ca{2}t	A가 2회 반복
ca{2, 5}t	A가 2~5회 반복
ca{0,}t	A가 0회 이상(*)
ca{0, 1}t	A가 0~1회 반복
ca{,3}t	A가 0~3회 반복

- {0, 1}은 ?로 대신할 수 있다
  - (있어도 되고 없어도 됨)

```
import re
source = "ct cat caat caaat caaaat"
m1 = re.findall("ca{2}t", source)
m2 = re.findall("ca{2,5}t", source)
m3 = re.findall("ca{0,}t", source)
m4 = re.findall("ca{0,1}t", source)
m5 = re.findall("ca{,3}t", source)
print("m1 : ", m1)
print("m2 : ", m2)
print("m3 : ", m3)
print("m4 : ", m4)
print("m5 : ", m5)

m1 :  ['caat']
m2 :  ['caat', 'caaat', 'caaaat']
m3 :  ['ct', 'cat', 'caat', 'caaat', 'caaaat']
m4 :  ['ct', 'cat']
m5 :  ['ct', 'cat', 'caat', 'caaat']
```

파이썬 정규표현식
: 주요 메서드

메서드	설명
match	문자열의 처음부터 정규식과 매치되는지 조사
search	문자열 전체를 검색하여 정규식과 매치되는지 조사
findall	정규식과 매치되는 모든 문자열(substring)을 리스트로 반환
split	패턴으로 자르기
sub	패턴을 대체하기

파이썬 정규표현식
: match vs search

- match는 처음이 일치하지 않으면 None을 반환
- search는 처음이 일치하지 않더라도 전체를 수색하여 반환

```
>>> import re
>>> source = "Life is too short, you need Python"
>>> re.match("Python", source)
>>> re.search("Python", source)
<_sre.SRE_Match object; span=(28, 34), match='Python'>
>>> re.match("Life", source)
<_sre.SRE_Match object; span=(0, 4), match='Life'>
>>> re.search("Life", source)
<_sre.SRE_Match object; span=(0, 4), match='Life'>
```

파이썬 정규표현식
: findall vs finditer

- findall은 매치되는 모든 문자열을 리스트로 반환
- finditer는 매치되는 모든 문자열을 iterator로 반환
  - Iterator의 값을 불러오려면 반복문을 이용하여 읽어야 한다

```
>>> import re
>>> p = re.compile('[a-z]+')
>>> m = p.findall("Life is too short, you need Python")
>>> m
['ife', 'is', 'too', 'short', 'you', 'need', 'ython']
>>>
>>> iter = p.finditer("Life is too short, you need Python")
>>> iter
<callable_iterator object at 0x10cc5a898>
>>> for x in iter:
...     print(x)
...
<_sre.SRE_Match object; span=(1, 4), match='ife'>
<_sre.SRE_Match object; span=(5, 7), match='is'>
<_sre.SRE_Match object; span=(8, 11), match='too'>
<_sre.SRE_Match object; span=(12, 17), match='short'>
<_sre.SRE_Match object; span=(19, 22), match='you'>
<_sre.SRE_Match object; span=(23, 27), match='need'>
<_sre.SRE_Match object; span=(29, 34), match='ython'>
```