## [정규식 총정리](http://blog.bagesoft.com/178)

파일이나 문자열 내에 포함되어 있는 특별한 패턴(또는 특별한 조건을 만족하는 문자열)을 검색하기 위해 미리 정의된 다양한 특수 문자들의 조합을 정규식(regular expression)이라 한다. 정규식에서의 특수 문자(special character)는 다음과 같다.   
  
  
(1) ^ (caret) : 라인의 처음이나 문자열의 처음을 표시   
  
예 : ^aaa (문자열의 처음에 aaa를 포함하면 참, 그렇지 않으면 거짓)   
  
(2) $ (dollar) : 라인의 끝이나 문자열의 끝을 표시   
  
예 : aaa$ (문자열의 끝에 aaa를 포함하면 참, 그렇지 않으면 거짓)   
  
(3) . (period) : 임의의 한 문자를 표시   
  
예 : ^a.c (문자열의 처음에 abc, adc, aZc 등은 참, aa 는 거짓)   
  
a..b$ (문자열의 끝에 aaab, abbb, azzb 등을 포함하면 참)   
  
(4) [] (bracket) : 문자의 집합이나 범위를 나타냄, 두 문자 사이의 "-"는 범위를 나타냄   
  
[]내에서 "^"이 선행되면 not을 나타냄   
  
이외에도 "문자클래스"를 포함하는 [:문자클래스:]의 형태가 있다.   
  
여기에서 "문자클래스"에는 alpha, blank, cntrl, digit, graph, lower,   
  
print, space, uppper, xdigit가 있다.   
  
이에 대한 자세한 내용은 C언어의 를 참조하면 된다.   
  
예를 들어 [:digit:]는 [0-9]와 [:alpha:]는 [A-Za-z]와 동일하다.   
  
이외에 [::]는 어떤 단어(숫자, 알파벳, '\_'로 구성됨)의 시작과 끝   
  
을 나타낸다.   
  
예 : [abc] (a, b, c 중 어떤 문자, "[a-c]."과 동일)   
  
[Yy] (Y 또는 y)   
  
[A-Za-z0-9] (모든 알파벳과 숫자)   
  
[-A-Z]. ("-"(hyphen)과 모든 대문자)   
  
[^a-z] (소문자 이외의 문자)   
  
[^0-9] (숫자 이외의 문자)   
  
[[:digit:]] ([0-9]와 동일)   
  
(5) {} (brace) : {} 내의 숫자는 직전의 선행문자가 나타나는 횟수 또는 범위를 나타냄   
  
예 : a{3} ('a'의 3번 반복인 aaa만 해당됨)   
  
a{3,} ('a'가 3번 이상 반복인 aaa, aaaa, aaaa, ... 등을 나타냄)   
  
a{3,5} (aaa, aaaa, aaaaa 만 해당됨)   
  
ab{2,3} (abb와 abbb 만 해당됨)   
  
[0-9]{2} (두 자리 숫자)   
  
doc[7-9]{2} (doc77, doc87, doc97 등이 해당)   
  
[^Zz]{5} (Z와 z를 포함하지 않는 5개의 문자열, abcde, ttttt 등이 해당)   
  
.{3,4}er ('er'앞에 세 개 또는 네 개의 문자를 포함하는 문자열이므로 Peter, mother 등이 해당)   
  
(6) \* (asterisk) : "\*" 직전의 선행문자가 0번 또는 여러번 나타나는 문자열   
  
예 : ab\*c ('b'를 0번 또는 여러번 포함하므로 ac, ackdddd, abc, abbc, abbbbbbbc 등)   
  
\* (선행문자가 없는 경우이므로 임의의 문자열 및 공백 문자열도 해당됨)   
  
.\* (선행문자가 "."이므로 하나 이상의 문자를 포함하는 문자열, 공백 문자열은 안됨)   
  
ab\* ('b'를 0번 또는 여러번 포함하므로 a, accc, abb, abbbbbbb 등)   
  
a\* ('a'를 0번 또는 여러번 포함하므로 k, kdd, sdfrrt, a, aaaa, abb, 공백문자열 등)   
  
doc[7-9]\* (doc7, doc777, doc778989, doc 등이 해당)   
  
[A-Z].\* (대문자로만 이루어진 문자열)   
  
like.\* (직전의 선행문자가 '.'이므로 like에 0 또는 하나 이상의 문자가 추가된 문자열이 됨, like, likely, liker, likelihood 등)   
  
(7) + (asterisk) : "+" 직전의 선행문자가 1번 이상 나타나는 문자열   
  
예 : ab+c ('b'를 1번 또는 여러번 포함하므로 abc, abckdddd, abbc, abbbbbbbc 등, ac는 안됨)   
  
ab+ ('b'를 1번 또는 여러번 포함하므로 ab, abccc, abb, abbbbbbb 등)   
  
like.+ (직전의 선행문자가 '.'이므로 like에 하나 이상의 문자가 추가된 문자열이 됨, likely, liker, likelihood 등, 그러나 like는 해당안됨)   
  
[A-Z]+ (대문자로만 이루어진 문자열)   
  
(8) ? (asterisk) : "?" 직전의 선행문자가 0번 또는 1번 나타나는 문자열   
  
예 : ab?c ('b'를 0번 또는 1번 포함하므로 abc, abcd 만 해당됨)   
  
(9) () (parenthesis) : ()는 정규식내에서 패턴을 그룹화 할 때 사용   
  
(10) | (bar) : or를 나타냄   
  
예 : a|b|c (a, b, c 중 하나, 즉 [a-c]와 동일함)   
  
yes|Yes (yes나 Yes 중 하나, [yY]es와 동일함)   
  
korea|japan|chinese (korea, japan, chinese 중 하나)   
  
(11) (backslash) : 위에서 사용된 특수 문자들을 정규식내에서 문자를 취급하고 싶을 때 ''를 선행시켜서 사용하면됨   
  
예 : filename.ext ("filename.ext"를 나타냄)   
  
[?[\]] ('?', '[', '', ']' 중 하나)   
  
  
정규식에서는 위에서 언급한 특수 문자를 제외한 나머지 문자들은 일반 문자로 취급함   
  
  
정규식은 Unix의 대표적인 유틸리티인 vi, emacs, ed, sed, awk, grep, egrep 등에서 사용할 수 있다. 다음은 grep에서 정규식을 활용한 예를 보여 주고 있다.   
  
(1) $ 명령어 | grep '정규식'   
  
17.11.1999   
  
예 4 :   
  
코드 => $joomin="711011-1234567";   
  
if (ereg("([0-9]{2})([01]{1}[09]{1}[0-3]{1}[0-9]{1})-([12]{1}[0-9]{6})",$date, $regs))   
  
print("Valid");   
  
else print("Invalid format: $joomin");   
  
  
int eregi(string givenPattern, string givenString, array matched);   
  
- ereg의 'case insensitive' 버젼   
  
  
예 :   
  
코드 => $email="[xs9\_tx-abc.yyy\_c@cne.kyungsung.ac.kr](mailto:xs9_tx-abc.yyy_c@cne.kyungsung.ac.kr)";   
  
eregi("(^[\_.0-9a-z-]+)@(([0-9a-z][0-9a-z-]+.)+)([a-z]{2,3}$)",$email,$matched);   
  
while (list($a,$b)=each($matched))   
  
if ($b) print("$a, $b ");   
  
  
결과 => 0, [xs9\_tx-abc.yyy\_c@cne.kyungsung.ac.kr](mailto:xs9_tx-abc.yyy_c@cne.kyungsung.ac.kr)   
  
1, xs9\_tx-abc.yyy\_c   
  
2, cne.kyungsung.ac.   
  
3, ac.   
  
4, kr   
  
  
  
코드 => eregi("^[\_.0-9a-z-]+@([0-9a-z][0-9a-z-]+.)+[a-z]{2,3}$",$email,$matched);   
  
while (list($a,$b)=each($matched))   
  
if ($b) print("$a, $b ");   
  
결과 => 0, [xs9\_tx-abc.yyy\_c@cne.kyungsung.ac.kr](mailto:xs9_tx-abc.yyy_c@cne.kyungsung.ac.kr)   
  
1, ac.   
  
  
  
string ereg\_replace(string givenPattern, string replacementPattern, string givenString);   
  
- givenString에서 givenPattern에 부합하는 텍스트(matched text)를 찾아서,   
  
replacementPattern으로 대체   
  
- givenPattern이 "(패턴)"으로 묶인 문자열들을 포함하고 있으면, replacementPattern에는 이에 대응하는 "\digit(문자열)" 형태의 문자열들을 포함하고 있어야 한다(digit는 0, 1, ... ,9 중 하나). 그리고 givenString은 "(패턴)"을 이용해 찾은 결과들을 "\digit(문자열)"에 있는 "문자열"들로 대체하게 된다. "\0" 는 givenString 전체에 대해 "(패턴)"의 결과를 적용할 때 이용된다.   
  
- 변경된 문자열을 리턴   
  
- case sensitive   
  
  
예 :   
  
코드 => $string = "This is a test";   
  
print(ereg\_replace(" is", " was",$string)); print("");   
  
print(ereg\_replace("( )is","\1was",$string)); print("");   
  
print(ereg\_replace("(( )is)","\2was",$string)); print("");   
  
print(ereg\_replace("(( )is)(( )a)(( )test)", "\1was\2an\3exam",$string));   
  
결과 => "This was a test";   
  
"This was a test";   
  
"This was a test";   
  
"This was an exam";   
  
  
예 2 : redundant whitespace 없애기   
  
코드 => $str ="~ s/s+/ /g";   
  
$str = eregi\_replace("[[:space:]]+", " ", $str);   
  
print("$str");   
  
결과 => ~ s/s+/ /g   
  
  
string eregi\_replace(string givenPattern, string replacementPattern, string givenString);   
  
- ereg\_replace의 'case insensitive' 버젼

<http://blog.bagesoft.com/178>

본 문서의 저작권은 [anti-nhn license](http://iilii.egloos.com/4445254)에 따릅니다.

\*본 페이지에는 자바스크립트가 많이 들어있습니다. 자바스크립트가 실행되지 않으면 제대로 보이지 않습니다.  
\*본 페이지는 IE용 태그를 사용하였으므로, firefox 등에서는 정상작동하지 않을 수 있습니다.  
\*테스트를 해보실라문 [요기를 클릭](http://kio.zc.bz/Lecture/reg_test.html)

**차례**

* [1. 정규식이란?](http://kio.zc.bz/Lecture/regexp.html#chap01)
* [2. 정규식 만들기](http://kio.zc.bz/Lecture/regexp.html#chap02)
* [3. 정규식 표현법](http://kio.zc.bz/Lecture/regexp.html#chap03)
* [4. 정규식 사용 예제](http://kio.zc.bz/Lecture/regexp.html#chap04)
* [5. Javascript 정규식 함수](http://kio.zc.bz/Lecture/regexp.html#chap05)
* [6. 정규식으로 만든 유용한 Javascript 함수](http://kio.zc.bz/Lecture/regexp.html#chap06)
* [7. Java 정규식 함수](http://kio.zc.bz/Lecture/regexp.html#chap07)

**1. 정규식이란?**

* String의 **검색**, **치환**, **추출**을 위한 패턴.
* 언어별 사용법은 대동소이함.
* 패턴예>전화번호 형식, 이메일 형식 등.

**2. 정규식 만들기**

1. Javascript
   * var regexp = */pattern/[flags]* ;  
     var test = regexp.test(*to be checked*)
   * var regexp = new RegExp(*"pattern"[, "flags"]*);  
     var test = regexp.test(*to be checked*)
   * flags for javascript
     + g : global match, 일반적으로 패턴이 1번만 발견되면 찾기를 종료하지만, g flag가 있으면, 문자열 내에서 모든 패턴을 찾는다.
     + i : ignore case, 대소문자를 고려하지 않고 체크한다.[a-z]와 [A-Z]는 같은 표현이 된다.
     + m : match over multiple lines, 여러 줄에 걸쳐 체크를 한다.
2. Java
   * java.util.regex package
   * Pattern p = Pattern.compile(*"pattern"*);   
     Matcher m = p.matcher(*"string to be checked"*);   
     boolean b = m.matches();
   * boolean b = Pattern.matches(*"pattern"*, *"string to be checked"*);

**3. 정규식 표현법**

\***ⓥ**는 valid, *ⓘ*는 invalid  
\*두꺼운 글씨체는 매칭되는 부분.  
\*예제는 javascript 기준이며, 언어에 따라 다소 차이가 발생할 수 있다.

|  |  |  |
| --- | --- | --- |
| **문자** | **용도** | **예제** |
| \ | * 특수문자를 의미 * 특수문자의 사용을 제외(특수문자 앞에서) | * b는 b라는 글자를 의미 하지만 \b는 단어 경계를 의미 * \*은 0번이상 반복이라는 의미이지만, \\*는 \*이라는 글자를 의미. |
| ^ | 문자열의 시작. []안에서는 not의 의미 \* ^A는 "A로 시작"이라기 보다는 "시작 직후에 A가 나온다"는 의미로 해석하는 것이 좋다. 즉, 시작과 끝과 같은 빈 공간을 하나의 문자로 간주하는 것이 좋다. | **/^A/g**   * **A** string**ⓥ** * an A*ⓘ*   **/[^A]/g**   * A**stringⓥ** * **an**A**ⓥ** |
| $ | 문자열의 마지막 | **/t$/**   * ea**tⓥ** * GREAT*ⓘ* |
| \* | 0번 이상 반복 | **/ab\*d/g**   * **adⓥ** * abd*ⓘ* * **abd**cde**abbbbd**edb**ⓥ** * ab*ⓘ* * axd*ⓘ* |
| + | 1번 이상 반복 ( = {1,} ) | **/ab+d/g**   * ad*ⓘ* * **abdⓥ** * **abd**cde**abbbbd**edb**ⓥ** * ab*ⓘ* * axd*ⓘ* |
| ? | 0번 이나 1번 | **/e?le?/g**   * ang**elⓥ** * angle*ⓘ* * **ele**ment**ⓥ**   **/abc\-?d/g**   * **abc-dⓥ** * abcd*ⓘ* |
| . | new line 을 제외한 모든 글자 | **/.n/g**   * nay, **an** apple is **on** the tree**ⓥ** * nay*ⓘ* |
| (x) | x를 체크하고 체크한 값을 변수로 저장 | **/(f..) (b..)/**   * **foo barⓥ** 1th :foo 2th :bar |
| (?:x) | x를 체크하고 체크한 값을 변수로 저장하지 않음 | **/(?:f..) (b..)/**   * **foo barⓥ** 1th :bar * bar foo*ⓘ* |
| x|y | x 또는 y | **/green|red/**   * **green** apple**ⓥ** * **red** apple**ⓥ** * yellow apple*ⓘ* |
| x(?=y) | x후에 y가 나오고, x부분만 매칭되는 부분으로 간주 | **/blah(?=soft|hard)/**   * **blah**soft**ⓥ** * **blah**hard**ⓥ** * blah soft*ⓘ*   **/blah(?=soft).\*/**   * **blahsoftⓥ** * blahhard*ⓘ* * blah soft*ⓘ* |
| x(?!y) | x가 나오고 그 뒤에 y가 있으면 안 됨 | **/blah(?!hard)/**   * **blah**soft**ⓥ** * blahhard*ⓘ* * **blah** soft**ⓥ** |
| {n} | 앞에 지정한 것이 n개 | **/.{3}/**   * ab*ⓘ* * **abcⓥ** * **abc**d**ⓥ** * **홍길동ⓥ** |
| {n,} | 앞에 지정한 것이 n개 이상 | **/.{3,}/**   * ab*ⓘ* * **abcⓥ** * **abcdⓥ** |
| {n,m} | 앞에 지정한 것이 n~m개 | **/.{3,5}/**   * ab*ⓘ* * **abcⓥ** * **abcdⓥ** * **홍길동ⓥ** |
| [xyz] | x나 y나 z. []안에는 얼마든지 쓸 수 있다. | **/[abc]{2}/**   * **abⓥ** * **ab**c**ⓥ** * adbd*ⓘ* |
| [x-z] | x에서 z까지 | **/[a-z]{4,}/g**   * She **sells** sea **shells** by the sea **shore**는 Very 어렵다!**ⓥ** |
| [^xyz] | x,y,z를 제외한 나머지 모든 것 | **/[^a-z]{2,}/g**   * **I'**m a good man**ⓥ** * **I**am**A**good**M**an**ⓥ** |
| [\b] | 백스페이스. \b와 혼동하지 말것. | **/[\b]/g**   * abcd**ⓥ**   일반적인 String에서는 \b가 백스페이스를 의미한다. |
| \b | 단어의 경계.[\b]와 혼동하지 말것. | **/\bn[a-z]/g**   * I am **no**t a boy**ⓥ** * online*ⓘ* * **no**pe**ⓥ** |
| \B | \b 를 제외한 전부 | **/\Bn[a-z]/g**   * noo**nd**ay**ⓥ** * online*ⓘ* * nope*ⓘ* |
| \cX | 컨트롤X와 매칭. \cM은 컨트롤M과 매칭 |  |
| \d | 숫자.[0-9]와 같음 | **/\d/g**   * **7** eight **9ⓥ** * **123ⓥ**   **/^0[0-9]{2}/g**   * **012**0**ⓥ** * 12011*ⓘ* |
| \D | \d 를 제외한 전부 | **/\D/g**   * 7**eight**9**ⓥ** * 12**?**3**ⓥ** |
| \f | form-feed |  |
| \n | new line |  |
| \r | carriage return |  |
| \s | white space ex>탭, 띄어쓰기, \n, \r | **/k\s/g**   * korea*ⓘ* * blan**k**is**ⓥ** * blank*ⓘ* |
| \S | \s 를 제외한 전부 | **/k\S/g**   * **ko**rea**ⓥ** * blank is*ⓘ* |
| \t | 탭 |  |
| \v | vertical tab |  |
| \w | 알파벳+숫자+\_. [A-Za-z0-9\_]와 동일 | **/\w/g**   * !@#$%^&\*()+**\_**-[]{}\|"':;,.<>?/**ⓥ**           \_가 <b>를 먹여도 별로 티가 안 난다. |
| \W | \w 빼고 전부 | **/\W/g**   * **!@#$%^&\*()+**\_**-[]{}\|"':;,.<>?/ⓥ** |
| \n | \n이 자연수일때, ()로 지정한 n번째 정규식 | **/(.{2})e tru\1 is out \1ere/**   * **the truth is out there** ...**ⓥ** 1th :th   (th)가 \1로 지정된다. |
| \xhh | hh는 hexacode, | **/[\x21-\x40]/g**   * **!@#$%**^**&\*()**po**ⓥ**   [Code table 보기](http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/iso_table.html) |
| \uhhhh | hhhh는 hexacode, | **/[\u3131-\u3163\uac00-\ud7a3]/g**   * **나는** blah **직원입니다**.**ㅋㅋⓥ**   코드 번호> 3131:ㄱ 3163:ㅣ ac00:가 d7a3:힣 (javascript, java) |

**4. 정규식 사용 예제**

|  |
| --- |
| **/^[0-9]/**   * **0**9없다**ⓥ** * **1**00점**ⓥ** * 집이 10평*ⓘ* |
| **/^\w+$/**   * **blahsoftⓥ** * blah(co)*ⓘ* * blah soft*ⓘ* |
| **/^[a-zA-Z][\w\-]{4,11}$/**   * **blah2010ⓥ** * blah-2010!*ⓘ* * 2010blah*ⓘ* * ILikegoooooooooooooooooogle*ⓘ* |
| **/^[0-9]{2,3}-[0-9]{3,4}-[0-9]{4}/**   * **02-6288-2114ⓥ** * **031-779-7114ⓥ** * **12-1234-5678ⓥ** * **02-6288-2114**545**ⓥ** * **02-0288-2114ⓥ** |
| **/^0\d{1,2}-[1-9]\d{2,3}-\d{4}$/**   * **02-6288-2114ⓥ** * **031-779-7114ⓥ** * 12-1234-5678*ⓘ* * 02-2123-12314545*ⓘ* * 02-0288-2114*ⓘ* |
| **/^[\.a-zA-Z0-9\-]+\.[a-zA-Z]{2,}/**   * **r-d.blah.co.krⓥ** * **r-d.blah.co.kr**입니다.**ⓥ** * **blah..co.krⓥ** * **a.comⓥ** |
| **/^(?:[\w\-]{2,}\.)+[a-zA-Z]{2,}$/**   * **r-d.blah.co.krⓥ** * r-d.blah.co.kr입니다.*ⓘ* * blah..co.kr*ⓘ* * a.com*ⓘ* |
| **/^[\_a-zA-Z0-9\-]+@[\.\_a-zA-Z0-9\-]+\.[a-zA-Z]{2,}/**   * **abc@haha.co.krⓥ** * **abc@haha..co.krⓥ** * **hwang@a.comⓥ** |
| **/^[\w\-]+@(?:(?:[\w\-]{2,}\.)+[a-zA-Z]{2,})$/**   * **abc@haha.co.krⓥ** * abc@haha..co.kr*ⓘ* * hwang@a.com*ⓘ* |
| **/^([a-z]+):\/\/((?:[a-z\d\-]{2,}\.)+[a-z]{2,})(:\d{1,5})?(\/[^\?]\*)?(\?.+)?$/i**   * **http://www.blah.co.kr/main/index.jsp?var=valueⓥ** 1th :http 2th :www.blah.co.kr 3th :undefined 4th :/main/index.jsp 5th :?var=value * **http://www.blah.co.kr/main/index.jspⓥ** 1th :http 2th :www.blah.co.kr 3th :undefined 4th :/main/index.jsp 5th :undefined * **http://blah.co.kr/ⓥ** 1th :http 2th :blah.co.kr 3th :undefined 4th :/ 5th :undefined * **http://blah.co.krⓥ** 1th :http 2th :blah.co.kr 3th :undefined 4th :undefined 5th :undefined * **http://blah.co.kr:8088/main/ⓥ** 1th :http 2th :blah.co.kr 3th ::8088 4th :/main/ 5th :undefined |
| **/^[ㄱ-ㅣ가-힣]+$/**   * **티맥스소프트ⓥ** * **ㅜㅜⓥ** * **ㅎㅎⓥ** |

**5. Javascript 정규식 함수**

|  |  |  |
| --- | --- | --- |
| **함수** | **코드예제** | **코드설명** |
| Array RegExp.exec (*to be checked*) | var myRe=/d(b+)(d)/ig; var myArray = myRe.exec("cdbBdbsbz");  **/d(b+)(d)/gi**   * c**dbBd**bsbz**ⓥ** 1th :bB 2th :d | myArray.index =1 ; (처음으로 매칭되는 위치, 컴터가 늘 그렇듯 위치는 0번째부터 센다.) myArray.input = cdbBdbsbz; (체크할 대상) myArray[0] = dbBd;(검사에 통과한 부분) myArray[1] = bB;(1번째 괄호에서 체크된 부분) myArray[2] = d;(2번째 괄호에서 체크된 부분)  myRe.lastIndex =5 ; (다음번 체크를 하기위한 위치.) myRe.ignoreCase = true; (/i 플래그 체크) myRe.global = true; (/g 플래그 체크) myRe.multiline = false; (/m 플래그 체크)  RegExp.$\_ = cdbBdbsbz;(입력한 스트링) RegExp.$1 = bB;(1번째 괄호에서 체크된 부분 ) |
| boolean RegExp.test(*to be checked*) | var myRe=/d(b+)(d)/ig; var checked = myRe.test("cdbBdbsbz"); document.write("checked = " + checked +";<br>");  **/d(b+)(d)/gi**   * c**dbBd**bsbz**ⓥ** 1th :bB 2th :d | 실행결과: checked = true; |
| String RegExp.toString() | var myRe=/d(b+)(d)/ig; var str = myRe.toString(); document.write(str); | 실행 결과: /d(b+)(d)/gi |
| String String.replace(*pattern or string*, *to be replaced*) | var str = "abcdefe"; document.write(str.replace("e" , "f")); | 실행 결과: abcdffe   e가 2번 있지만, 첫번째 인자가 정규식이 아니라 문자열일 경우는 첫번째 것만 바꾼다. |
| var str = "aba"; document.write(str.replace(/^a/ , "c")); | 실행 결과: cba |
| var re = /(\w+)\s(\w+)/; var str = "John Smith"; newstr = str.replace(re, "$2, $1"); document.write(newstr) | 실행 결과: Smith, John   re에 의해서 찾아진 문자열 들은 re에서 ()로 표현된 순서대로 $1, $2와 같이 변수로 저장된다. |
| var re = /\s(?:http|https):\/\/\S\*(?:\s|$)/g; var str = "url is http://iilii.egloos.com/ !!\n"; str += "blah home: http://www.blah.co.kr"; newstr = str.replace(re, function (str,p1,offset,s) {      return "<a href='" + str + "'>" + str + "</a>";   } ).replace(/\n/, "<br>"); document.write(newstr); | url is<http://iilii.egloos.com/>!! blah home:[http://www.blah.co.kr](http://www.blah.co.kr/)   str: 찾은 문자열 p1: ()에서 검색된 1번째 문자열. 마찬가지로 p2,p3 등도 가능 offset: str을 찾은 위치 s : 원본 문자열. |
| Array String.match(*regular expression* | var str = "ABCdEFgHiJKL"; var myResult = str.match(/[a-z]/g ); for(var cnt = 0 ; cnt < myResult.length; cnt++){     document.write(cnt +":" + myResult[cnt] +"<br>"); }  document.write("비교<br>");  var str = "ABCdEFgHiJKL"; var myResult = /[a-z]/g.exec(str); for(var cnt = 0 ; cnt < myResult.length; cnt++){     document.write(cnt +":" + myResult[cnt] +"<br>"); } | 실행 결과: 0:d 1:g 2:i 비교 0:d  String.match(RegExp) =>g flag가 있어도 다 찾아낸다.  RegExp.exec(String) =>g flag가 있으면, 한 개만 찾고 끝낸다. |
| Array String.split(*[separator[, limit]]*) | var str = "ABCdEFgHiJKL"; var myResult = str.split(/[a-z]/g , 3); for(var cnt = 0 ; cnt < myResult.length; cnt++){     document.write(cnt +":" + myResult[cnt] +"<br>"); } | 실행 결과: 0:ABC 1:EF 2:H  주어진 문자열을 separator를 기준으로 limit 만큼 자른다. |

**6. 정규식으로 만든 유용한 Javascript 함수**

|  |
| --- |
| **String removeTags(*input*)**  HTML tag부분을 없애준다 |
| function removeTags(input) {     return input.replace(/<[^>]+>/g, "");  }; |
| example>  var str = "<b>blah</b> <i>soft</i>"; document.write(str +"<br>"); document.write(removeTags(str));  result> **blah** *soft* blah soft |
| **String String.trim()**  문자열의 앞뒤 공백을 없애준다. |
| String.prototype.trim = function() {     return this.replace(/^\s+|\s+$/g, '');  }; |
| example>  var str = "         untrimed string            "; document.write("========" + str+ "==============<br>"); document.write("========" + str.trim() + "==============");  result> ======== untrimed string ============== ========untrimed string============== |
| **String String.capitalize()**  단어의 첫 글자를 대문자로 바꿔준다. |
| String.prototype.capitalize = function() {     return this.replace(/\b([a-z])/g, function($1){         return $1.toUpperCase();     }) ;   }; |
| example>  var str = "korea first world best"; document.write(str.capitalize());  result> Korea First World Best |
| **String number\_format(*input*)**  입력된 숫자를 ,를 찍은 형태로 돌려준다 |
| function number\_format(input){     var input = String(input);     var reg = /(\-?\d+)(\d{3})($|\.\d+)/;     if(reg.test(input)){         return input.replace(reg, function(str, p1,p2,p3){                 return number\_format(p1) + "," + p2 + "" + p3;             }             );     }else{         return input;     } } |
| example>  document.write(number\_format(1234562.12) + "<br>"); document.write(number\_format("-9876543.21987")+ "<br>"); document.write(number\_format("-123456789.12")+ "<br>");  result> 1,234,562.12 -9,876,543.21987 -123,456,789.12 |

**7. Java 정규식 함수**

|  |
| --- |
| Pattern p = Pattern.compile("(a\*)(b)");  Matcher m = p.matcher("aaaaab");  if (m.matches()) {  for (int i = 0; i < m.groupCount() + 1; i++) {  System.out.println(i + ":" + m.group(i));  }  } else {  System.out.println("not match!");  }  result>  0:aaaaab  1:aaaaa  2:b  0번째는 매칭된 부분. |
| String a = "I love her";  System.out.println(a.replaceAll("([A-Z])", "\"$1\""));  result>  "I" love her  자바도 $1을 쓸 수 있다. |
| Pattern p = Pattern.compile("cat");  Matcher m = p.matcher("one cat two cats in the yard");  StringBuffer sb = new StringBuffer();  while (m.find()) {  m.appendReplacement(sb, "dog");  System.out.println(sb.toString());  }  m.appendTail(sb);  System.out.println(sb.toString());  result>  one dog  one dog two dog  one dog two dogs in the yard |

<http://kio.zc.bz/Lecture/regexp.html>

2006년 8월 [노세환](mailto:sehnoh%20at%20java2go.net) 작성

# Regular Expressions in Java

|  |
| --- |
| [코드 크게](http://www.java2go.net/java/java_regex.html) |

* [자바 정규표현식](http://www.java2go.net/java/java_regex.html#Java_Regex_Basics)
* [자바 정규표현식 사용 예제](http://www.java2go.net/java/java_regex.html#Java_Regex_Samples)
* [Java Regex References](http://www.java2go.net/java/java_regex.html#Java_Regex_References)

## 자바 정규표현식

자바의 정규표현식은 J2SE 1.4 부터 지원되지 시작했습니다. 관련된 주요 클래스들는 java.util.regex 팩키지에 있습니다.

### Pattern 클래스

Pattern 객체는 Perl 문법과 비슷한 형태로 정의된 정규표현식을 나타냅니다. 문자열로 정의한 정규표현식은 사용되기 전에 반드시 Pattern 클래스의 인스턴스로 컴파일되어야 합니다. 컴파일된 패턴은 Matcher 객체를 만드는 데 사용되며, Matcher 객체는 임의의 입력 문자열이 패턴에 부합되는 지 여부를 판가름하는 기능을 담당하합니다. 또한 Pattern 객체들은 비상태유지 객체들이기 때문에 여러 개의 Matcher 객체들이 공유할 수 있습니다.

### Matcher 클래스

Matcher 객체는 특정한 문자열이 주어진 패턴과 일치하는가를 알아보는데 이용됩니다. Matcher 클래스의 입력값으로는 CharSequence라는 새로운 인터페이스가 사용되는데 이를 통해 다양한 형태의 입력 데이터로부터 문자 단위의 매칭 기능을 지원 받을 수 있습니다. 기본적으로 제공되는 CharSequence 객체들은 CharBuffer, String, StringBuffer 클래스가 있습니다.

Matcher 객체는 Pattern 객체의 matcher 메소드를 통해 얻어진다. Matcher 객체가 일단 만들어지면 주로 세 가지 목적으로 사용됩다.

* 주어진 문자열 전체가 특정 패턴과 일치하는 가를 판단(matches).
* 주어진 문자열이 특정 패턴으로 시작하는가를 판단(lookingAt).
* 주어진 문자열에서 특정 패턴을 찾아낸다(find).

이들 메소드는 성공 시 true를 실패 시 false 를 반환합니다.

또한 특정 문자열을 찾아 새로운 문자열로 교체하는 기능도 제공됩니다. appendRepalcement 메소드는 일치하는 패턴이 나타날 때까지의 모든 문자들을 버퍼로 옮기고 찾아진 문자열 대신 교체 문자열로 채워 넣습니다. 또한 appendTail 메소드는 캐릭터 시퀀스의 현재 위치 이후의 문자들을 버퍼에 복사해 넣습니다. 다음 절에 나오는 예제 코드를 참고하도록 합시다.

### CharSequence 인터페이스

CharSequence 인터페이스는 다양한 형태의 캐릭터 시퀀스에 대해 일관적인 접근 방법을 제공하기 위해 새로 생겨났으며, java.lang 패키지에 존재합니다. 기본적으로 String, StringBuffer, CharBuffer 클래스가 이를 구현하고 있으므로 적절한 것을 골라 사용하면 되며, 인터페이스가 간단하므로 필요하면 직접 이를 구현해 새로 하나 만들어도 됩니다.

## 자바 정규표현식 사용 예제

### 기본 사용 예제

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* 정규표현식 기본 사용 예제  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest01 {  public static void main(String[] args) {  Pattern p = Pattern.compile("a\*b");  Matcher m = p.matcher("aaaaab");  boolean b = m.matches();    if (b) {  System.out.println("match");  } else {  System.out.println("not match");  }  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| match |

### 문자열 치환하기

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* 문자열 치환 예제  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest02 {  public static void main(String[] args) {  Pattern p = Pattern.compile("cat");  Matcher m = p.matcher("one cat two cats in the yard");    StringBuffer sb = new StringBuffer();  while (m.find()) {  m.appendReplacement(sb, "dog");  }  m.appendTail(sb);  System.out.println(sb.toString());    // or  //String str = m.replaceAll("dog");  //System.out.println(str);  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| one dog two dogs in the yard |

### 이메일주소 유효검사

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* 이메일주소 유효검사  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest03 {    public static boolean isValidEmail(String email) {  Pattern p = Pattern.compile("^(?:\\w+\\.?)\*\\w+@(?:\\w+\\.)+\\w+$");  Matcher m = p.matcher(email);  return m.matches();  }  public static void main(String[] args) {    String[] emails = { "test@abc.com", "a@.com", "abc@mydomain" };    for (int i = 0; i < emails.length; i ++) {  if (isValidEmail(emails[i])) {  System.out.println(emails[i]);  }  }  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| test@abc.com |

### HTML 태그 제거

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* HTML 태그 제거  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest04 {  public static String stripHTML(String htmlStr) {  Pattern p = Pattern.compile("<(?:.|\\s)\*?>");  Matcher m = p.matcher(htmlStr);  return m.replaceAll("");  }    public static void main(String[] args) {  String htmlStr = "<html><body><h1>Java2go.net</h1>"  + " <p>Sehwan@Noh's Personal Workspace...</p></body></html>";  System.out.println(stripHTML(htmlStr));  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| Java2go.net Sehwan@Noh's Personal Workspace... |

### HTML 링크 만들기

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* HTML 링크 만들기  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest05 {  public static String linkedText(String sText) {  Pattern p = Pattern.compile("(http|https|ftp)://[^\\s^\\.]+(\\.[^\\s^\\.]+)\*");  Matcher m = p.matcher(sText);  StringBuffer sb = new StringBuffer();  while (m.find()) {  m.appendReplacement(sb, "<a href='" + m.group()+"'>" + m.group() + "</a>");  }  m.appendTail(sb);  return sb.toString();  }    public static void main(String[] args) {  String strText = "My homepage URL is http://www.java2go.net/home/index.html.";  System.out.println(linkedText(strText));  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| My homepage URL is <a href='http://www.java2go.net/index.html'>http://www.java2go.net/index.html</a>. |

### 금지어 필터링하기

[소스](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* 금지어 필터링하기  \*  \* @author Sehwan Noh <sehnoh at java2go.net>  \* @version 1.0 - 2006. 08. 22  \* @since JDK 1.4  \*/  public class RegExTest06 {    public static String filterText(String sText) {  Pattern p = Pattern.compile("fuck|shit|개새끼", Pattern.CASE\_INSENSITIVE);  Matcher m = p.matcher(sText);  StringBuffer sb = new StringBuffer();  while (m.find()) {  //System.out.println(m.group());  m.appendReplacement(sb, maskWord(m.group()));  }  m.appendTail(sb);    //System.out.println(sb.toString());  return sb.toString();  }    public static String maskWord(String word) {  StringBuffer buff = new StringBuffer();  char[] ch = word.toCharArray();  for (int i = 0; i < ch.length; i++) {  if (i < 1) {  buff.append(ch[i]);  } else {  buff.append("\*");  }  }  return buff.toString();  }    public static void main(String[] args) {  String sText = "Shit! Read the fucking manual. 개새끼야.";  System.out.println(filterText(sText));  }  } |

[결과](http://www.java2go.net/java/java_regex.html)

|  |
| --- |
| S\*\*\*! Read the f\*\*\*ing manual. 개\*\*야. |

## Java Regex References

A regular expression, specified as a string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a Matcher object that can match arbitrary character sequences against the regular expression. All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

A typical invocation sequence is thus

Pattern p = Pattern.compile("a\*b");

Matcher m = p.matcher("aaaaab");

boolean b = m.matches();

A matches method is defined by this class as a convenience for when a regular expression is used just once. This method compiles an expression and matches an input sequence against it in a single invocation. The statement

boolean b = Pattern.matches("a\*b", "aaaaab");

is equivalent to the three statements above, though for repeated matches it is less efficient since it does not allow the compiled pattern to be reused.

Instances of this class are immutable and are safe for use by multiple concurrent threads. Instances of the Matcher class are not safe for such use.

### Summary of regular-expression constructs

|  |  |
| --- | --- |
| **Construct** | **Matches** |
|  |  |
| **Characters** | |
| x | The character x |
| \\ | The backslash character |
| \0n | The character with octal value 0n (0 <= n <= 7) |
| \0nn | The character with octal value 0nn (0 <= n <= 7) |
| \0mnn | The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7) |
| \xhh | The character with hexadecimal value 0xhh |
| \uhhhh | The character with hexadecimal value 0xhhhh |
| \t | The tab character ('\u0009') |
| \n | The newline (line feed) character ('\u000A') |
| \r | The carriage-return character ('\u000D') |
| \f | The form-feed character ('\u000C') |
| \a | The alert (bell) character ('\u0007') |
| \e | The escape character ('\u001B') |
| \cx | The control character corresponding to x |
|  |  |
| **Character classes** | |
| [abc] | a, b, or c (simple class) |
| [^abc] | Any character except a, b, or c (negation) |
| [a-zA-Z] | a through z or A through Z, inclusive (range) |
| [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| [a-z&&[def]] | d, e, or f (intersection) |
| [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |
|  |  |
| **Predefined character classes** | |
| . | Any character (may or may not match line terminators) |
| \d | A digit: [0-9] |
| \D | A non-digit: [^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z\_0-9] |
| \W | A non-word character: [^\w] |
|  |  |
| **POSIX character classes (US-ASCII only)** | |
| \p{Lower} | A lower-case alphabetic character: [a-z] |
| \p{Upper} | An upper-case alphabetic character:[A-Z] |
| \p{ASCII} | All ASCII:[\x00-\x7F] |
| \p{Alpha} | An alphabetic character:[\p{Lower}\p{Upper}] |
| \p{Digit} | A decimal digit: [0-9] |
| \p{Alnum} | An alphanumeric character:[\p{Alpha}\p{Digit}] |
| \p{Punct} | Punctuation: One of !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~ |
| \p{Graph} | A visible character: [\p{Alnum}\p{Punct}] |
| \p{Print} | A printable character: [\p{Graph}\x20] |
| \p{Blank} | A space or a tab: [ \t] |
| \p{Cntrl} | A control character: [\x00-\x1F\x7F] |
| \p{XDigit} | A hexadecimal digit: [0-9a-fA-F] |
| \p{Space} | A whitespace character: [ \t\n\x0B\f\r] |
|  |  |
| **java.lang.Character classes (simple java character type)** | |
| \p{javaLowerCase} | Equivalent to java.lang.Character.isLowerCase() |
| \p{javaUpperCase} | Equivalent to java.lang.Character.isUpperCase() |
| \p{javaWhitespace} | Equivalent to java.lang.Character.isWhitespace() |
| \p{javaMirrored} | Equivalent to java.lang.Character.isMirrored() |
|  |  |
| **Classes for Unicode blocks and categories** | |
| \p{InGreek} | A character in the Greek block (simple block) |
| \p{Lu} | An uppercase letter (simple category) |
| \p{Sc} | A currency symbol |
| \P{InGreek} | Any character except one in the Greek block (negation) |
| [\p{L}&&[^\p{Lu}]] | Any letter except an uppercase letter (subtraction) |
|  |  |
| **Boundary matchers** | |
| ^ | The beginning of a line |
| $ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | The end of the input but for the final terminator, if any |
| \z | The end of the input |
|  |  |
| **Greedy quantifiers** | |
| X? | X, once or not at all |
| X\* | X, zero or more times |
| X+ | X, one or more times |
| X{n} | X, exactly n times |
| X{n,} | X, at least n times |
| X{n,m} | X, at least n but not more than m times |
|  |  |
| **Reluctant quantifiers** | |
| X?? | X, once or not at all |
| X\*? | X, zero or more times |
| X+? | X, one or more times |
| X{n}? | X, exactly n times |
| X{n,}? | X, at least n times |
| X{n,m}? | X, at least n but not more than m times |
|  |  |
| **Possessive quantifiers** | |
| X?+ | X, once or not at all |
| X\*+ | X, zero or more times |
| X++ | X, one or more times |
| X{n}+ | X, exactly n times |
| X{n,}+ | X, at least n times |
| X{n,m}+ | X, at least n but not more than m times |
|  |  |
| **Logical operators** | |
| XY | X followed by Y |
| X|Y | Either X or Y |
| (X) | X, as a capturing group |
|  |  |
| **Back references** | |
| \n | Whatever the nth capturing group matched |
|  |  |
| **Quotation** | |
| \ | Nothing, but quotes the following character |
| \Q | Nothing, but quotes all characters until \E |
| \E | Nothing, but ends quoting started by \Q |
|  |  |
| **Special constructs (non-capturing)** | |
| (?:X) | X, as a non-capturing group |
| (?idmsux-idmsux) | Nothing, but turns match flags on - off |
| (?idmsux-idmsux:X) | X, as a non-capturing group with the given flags on - off |
| (?=X) | X, via zero-width positive lookahead |
| (?!X) | X, via zero-width negative lookahead |
| (?<=X) | X, via zero-width positive lookbehind |
| (?<!X) | X, via zero-width negative lookbehind |
| (?>X) | X, as an independent, non-capturing group |

### Backslashes, escapes, and quoting

The backslash character ('\') serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression \\ matches a single backslash and \{ matches a left brace.

It is an error to use a backslash prior to any alphabetic character that does not denote an escaped construct; these are reserved for future extensions to the regular-expression language. A backslash may be used prior to a non-alphabetic character regardless of whether that character is part of an unescaped construct.

Backslashes within string literals in Java source code are interpreted as required by the Java Language Specification as either Unicode escapes or other character escapes. It is therefore necessary to double backslashes in string literals that represent regular expressions to protect them from interpretation by the Java bytecode compiler. The string literal "\b", for example, matches a single backspace character when interpreted as a regular expression, while "\\b" matches a word boundary. The string literal "\(hello\)" is illegal and leads to a compile-time error; in order to match the string (hello) the string literal "\\(hello\\)" must be used.

### Character Classes

Character classes may appear within other character classes, and may be composed by the union operator (implicit) and the intersection operator (&&). The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that is in both of its operand classes.

The precedence of character-class operators is as follows, from highest to lowest:

|  |  |  |
| --- | --- | --- |
| **1** | Literal escape | \x |
| **2** | Grouping | [...] |
| **3** | Range | a-z |
| **4** | Union | [a-e][i-u] |
| **5** | Intersection | [a-z&&[aeiou]] |

Note that a different set of metacharacters are in effect inside a character class than outside a character class. For instance, the regular expression . loses its special meaning inside a character class, while the expression - becomes a range forming metacharacter.

### Line terminators

A line terminator is a one- or two-character sequence that marks the end of a line of the input character sequence. The following are recognized as line terminators:

* A newline (line feed) character ('\n'),
* A carriage-return character followed immediately by a newline character ("\r\n"),
* A standalone carriage-return character ('\r'),
* A next-line character ('\u0085'),
* A line-separator character ('\u2028'), or
* A paragraph-separator character ('\u2029).

If UNIX\_LINES mode is activated, then the only line terminators recognized are newline characters.

The regular expression . matches any character except a line terminator unless the DOTALL flag is specified.

By default, the regular expressions ^ and $ ignore line terminators and only match at the beginning and the end, respectively, of the entire input sequence. If MULTILINE mode is activated then ^ matches at the beginning of input and after any line terminator except at the end of input. When in MULTILINE mode $ matches just before a line terminator or the end of the input sequence.

### Groups and capturing

Capturing groups are numbered by counting their opening parentheses from left to right. In the expression ((A)(B(C))), for example, there are four such groups:

|  |  |
| --- | --- |
| **1** | ((A)(B(C))) |
| **2** | (A) |
| **3** | (B(C)) |
| **4** | (C) |

Group zero always stands for the entire expression.

Capturing groups are so named because, during a match, each subsequence of the input sequence that matches such a group is saved. The captured subsequence may be used later in the expression, via a back reference, and may also be retrieved from the matcher once the match operation is complete.

The captured input associated with a group is always the subsequence that the group most recently matched. If a group is evaluated a second time because of quantification then its previously-captured value, if any, will be retained if the second evaluation fails. Matching the string "aba" against the expression (a(b)?)+, for example, leaves group two set to "b". All captured input is discarded at the beginning of each match.

Groups beginning with (? are pure, non-capturing groups that do not capture text and do not count towards the group total.

### Unicode support

This class is in conformance with Level 1 of [Unicode Technical Standard #18: Unicode Regular Expression Guidelines](http://www.unicode.org/reports/tr18/), plus RL2.1 Canonical Equivalents.

Unicode escape sequences such as \u2014 in Java source code are processed as described in ?.3 of the Java Language Specification. Such escape sequences are also implemented directly by the regular-expression parser so that Unicode escapes can be used in expressions that are read from files or from the keyboard. Thus the strings "\u2014" and "\\u2014", while not equal, compile into the same pattern, which matches the character with hexadecimal value 0x2014.

Unicode blocks and categories are written with the \p and \P constructs as in Perl. \p{prop} matches if the input has the property prop, while \P{prop} does not match if the input has that property. Blocks are specified with the prefix In, as in InMongolian. Categories may be specified with the optional prefix Is: Both \p{L} and \p{IsL} denote the category of Unicode letters. Blocks and categories can be used both inside and outside of a character class.

The supported categories are those of [The Unicode Standard](http://www.unicode.org/unicode/standard/standard.html) in the version specified by the Character class. The category names are those defined in the Standard, both normative and informative. The block names supported by Pattern are the valid block names accepted and defined by UnicodeBlock.forName.

Categories that behave like the java.lang.Character boolean ismethodname methods (except for the deprecated ones) are available through the same \p{prop} syntax where the specified property has the name javamethodname.

### Comparison to Perl 5

The Pattern engine performs traditional NFA-based matching with ordered alternation as occurs in Perl 5.

Perl constructs not supported by this class:

* The conditional constructs (?{X}) and (?(condition)X|Y),
* The embedded code constructs (?{code}) and (??{code}),
* The embedded comment syntax (?#comment), and
* The preprocessing operations \l \u, \L, and \U.

Constructs supported by this class but not by Perl:

* Possessive quantifiers, which greedily match as much as they can and do not back off, even when doing so would allow the overall match to succeed.
* Character-class union and intersection as described above.

Notable differences from Perl:

* In Perl, \1 through \9 are always interpreted as back references; a backslash-escaped number greater than 9 is treated as a back reference if at least that many subexpressions exist, otherwise it is interpreted, if possible, as an octal escape. In this class octal escapes must always begin with a zero. In this class, \1 through \9 are always interpreted as back references, and a larger number is accepted as a back reference if at least that many subexpressions exist at that point in the regular expression, otherwise the parser will drop digits until the number is smaller or equal to the existing number of groups or it is one digit.
* Perl uses the g flag to request a match that resumes where the last match left off. This functionality is provided implicitly by the Matcher class: Repeated invocations of the find method will resume where the last match left off, unless the matcher is reset.
* In Perl, embedded flags at the top level of an expression affect the whole expression. In this class, embedded flags always take effect at the point at which they appear, whether they are at the top level or within a group; in the latter case, flags are restored at the end of the group just as in Perl.
* Perl is forgiving about malformed matching constructs, as in the expression \*a, as well as dangling brackets, as in the expression abc], and treats them as literals. This class also accepts dangling brackets but is strict about dangling metacharacters like +, ? and \*, and will throw a PatternSyntaxException if it encounters them.

*마지막 수정일: 2006년 8월 23일*

<http://www.java2go.net/java/java_regex.html>