Le espressioni regolari (in inglese regular expression, che può trovarsi abbreviata in regexp, regex o RE) sono sintassi attraverso le quali si possono rappresentare insiemi di stringhe. Gli insiemi caratterizzabili con espressioni regolari sono anche detti linguaggi regolari (e coincidono con quelli generabili dalle grammatiche regolari e riconoscibili dagli automi a stati finiti).

E a questo punto immagino le vostre espressioni facciali che sono un misto tra il confuso e l'inebetito … tranqui ogni tanto pure a me le definizioni di wikipedia fanno questo effetto, ma ora vi spiego in parole povere XD .

Immaginate le espressioni regolari come un linguaggio che ha lo scopo di fornire un insieme di strumenti per trovare particolari corrispondenze su una stringa, estrarne una parte specifica e tante altre cosine interessanti che andremo pian piano a vedere insieme .

Avete presente quando dal prompt del dos digitate il comando "dir *.txt" per trovare tutti i file con estensione txt ? Beh avete appena usato una piccola espressione regolare ^^ .

Come ogni linguaggio che si rispetti, le regex hanno le loro strutture semantiche, vale a dire hanno delle "parole chiave" o più precisamente dei simboli che descrivono il linguaggio e che messi insieme formano l'espressione per intero .

Vediamo i costrutti di base e qualche esempio sempre preso da wikipedia .

. Trova ogni singolo carattere (se è nella modalità linea singola altrimenti se è in multiriga prende tutti i caratteri diversi da \n, ovvero un ritorno a capo).

[ ] Trova un singolo carattere contenuto nelle parentesi. Ad esempio, [abc] trova o una "a", "b", o "c". [a-z] è un intervallo e trova ogni lettera minuscola dell'alfabeto. Possono esserci casi misti: [abcq-z] trova a, b, c, q, r, s, t, u, v, w, x, y, z, esattamente come [a-cq-z].Il carattere '-' è letterale solo se è primo o ultimo carattere nelle parentesi: [abc-] o [-abc]. Per trovare un carattere '[' o ']', il modo più semplice è metterli primi all'interno delle parentesi: [][ab] trova ']', '[', 'a' o 'b'.

[^] 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". Ciò che è incluso nell'espressione, può essere richiamato in seguito. Vedi sotto, \n.

\n Dove n è una cifra da 1 a 9; trova ciò che la nesima sottoespressione ha trovato. Tale costrutto è teoricamente irregolare e non è stato adottato nella sintassi estesa delle regexp.

*

* 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 n è una cifra da 1 a 9, trova zero o più iterazioni di ciò che la nesima 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 "*".

{x,y} Trova l'ultimo "blocco" almeno x volte e non più di y volte. Ad esempio, "a{3,5}" trova "aaa", "aaaa" o "aaaaa".

Vecchie versioni di grep non supportano il separatore alternativo “|”.

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

Ovviamente questi sono esempi decisamente banali e poco utili nella pratica, quindi a questo punto mi sembra dovuto introdurre qualche esempio un po più interessante utilizzando il linguaggio di scripting PHP (le regex sono supportate dal 99% dei linguaggi, ma io mi trovo comodo con il PHP :D) .

Il php offre diverse funzioni, più o meno avanzate, per gestire corrispondenze, estrazioni etc tramite le regex, ma per questo tutorial introduttivo noi ci concentreremo su due funzioni in particolare, preg_match e preg_match_all, che useremo rispettivamente per trovare corrispondenze e per l’estrazione dei dati che ci interessano .

preg_match accetta due argomenti principali, l’espressione regolare da utilizzare e la stringa alla quale sottoporla, restituendo in output un numero intero che può essere 0 se non ci sono corrispondenze o 1 se ce n’è almeno una .

Uno degli esempi classici di utilizzo della funzione è il controllo della validità di un email, ovvero data una stringa che *teoricamente* dovrebbe contenere un email, si può usare una regex per controllare se effettivamente quell’indirizzo è valido a tutti gli effetti :

```
<?php

$email = “someone@example.com”;

if( preg_match( “/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$/”,
$email ))Â {
echo “Email valida .”;
}
elseÂ {
echo “Email NON valida !”;
}

?>
```

Da notare che nel primo parametro, oltre alla regex, vengono usati i delimitatori //, questo perchè dopo il secondo / possono essere usati dei flag per determinare il comportamento della regex stessa, ad esempio

```
/regex/i
```

Controllerà delle corrispondenze senza considerare se i caratteri sono maiuscoli o minuscoli, ovvero in modalità case-insensitive . Per gli altri flag vi invito a consultare la documentazione precedentemente linkata della funzione preg_match .

Ok, figo, finqui (spero) niente di eccessivamente difficile e magari può tornare anche utile controllare la validità di un indirizzo email, magari in un modulo di registrazione di un utente … ma non è questo il vero divertimento !

Personalmente, trovo l'estrazione molto più divertente delle corrispondenze, perchè come ho detto prima ci consente di estrarre dati da una qualsiasi fonte filtrandoli in maniera estremamente precisa e, passatemi il termine, "intelliggente" .

Ad esempio, immaginate di dover scrivere, per lavoro, divertimento o puro masochismo, un piccolo script in php che, dato l'indirizzo di una pagina web estragga tutti i link dalla stessa pagina e li stampi in output, ovvero rilevi tutti i tag

testo del link

e per l'appunto ne estragga il link che ci interessa .

Volendo si potrebbe fare a suon di strpos e così via, ma considerando in quante forme diverse si può presentare il tag (basta solo invertire gli attributi href e target per mandare a puttane un algoritmo basato su strpos e affini) e soprattutto considerando che questo è solo un cazzo di esempio XD direi che è meglio usare le regex :D

```
<?php
$sito = "http://www.evilsocket.net/";
// ebbene si, file_get_contents può essere usata anche per richiedere una pagina web :P
$html = file_get_contents($sito);

if( preg_match_all( "/<\s*a\s+[^>]*href\s*=\s*[\"']?([^\"' >]+)[\"' >]/i", $html, $corrispondenze ) ){
/*
la variabile $corrispondenze a questo punto è un vettore bidimensionale .
Nella prima dimensione, ovvero $corrispondenze[0], la funzione preg_match_all
mette le parti di stringa nelle quali ci sono le corrispondenze rilevate,
ovvero i tag <a href="…" per intero, mentre nella seconda dimensione,
ovvero $corrispondenze[1], estrae tramite gli operatori ( ) i link veri e
propri che ci interessano .
*/

print_r($corrispondenze[1]);
}
?>
```

Il che una volta eseguito stamperà in output una cosa del tipo :

```
Array
(
[0] => http://www.evilsocket.net/
[1] => http://www.evilsocket.net/
[2] => http://www.evilsocket.net/category/about-me
[3] => http://www.evilsocket.net/category/hacking
[4] => http://www.evilsocket.net/category/hacking/exploiting
[5] => http://www.evilsocket.net/category/hacking/reversing-hacking
```

[6] => http://www.evilsocket.net/category/hacking/web-hacking
…

….

)

Beh, vi basti pensare che interi web crawler (ovvero i bot sui quali si basano i motori di indicizzazione come google) si basano su questo principio, così come interpreti di linguaggi, gestori di traffico etc etc etc etc per capire, se ancora nn lo aveste capito da soli, quante e quali sono le potenzialità di questo strumento .

Che dire, per questa piccola introduzione è tutto, spero di aver acceso l'interesse per le regexp e di avervi reso le idee un po più chiare in proposito ^^

Per ulteriori riferimenti consiglio questo sito, che oltre ad essere il mio preferito racchiude sempre ciò che ci serve