

Regular Expression (RegEx)- Espressioni regolari

Una espressione regolare definisce una funzione che prende in ingresso una stringa, e restituisce in uscita un valore del tipo sì/no, a seconda che la stringa segua o meno un certo *pattern*.

Ci sono 3 dialetti: classiche, avanzate e di .net. Cambia il modo in cui il calcolatore interpreta le regex (cambiano i tempi di risposta della *isMatch()*)

Sono racchiuse tra "/" "/" o '/' '/' a seconda del linguaggio

I metacaratteri

Nelle espressioni regolari esistono diversi "*caratteri speciali*" dalle diverse funzioni

Meta	Descrizione
.	significa qualsiasi carattere ad eccezione di quelli che identificano una riga nuova (\n e \r per intenderci)
^	identifica l'inizio di una riga; inoltre all'inizio di un gruppo nega il gruppo stesso
\$	identifica la fine di una riga
	è una condizione OR
()	le parentesi tonde identificano dei gruppi di caratteri
[]	le parentesi quadre identificano intervalli e classi di caratteri
\	questo carattere annulla gli effetti del metacarattere successivo 1 \$testo = "espressioni.regolari!"; 2 preg_match_all('/\./', \$testo, \$ris); 3 // Troverà solo il punto

I quantificatori

I quantificatori, come dice il termine stesso, indicano quante volte ricercare una data sequenza di caratteri.

Classi	Descrizione
*	indica 0 o più occorrenze
+	indica 1 o più occorrenze
?	indica 1 o 0 occorrenze
{n}	ricerca esattamente n occorrenze; da ricordare che le parentesi grafe vengono considerate caratteri normali in tutti gli altri contesti
{n,}	ricerca minimo n occorrenze; vedi sopra
{n,m}	ricerca minimo n occorrenze ma non superiori alle m; vedi sopra

i quantificatori normali sono "golosi" (in inglese *greedy*), cioè cercano l'occorrenza il più grande possibile.

Vediamo con un esempio:

```
1 $testo = 'class="pluto" id="pippo"';
2 preg_match_all('/".*"/', $testo, $ris);
3 // Troverà un'unica occorrenza:
4 // "pluto" id="pippo"
```

Basta aggiungere un punto interrogativo alla fine dei nostri quantificatori per trovare tutte le parole

```
1 $testo = 'class="pluto" id="pippo"';
2 preg_match_all('/".*?"/', $testo, $ris);
3 // Ora troverà "pluto" e "pippo" !
```

Ancore

Le ancore identificano la posizione in cui ricercare il nostro testo.

Ancora	Descrizione
^	identifica l'inizio della stringa; con il modificatore /m identifica l'inizio di ogni riga
\$	identifica la fine della stringa; con il modificatore /m identifica la fine di ogni riga

`"/ a5b /"` accetta **a5b** **za5bk** ma non `ab5a` né `5b`

`"/^[0-9]+$/"` numeri naturali

`"/^[\\-\\+]?[0-9]+$/"` numeri interi

`"/(b{2})|(^b{2})/"` 2 b or not 2 b

In questa sintassi, la maggior parte dei caratteri sono visti come letterali, e trovano solo se stessi. Ad esempio: `"a"` trova `"a"`; `"bc)"` trova `"bc)"`; ecc. Le eccezioni a questa regola sono i **metacaratteri** o **operatori**:

.	Trova un singolo carattere (se è nella modalità <i>linea singola</i> altrimenti se è in <i>multiriga</i> prende tutti i caratteri diversi da <code>\n</code> , ovvero un ritorno a capo).
[]	<p>Trova un singolo carattere contenuto nelle parentesi (sono in <i>or</i>). Ad esempio, <code>[abc]</code> trova o una <code>"a"</code>, <code>"b"</code>, o <code>"c"</code>. <code>[a-z]</code> è un intervallo e trova ogni lettera minuscola dell'alfabeto. Possono esserci casi misti: <code>[abcq-z]</code> trova a, b, c, q, r, s, t, u, v, w, x, y, z, esattamente come <code>[a-cq-z]</code>.</p> <p>Il carattere <code>'-'</code> è letterale solo se è primo o ultimo carattere nelle parentesi: <code>[abc-]</code> o <code>[-abc]</code>. Per trovare un carattere <code>'['</code> o <code>']'</code>, il modo più semplice è metterli primi all'interno delle parentesi: <code>[] [ab]</code> trova <code>']'</code>, <code>['</code>, <code>'a'</code> o <code>'b'</code>.</p>

[^]	Trova ogni singolo carattere non incluso nelle parentesi. Ad esempio, [^abc] trova ogni carattere diverso da "a", "b", o "c". [^a-z] trova ogni singolo carattere che non sia una lettera minuscola. Come sopra, questi due metodi possono essere usati insieme.
^	Corrisponde all'inizio della stringa (o di ogni riga della stringa, quando usato in modalità multilinea)
\$	Corrisponde alla fine della stringa o alla posizione immediatamente precedente un carattere di nuova linea (o alla fine di ogni riga della stringa, quando usato in modalità multilinea)
()	Definisce una "sottoespressione marcata". Il risultato di ciò che è incluso nell'espressione, può essere richiamato in seguito. Esempio (ab) (cd) Vedi sotto, \n.
(^)	Definisce un insieme di caratteri da escludere (^ab) non ab
\n	Dove <i>n</i> è una cifra da 1 a 9; trova ciò che la <i>nesima</i> sottoespressione ha trovato. Tale costrutto, detto <i>backreference</i> , estende le potenzialità delle regexp oltre i linguaggi regolari e non è stato adottato nella sintassi estesa delle regexp.
*	<ul style="list-style-type: none"> Cerca l'occorrenza (zero o più volte) del carattere o insieme di caratteri cui segue. Corrisponde a {,} Un'espressione costituita da un singolo carattere seguito da "*", trova zero o più copie di tale espressione. Ad esempio, "[xyz]*" trova "", "x", "y", "zx", "zyx", e così via. \n*, dove <i>n</i> è una cifra da 1 a 9, trova zero o più iterazioni di ciò che la <i>nesima</i> sottoespressione ha trovato. Ad esempio, "(a.)c\1*" trova "abcab" e "accac" ma non "abcac". Un'espressione racchiusa tra "(" e ")" seguita da "*" non è valida. In alcuni casi (es. /usr/bin/xpg4/grep di SunOS 5.8), trova zero o più ripetizioni della stringa che l'espressione racchiusa ha trovato. In altri casi (es. /usr/bin/grep di SunOS 5.8), trova ciò che l'espressione racchiusa ha trovato, seguita da un letterale "*".
?	Cerca l'occorrenza (zero o una volta) del carattere o insieme di caratteri cui segue. Corrisponde a {0,1}
+	Cerca l'occorrenza (una o più volte) del carattere o insieme di caratteri cui segue. Corrisponde a {1,}
{x,y}	Trova l'ultimo "blocco" almeno <i>x</i> volte e non più di <i>y</i> volte. Ad esempio, "a{3,5}" trova "aaa", "aaaa" o "aaaaa". Si può scrivere {,3} vuol dire tra 0 e 3 se scrivo {2,} da 2 in su
\	Per caratteri di escape

Esempi:

".atto" trova ogni stringa di cinque caratteri come *gatto*, *matto* o *patto*
 "[gm]atto" trova *gatto* e *matto*
 "[^p]atto" trova tutte le combinazioni dell'espressione ".atto" tranne *patto*
 "^ [gm]atto" trova *gatto* e *matto* ma solo all'inizio di una riga
 "[gm]atto\$" trova *gatto* e *matto* ma solo alla fine di una riga

\s=blank

\u002d = -

/^(([A-Z]((\'s?[A-Z])|(\s[A-Z]))?[a-z]+)(\s?\u002D?\s)|\'s?)+ \$/ per i cognomi (sono inclusi D'Angelo, De Angelis, Acits-Alesina, Erbi')

linea vuota

^\$

indirizzo email

[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}

data in formato gg/mm/aaaa

(0[1-9]|1[012])[- /.](0[1-9]|1[2][0-9]|3[01])[- /.](19|20)\d\d

ora in formato 12 ore

(1[012]|1[1-9]):[0-5][0-9](\\s)?(?i)(am|pm)

ora in formato 24 ore

([01]?[0-9]|2[0-3]):[0-5][0-9]

url http

http\\:\/\/[a-zA-Z0-9\\-\\.]+\\.[a-zA-Z]{2,3}(\\S*)?

codice fiscale

[a-zA-Z]{6}\\d\\d[a-zA-Z]\\d\\d[a-zA-Z]\\d\\d\\d[a-zA-Z]

nome utente

^[a-z0-9_-]{3,15}\$

(nome utente formato da soli caratteri alfanumerici più _ e - di lunghezza min 3 e max 15)

password

((?=.*\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#\$%]).{8,20})

(password che deve contenere un numero, un carattere minuscolo, uno maiuscolo e un carattere speciale tra @#\$% e deve avere lunghezza min 8 e max 20)

tag HTML

<([A-Z][A-Z0-9]*)\\b[^>]*>(.*?)

uno specifico tag HTML

<TAG\\b[^>]*>(.*?)

codice esadecimale colore

^#[A-Fa-f0-9]{6}|[A-Fa-f0-9]{3}\$

estensione di un file immagine

([^\s]+\\.?(?i)(jpg|png|gif|bmp))\$

Indirizzo IP

^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\$

Sul sito regexr.com si possono scrivere le espressioni e testarle subito

In Java

```
String s="123";
if ( s.matches("[+-]?[0-9]+"))
    //ok
else
    //no matches
```

ATTENZIONE se si deve usare lo \ mettere \\ perché è un carattere di escape

Method	Description
<code>s.matches("regex")</code>	Evaluates if "regex" matches s. Returns only true if the WHOLE string can be matched.
<code>s.split("regex")</code>	Creates an array with substrings of s divided at occurrence of "regex". "regex" is not included in the result.
<code>s.replaceFirst("regex"), "replacement"</code>	Replaces first occurrence of "regex" with "replacement".
<code>s.replaceAll("regex"), "replacement"</code>	Replaces all occurrences of "regex" with "replacement".