정규표현식

정규표현식 메타문자

- 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식언어
  - 많은 프로그래밍언어들이 문자열의 검색과 치환을 위해 정규표현식을 지원하고 있음
- 반복 메타문자

메타문자	의미	예
*	0회 이상 반복	ab*c : ac, abc, abbc…..
+	1회 이상 반복	ab+c : abc, abbc, abbbc…..
?	0회 또는 1회	ab?c: ac, abc
{m}	m회 반복	ab{5} : abbbbb
{m,n}	m회부터 n회까지	ab{2,4}c : abbc, abbbc, abbbbc

정규표현식 메타문자

- 매칭 메타 문자

메타문자	의미	예
.	줄바꿈을 제외한 모든 문자와 매치. re.DOTALL 모드를 사용하면 줄바꿈 문자도 매치가능	a.c : aac, abc, acc, adc,.....
^	-. 문자열의 시작과 매치. re.MULTILINE 모드에서는 각 라인의 시작과 매치 -. [] 기호 안에서는 반대의 문자열을 나타냄	^abc : "abc def xyz"(0) "xyz abc"(x) a[^a]c : adc,abc, aec(0), aac(x)
$	-. 문자열의 마자믹과 매치 -. [] 기호 안에서는 순수한 $문자와 매치됨	^와 반대
[]	문자의 집합을 나타냄 [a-c], [a-zA-Z0-9]	c[abc]t : cat,cbt,cct와 매치 c[a-c]t와 동일표현
|	a|b는 a 또는 b의 의미	ca|bt : cat, cbt
()	정규식을 그룹으로 묶는다	

정규표현식 메타문자

- 이스케이프 기호
  - ₩₩ : 역슬래쉬 문자 자체를 의미
  - ₩d : 숫자와 매치, [0-9]
  - ₩D : 숫자가 아닌것과 매치, [^0-9]
  - ₩s : whitespace 문자와 매치, [₩t₩n₩r₩f₩v]
  - ₩S : whitespace 문자가 아닌것과 매치 [^ ₩t₩n₩r₩f₩v]
  - ₩w : 문자+숫자와 매치, [a-zA-Z0-9]
  - ₩W : alphanumeric이 아닌 문자와 매치, [^a-zA-Z0-9]
  - ₩b : 단어의 경계를 나타냄.
  - ₩B : ₩b의 반대로 단어의 경계가 아님을 나타냄

정규표현식

- re (regular expression)
  - 정규표현식을 지원하는 모듈

```
import re
p = re.compile('ab*')
p = re.compile('ab*', re.IGNORECASE)
p = re.compile(r'₩section')
p = re.compile(r'₩₩section')
```

- Regular Expression Objects
  - match() : 문자열의 처음부터 정규식과 매치되는지 조사함
  - search() : 문자열 전체를 검색하여 정규식과 매치되는지 조사함
  - findall() : 정규식과 매치되는 모든 문자열을 리스트로 리턴함
  - finditer() : 정규식과 매치되는 모든 문자열을 iterator 객체로 리턴하

```
import re
p = re.compile('ab*')
Result = p.match('abbbb')
```

```
import re
p = re.match('ab*','abbbb')
```

정규식 객체

- Search(string[, pos,[,endpos]]) 의 사용형식
  - String의 전역에서 일치하는 부분을 찾아 match_object를 반환/ None

```
import re
pattern = re.compile("d")
result = pattern.search("dog")
result = pattern.search("dog",1)
```

```
<raw mode>
re.search("₩₩₩₩", "C:₩₩test")
re.search(r"₩₩", "C:₩₩test")
```

- match(string[, pos,[,endpos]]) 의 사용형식
  - String의 시작부분에서 일치하는 부분을 찾아 match_object를 반환 / None
  - 정확하게 일치해야 찾아 줌

```
import re
pattern = re.compile("o")
result = pattern.match("dog")
result = pattern.match("dog",1)
```

정규식 객체

- 플래그
  - I, IGNORECASE
    - 대소문자 구분하지 않고 매치
  - M, MULTILINE
    - ^가 문자열의 맨 처음, 각 라인의 맨 처음과 매치됨
    - $는 문자열의 맨 끝, 각 라인의 맨 끝과 매치
  - S, DOTALL
    - .을 줄바꾸기 문자 ₩n도 매치하게 함
  - X, VERBOSE
    - 정규식 안의 공백은 무시됨
    - 정규식 안에 #문자를 쓰면 이후 모든 문자는 무시됨

정규식 객체

- group
  - 추출된 값을 보고 싶을 때 사용하는 메소드
  - group 인덱스는 1부터 시작함

```
import re
pattern = re.compile(“o")
result = pattern.search("dog")
result.group()
```

```
import re
str="  abc  1234  xyz  "
pattern = re.compile("₩s*[a-zA-Z]+₩s+(₩d+)+₩s+([a-zA-Z]+)₩s*");
result = pattern.match(str)
print(result.group(1))
print(result.group(2))
```

정규식 객체

- fullmatch(string[, pos,[,endpos]]) 의 사용형식
  - 전체 스트링이 정규식에 일치하면 match_object 반환

```
pattern = re.compile("o[gh]")
result = pattern.fullmatch("dog")
result = pattern.fullmatch("ogre")
result = pattern.fullmatch("doggie",1,3)
```

- split(string, matxsplit)
  - 정규식을 만족하는 부분에서 분할된 리스트 반환

```
pattern = re.compile("₩W+")
result = pattern.split('words, words, words.')
result = pattern.split('words, words, words.',1)
```

```
pattern = re.compile("x*")
result = pattern.split('axbc')
```

- sub(변환문자, 문자열)

```
result = re.sub(r'₩W','','a:b:c, d.')
```

정규식 객체

- 탐욕적인 매칭

```
str = '<a href="index.html">HERE</a><font size="10">'
result = re.search(r'href="(.*)">', str)
print(result.group(1))
```

- 최소매칭

메타문자	의미
*?	*와 같으나 문자열을 최소로 매치
+?	+와 같으나 문자열을 최소로 매치
??	?와 같으나 문자열을 최소로 매치
{m,n}?	{m,n}와 같으나 문자열을 최소로 매치

```
result = re.search(r'href="(.*?)">', str)
```