REGEX - PCRE

CONVENCIÓN

Expresión regular en rojo. regex

Texto al cual le aplico la expresión regular en verde. regex

Coincidencia en celeste regex

LO BÁSICO

Las expresiones regulares utilizan conjuntos de carecteres, clases, rangos y conjuntos de rangos.

Caracteres - Simbolos que actuan como comodines

Ejemplo1:

La letra: a
puede ser usada para encontrar la primer ocurrencia de la letra a dentro del string:
Juan
Lo cual identifica la a que se encuentra antes de la letra n.

Ejemplo2: Meta-caracteres **[\ ^ $. | ? * + ( )**

Los meta-caracteres deben ser escapados si no se quiere que tomen el significado especial que tienen asociados. Por ejemplo si quieron encontrar **1+1=2** la regex correcta debe ser **1\+1=2**
El punto (.):

Clases

La parte de un patrón que está entre corchetes se llama una "clase carácter".
Las clases de caracteres va entre corchetes

[a-z]on	Especificamos letras minusculas delante de **on**
[aeiou]on	Especifica conjuntos de carecteres aeiou delante de **on**
gr[ae]y	Coincide con **gray** o con **grey**

Rangos

Para indicar un rango de caracteres dentro de una clase se utililza un guíon.

[a-z]	Digitos entre 0 y 9

[0-9a-fA-F] Indica un dígito hexadecimal
[0-9a-fA-FX] Indica un dígito hexadecimal o la letra X
q[^x] Identifica un carácter que no sea x, como question. En el caso de Iraq no identifica nada pues no existe carácter luego de la q.

Conjuntos de rangos

a-zA-Z

LIMITADORES

Cuando se usan funciones PCRE, se requiere que el patrón esté encerrado entre ***delimitadores***. Un delimitador puede ser cualquier carácter no alfanumérico, que no sea una barra invertida, y que no sea un espacio en blanco.

Los delimitadores que se usan a menudo son barras oblícuas (**/**), signos de número (**#**) y tildes (**~**). Los siguientes ejemplos son todos patrones delimitados válidos.

```
/foo bar/

#^[^0-9]$#

+php+

%[a-zA-Z0-9_-]%
```

Si el delimitador necesita ser comparado dentro del patrón se debe escapar usando una barra invertida. Si el delimitador aparece a menudo dentro del patrón, es una buena idea escoger otro delimitador para aumentar la legibilidad.

```
/http:\/\//

#http://#
```

La función preg_quote() se puede usar para escapar una cadena para inyectarla dentro de un patrón y su segundo parámetro opcional de puede usar parar especificar el delimitador a escapar.

Además de los delimitadores anteriormente mencionados, también es posible usar delimitadores estilo paréntesis donde los paréntesis de apertura y cierre son el delimitador de inicio y final, respectivamente.

```
{this is a pattern}
```

Puede añadir modificadores de patrón después del delimitador final. El siguiente es un ejemplo de comparación insensible a mayúsculas-minúsculas:

```
#[a-z]#i
```

META-CARACTERES

El poder de las expresiones regulares viene dado por la capacidad de incluir alternativas y repeticiones en el patrón. Éstos están codificados en el patrón por el uso de ***meta-caracteres***, los cuales no se representan a sí mismos, sino que son interpretados de una forma especial.

Hay dos conjuntos diferentes de meta-caracteres: aquéllos que son reconocidos en cualquier lugar de un patrón excepto dentro de los corchetes, y aquéllos que son reconocidos dentro de los corchetes.

Fuera de los corchetes, los meta-caracteres son los siguientes:

****

carácter de escape general con varios usos

^

declaración de inicio de sujeto (o línea, en modo multilínea)

$

declaración de fin de sujeto o antes de la terminación de nueva línea (o fin línea, en modo multilínea)

.

coincide con cualquier carácter excepto con el de nueva línea (por defecto)

[

inicio de la definición de la clase carácter

|

inicio de rama alternativa

(

inicio de sub-patrón

)

fin de sub-patrón

?

extiende el significado de (, también cuantificador 0 ó 1, hace perezosos a los cuantificadores codiciosos (véase repetición)

****

cuantificador 0 o más

+

cuantificador 1 o más

{

inicio de cuantificador mín/máx

}

fin de cuantificador mín/máx

En una clase carácter los únicos meta-caracteres son:

****

carácter de escape general

^

niega la clase, pero sólo si se trata del primer carácter

-

define el rango de caracteres

]

finaliza la clase carácter

Las siguientes secciones describen el uso de cada meta-carácter.

CARACTERES GENÉRICOS

\d cualquier dígito decimal
\D cualquier carácter que no es un dígito decimal
\h cualquier carácter espacio en blanco horizontal (desde PHP 5.2.4)
\H cualquier carácter que no es un carácter espacio en blanco horizontal (desde PHP 5.2.4)
\s cualquier carácter espacio en blanco
\S cualquier carácter que no es un carácter espacio en blanco
\v cualquier carácter espacio en blanco vertical (desde PHP 5.2.4)
\V cualquier carácter que no es un carácter espacio en blanco vertical (desde PHP 5.2.4)
\w cualquier carácter "palabra"
\W cualquier carácter que no es "palabra"

CARACTERES NO IMPRIMIBLES

\a alarma, es decir, el carácter BEL (07 hex)
\cx "control-x", donde x es cualquier carácter
\e escape (1B hex)

\f salto de página (0C hex)
\n nueva línea (0A hex)
\p{xx} un carácter con la propiedad xx, véase propiedades unicode para más información
\P{xx} un carácter sin la propiedad xx, véase propiedades unicode para más información
\r retorno de carro (0D hex)
\t tabulador (09 hex)
\xhh carácter con el código hexadecimal hh
\ddd carácter con el código octal ddd, o retro-referencia

OTRO USO DE LA BARRA INVERTIDA

Una declaración especifica una condición que se debe encontrar en un punto particular de una comparación, sin consumir ningún carácter de la cadena objetivo. El uso de sub-patrones para declaraciones más complicadas se describe después. Las declaraciones de la barra invertida son:

\b límite de palabra
\B distinto a límite de palabra
\A comienzo del sujeto (independientemente del modo multi-linea)
\Z fin del sujeto o nueva línea al final (independientemente del modo multi-linea)
\z final del sujeto (independientemente del modo multi-línea)
\G primera posición de coincidencia del sujeto

CARACTERES UNICODE

A partir de 5.1.0, están disponibles tres secuencias de escape adicionales para comparar tipos de caracteres genéricos cuando el ***modo UTF-8*** está seleccionado. Son:

\p{xx} un carácter con la propiedad xx
\P{xx} un carácter sin la propiedad xx
\X una secuencia Unicode extendida
Los nombres de las propiedades representadas arriba por ***xx*** están limitadas a las propiedades de la categoría general de Unicode. Cada carácter tiene exactamente una propiedad, especificada por una abreviatura de dos letras. Por compatibilidad con Perl, la negación se puede especificar incluyendo un acento circunflejo entre la llave de apertura y el nombre de la propiedad. Por ejemplo, ***\p{^Lu}*** es lo mismo que ***\P{Lu}***.

Si sólo se especifica una letra con **\p** o **\P**, se incluyen todas las propiedades que comienzan con esa letra. En este caso, en la ausencia de negación, las llaves en la secuencia de escape son opcionales; estos dos ejemplos tienen el mismo efecto:

```
\p{L}

\pL
```

Códigos de propiedades soportados		
Propiedad	**Coincidencias**	**Notas**
C	Otro	
Cc	Control	
Cf	Formato	
Cn	Sin asignar	
Co	Uso privado	
Cs	Sustituto	
L	Letra	Incluye las siguientes propiedades: ***Ll***, ***Lm***, ***Lo***, ***Lt*** y ***Lu***.
Ll	Letra minúscula	
Lm	Letra modificadora	
Lo	Otra letra	
Lt	Letra de título	
Lu	Letra mayúscula	
M	Marca	
Mc	Marca de espacio	
Me	Marca de cierre	
Mn	Marca de no-espacio	
N	Número	
Nd	Número decimal	
Nl	Número letra	
No	Otro número	
P	Puntuación	
Pc	Puntuación de conexión	
Pd	Puntuación guión	
Pe	Puntuación de	

	cierre	
Pf	Puntuación final	
Pi	Puntuación inicial	
Po	Otra puntuación	
Ps	Puntuación de apertura	
S	Símbolo	
Sc	Símbolo de moneda	
Sk	Símbolo modificador	
Sm	Símbolo matemático	
So	Otro símbolo	
Z	Separador	
Zl	Separador de línea	
Zp	Separador de párrafo	
Zs	Separador de espacio	

EL PUNTO

El punto detecta un carácter individual, excepto caracteres de salto de línea. Es la abreviatura de [^ \ n] (UNIX regex flavors) o [^ \ r \ n] (Windows regex flavors). Mayoría de los motores de expresiones regulares tienen un "punto coincide con todos" o el modo "línea única" que hace que el punto coincide con cualquier carácter, incluyendo saltos de línea.

gr.y coincide gris, gr% interanual, etc Utilice el punto moderación. A menudo, un carácter de clase o la clase de caracteres negada es más rápido y más preciso.

Anclas - circunflejo (^) y). Dólar ($)

Fuera de una clase carácter, en el modo de comparación por defecto, el carácter **circunflejo** (**^**) es una declaración que es verdadera sólo si el punto de coincidencia actual está en el inicio de de la cadena objetivo. Dentro de una clase carácter, circunflejo (**^**) tiene un significado totalmente diferente (véase más adelante).

Circunflejo (**^**) no necesita ser el primer carácter del patrón si están implicadas varias alternativas, pero debería ser la primera cosa en cada alternativa en la que aparece si el patrón es comparado siempre con esa rama. Si todas las posibles alternativas comienzan con un circunflejo (**^**), es decir, si el patrón es obligado a coincidir sólo con el comienzo de la cadena objetivo, se dice que el patrón está "anclado". (También hay otras construcciones que pueden causar que un patrón esté anclado.)

Un carácter dólar (***$***) es una declaración la cual es **TRUE** sólo si el punto actual de coincidencia está al final de la cadena objetivo, o inmediatamente antes de un carácter de nueva línea que es el último carácter en la cadena (por defecto). Dólar (***$***) no necesita ser el último carácter del patrón si están implicadas varias alternativas, pero debería ser el último elemento en cualquier rama en la que aparezca. Dólar no tiene un significado especial en una clase carácter.

Alternancia – Es la barra vertical - |

Los caracteres barra vertical se usan para separar patrones alternativos. Por ejemplo, el patrón ***gilbert|sullivan*** coincide con "gilbert" o con "sullivan". Pueden aparecer cualquier número de alternativas, y se permite una alternativa vacía (coincidiendo con la cadena vacía). El proceso de comparación prueba cada alternativa sucesivamente, de izquierda a derecha, y la primera que tenga éxito se usa.

Ejemplo:

cat|dog	Expresión regular
About cats and dogs	Cadena en la cual va a buscar

cat	Coincidencia
dog	Coincidencia si la busqueda se aplica nuevamente

Sub-patrones – ( )

Los sub-patrones están delimitados por paréntesis, los cuales pueden estar anidados. Localiza un conjunto de alternativas.

Por ejemplo:

cata(rata\|pulta\|)

Con parentesis coincide con una de las palabras "cata", "catarata", o "catapulta".
Sin los paréntesis, coincidiría con "catarata", "pulta" o la cadena vacía.

Repetición

La repetición se especifica mediante cuantificadores.

Por ejemplo:

z{2,4} coincide con "zz", "zzz", o "zzzz". Una llave de cierre por sí misma no es un carácter especial.
[aeiou]{3,} coincide al menos con 3 vocales sucesivas, pero puede coincidir con muchas más
\d{8} coincide exactamente con 8 dígitos.

Cuantificadores de carácter simple	
*****	equivale a ***{0,}***
+	equivale a ***{1,}***
?	equivale a ***{0,1}***

colou?r identifica colour o color
\b[1-9][0-9]{3}\b Identifica un número entre 1000 y 9999
\b[1-9][0-9]{2,4}\b Identifica un número entre 100 y 99999
<[A-Za-z][A-Za-z0-9]*> Identifica una etiqueta html sin atributos
<[A-Za-z0-9]+> Puede identificar etiquetas invalidas como <1>

LOOKAROUND

Es un tipo especial de grupo.

q(?=u) Identifica la q en question pero no en Iraq. La u no es parte de la expresión regular. Se identifica cualquier q previo a una u en un string.
q(?!u) Identifica la q en Iraq pero no en question. El signo ! actua como negación
(?<=a)b Identifica la letra b en abc

RETRO-REFERENCIAS

Fuera de una clase carácter, una barra invertida seguida por un dígito mayor que 0 (y posiblemente más dígitos) es una retro-referencia a un sub-patrón de captura

anterior (esto es a su izquierda) del patrón, siempre que hayan habido tantas capturas entre paréntesis previas a la izquierda.

(abraz|apreci)o de un \1ador coindice con "abrazo de un abrazador" y con "aprecio de un apreciador", pero no con "abrazo de un apreciador".

DECLARACIONES

Una declaración es una comprobación de los caracteres siguientes o anteriores al punto de coincidencia actual que en realidad no consumen carácter alguno. Las declaraciones simples codificadas como \b, \B, \A, \Z, \z, ^ y $ están descritas anteriormente.

Las declaraciones más complicadas están codificadas como sub-patrones. Hay dos tipos:

- Aquéllas que *buscan hacia delante* desde punto actual de la cadena objetivo, y
- Aquéllas que *buscan hacia atrás* desde él.

Busqueda hacia adelante (?= (?!

Las declaraciones de *búsqueda hacia delante* comienzan con (?= para declaraciones positivas y con (?! para declaraciones negativas.

Por ejemplo,

\w+(?=;) coincide con una palabra seguida de un punto y coma, pero no incluye el punto y coma en la coincidencia, y

foo(?!bar) coincide con cualquier incidencia de "foo" que no esté seguida de "bar".

Busqueda hacia atras (?<= (?<!

Las declaraciones de *búsqueda hacia atrás* comienzan con (?<= para declaraciones positivas y con (?<! para declaraciones negativas.

Por ejemplo,

(?<!foo)bar encuentra una incidencia de "bar" que no esté precedida por "foo". El contenido de una declaración de búsqueda hacia atrás está restringido de tal manera que todas las cadenas que se comparen con ella deben tener una longitud fija. Sin embargo, si hay varias alternativas, no es necesario que tengan todas la misma longitud fija. Así, (?<=buey|burro) está permitido, pero (?<!toros?|vacas?) produce un error en tiempo de compilación.

SUB-PATRONES DE UNA SÓLA APLICACIÓN (?>

Con las repeticiones maximizadoras y minimizadoras, el fallo de lo que se encuentra a continuación causa normalmente que la repetición del elemento sea re-evaluada para ver si un número diferente de repeticiones permite que el resto del patrón coincida. A veces es útil prevenir esto, tanto para cambiar la naturaleza de la comparación, como para ocasionar que falle antes (ya que de otra manera ocurriría el fallo después) cuando el autor del patrón sabe que no tiene sentido seguir.

Considere, por ejemplo, el patrón \d+foo cuando se aplica a la línea objetivo ***123456bar***

Después de coincidir los 6 dígitos y luego fallar al coincidir con "foo", la acción habitual del comparador es volver a intentarlo con sólo 5 dígitos, coincidiendo con el elemento \d+, y después con 4, y así sucesivamente, antes de que, por último, falle. Los sub-patrones de una sóla aplicación proporcionan el medio para especificar que una vez que una porción del patrón ha coincidido, no será re-evaluado de esta forma, por lo que el comparador se rendirá inmediatamente al fallar la comparación de "foo" la primera vez. La notación es otro tipo de paréntesis especiales, comenzando con (?> como en este ejemplo: *(?>\d+)bar*

Este tipo de paréntesis "bloquea" la parte del patrón que lo contiene una vez haya coincidido, y un fallo dentro del patrón evita que éste retroceda sobre sí mismo. El retroceso hacia elementos previos al sub-patrón funciona normalmente, después de todo.

SUB-PATRONES CONDICIONALES

Es posible hacer que el proceso de comparación obedezca a un sub-patrón condicionalmente o que elija entre dos sub-patrones alternativos, dependiendo del resultado de una declaración o de si un sub-patrón de captura previo coincidió o no. Las dos formas posibles de un sub-patrón condicional son

```
(?(condición)patrón-sí)
(?(condición)patrón-sí|patrón-no)
```

Si la condición es satisfecha, se usa el patrón-sí; de otro modo se usa el patrón-no (si está presente). Si hay más de dos alternativas en el sub-patrón, se producirá un error en tiempo de compilación.

Hay dos tipos de condiciones. Si el texto entre los paréntesis consisite en una secuencia de dígitos, la condición es satisfecha si el sub-patrón de captura de ese número ha coincidido anteriormente.

Considere el siguiente patrón, el cual contiene espacios en blanco no significativos para hacerlo más legible (se asume la opción PCRE_EXTENDED) y para dividirlo en tres partes para facilitar su discusión: *(\( )? [^()]+ (?(1) \))*

(\( )?

La primera parte compara un paréntesis de apertura opcional, y si el carácter está presente, lo establece como la primera sub-cadena capturada.

[^()]+

La segunda parte compara uno o más caracteres que no sean paréntesis.

(?(1) \))

La tercera parte es un patrón condicional que examina el primer conjunto de paréntesis coincididos o no. Si lo fueron, es decir, si el sujeto comenzó con un paréntesis de apertura, la condición es TRUE, y así el patrón-sí se ejecuta y es requerido un paréntesis de cierre. De otra manera, ya que el patrón-no no está presente, el subpatrón no coincidirá con nada. En otras palabras, este patrón coincide con una secuencia que no tenga paréntesis, opcionalmente encerrada entre paréntesis.

Si la condición es la cadena *(R)*, se satisface si ha sido hecha una llamada recursiva al patrón o sub-patrón. En el "nivel superior", la condición es falsa.

Si la condición no es una secuencia de dígitos o (R), debe de ser una declaración. Ésta puede ser una declaración de búsqueda hacia delante o hacia atrás, negativa o positiva. Considere este patrón, conteniendo de nuevo espacios en blanco no significativos, y con dos alternativas en la segunda línea:

```
(?(?=[^a-z]*[a-z]) \d{2}-[a-z]{3}-\d{2}  |  \d{2}-\d{2}-\d{2} )
```

La condición es una declaración de búsqueda hacia delante positiva que compara una secuencia opcional de algo que no sean letras seguida de una letra. En otras palabras, comprueba la presencia de al menos una letra en el sujeto. Si se encuentra una letra, el sujeto se compara con la primera alternativa; de otro modo se compara con la segunda. Este patrón coincide con cadenas en una de las dos formas dd-aaa-dd o dd-dd-dd, donde aaa son letras y dd son dígitos.

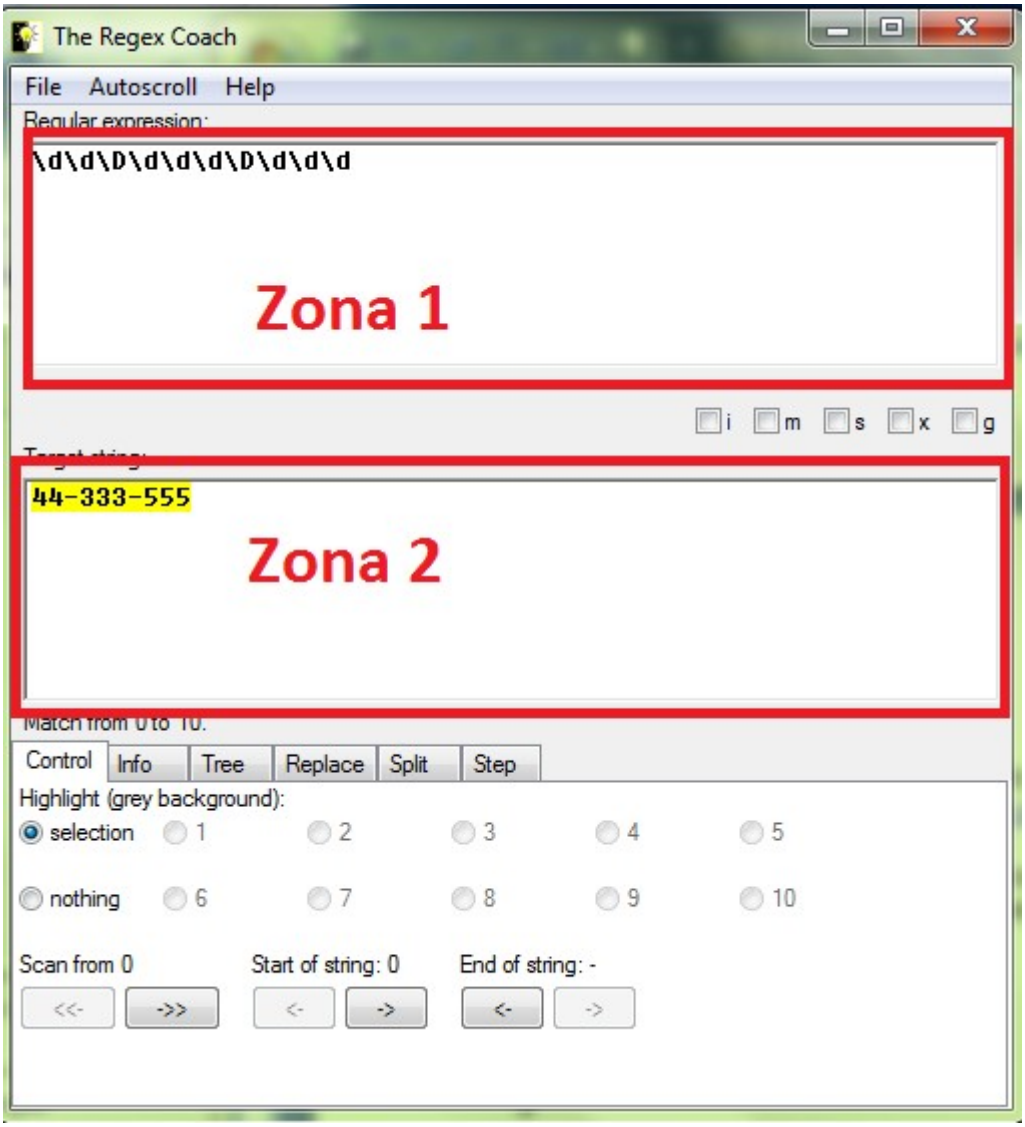
Comentarios (?#

La secuencia (?# marca el inicio de un comentario que continúa hasta el siguiente paréntesis de cierre. Los paréntesis anidados no están permitidos. Los caracteres que componen un comentario no toman parte a la hora de una comparación del patrón.

Herramienta de trabajo

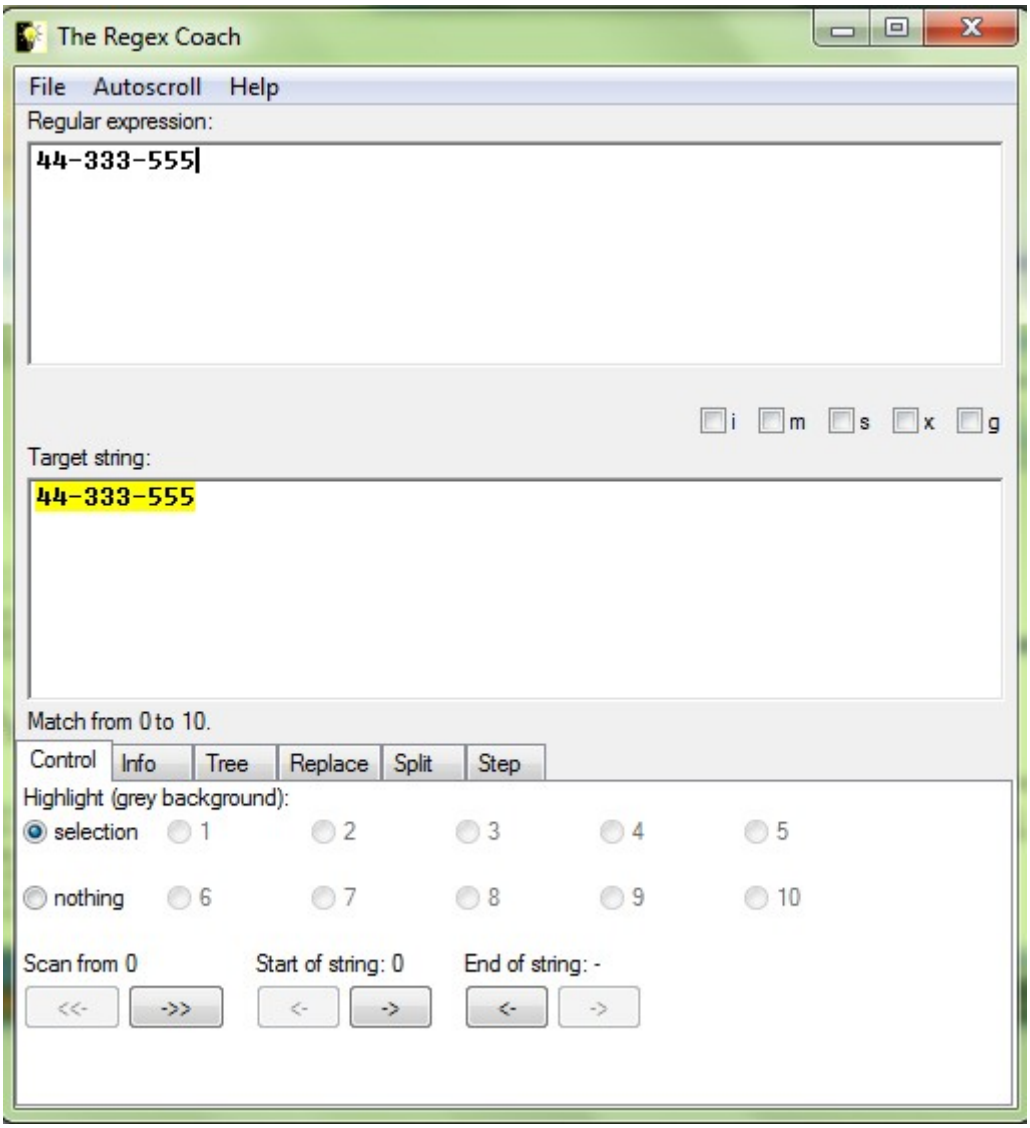
Para trabajar con expresiones regulares, primer vamos a descargar el software “The Regex Coach” del sitio http://www.weitz.de/regex-coach/

Este software tiene dos áreas, la primera (las superior) es donde se escribe la expresión regular, mientras que en la segunda se pone el texto en el cual se va a buscar mediante la expresión regular.



Utilizando String literal

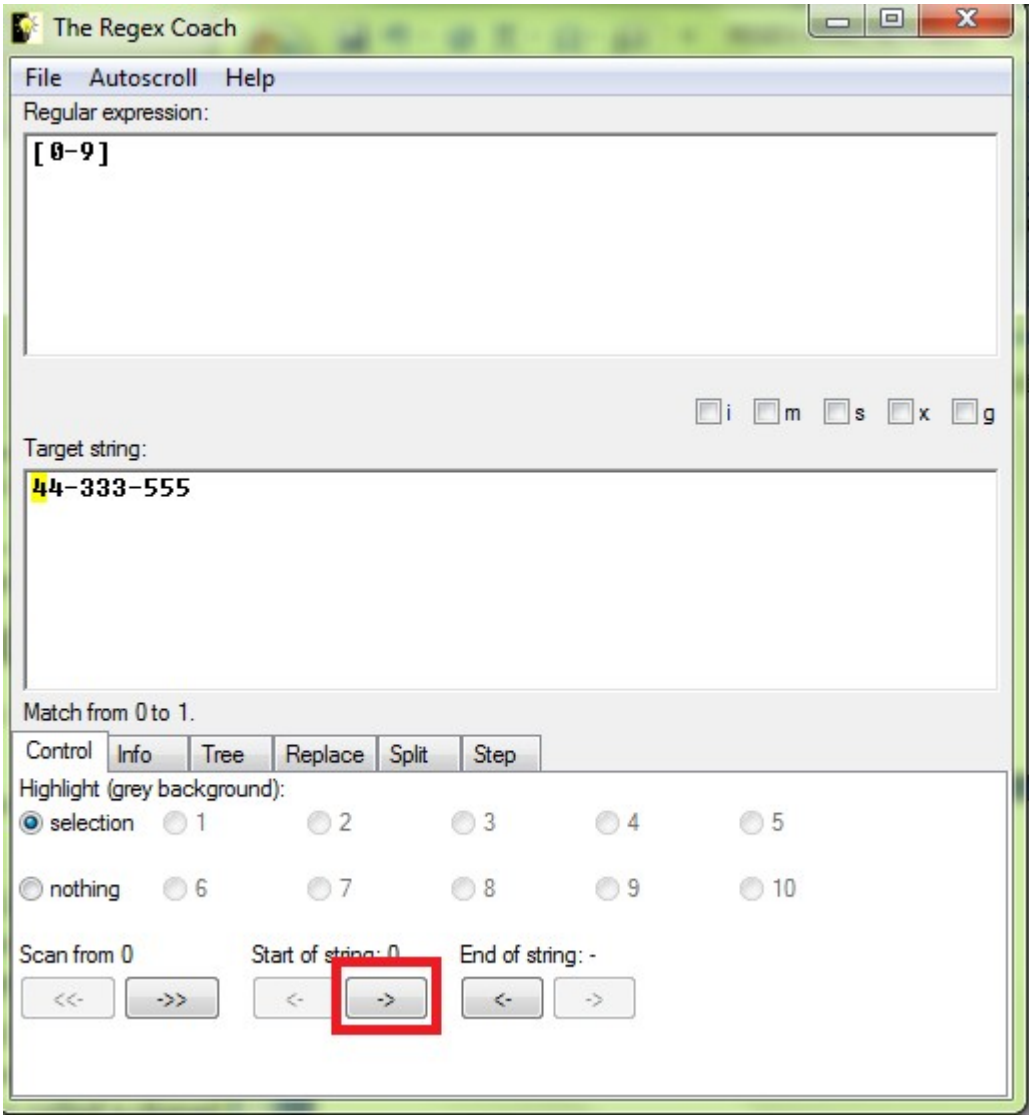
Para comenzar vamos a utilizar un string literal, es decir vamos a buscar un string, con el mismo string que estamos buscando. Supongamos que queremos encontrar el telefono: 44-333-555, si colocamos este número en las zonas 1 y 2, vemos que en la zona 2 el número aparece destacado con color amarillo, lo cual identifica que la expresión regular escrita en la zona 1 ha encontrado una coincidencia en la zona 2.



Utilizando clase caracter

Si utilizamos la clase carecter [0-9] que representa un digito entre 0 y 9, podemos ver que la primer coincidencia es el número 4. Presionando el botón de siguiente, podemos ver el resto de los caracteres identificados por esta clase carácter en el orden que van apareciendo.

También podriamos escribir [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] para identificar los números del telefono uqe compone el estring de la zona 2.



Otra forma de representar esto es mediante el uso de los caracteres genericos:

\d números decimales
\D cualquier carecter que no es dígito decimal

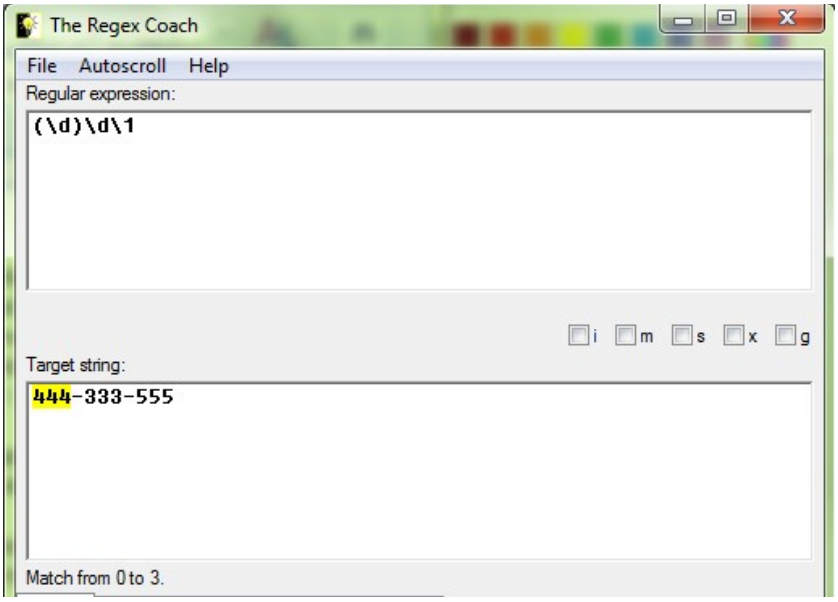
\d\d\D \d\d\d\D \d\d\d

O mediante el uso del punto que representa cualquier carácter:

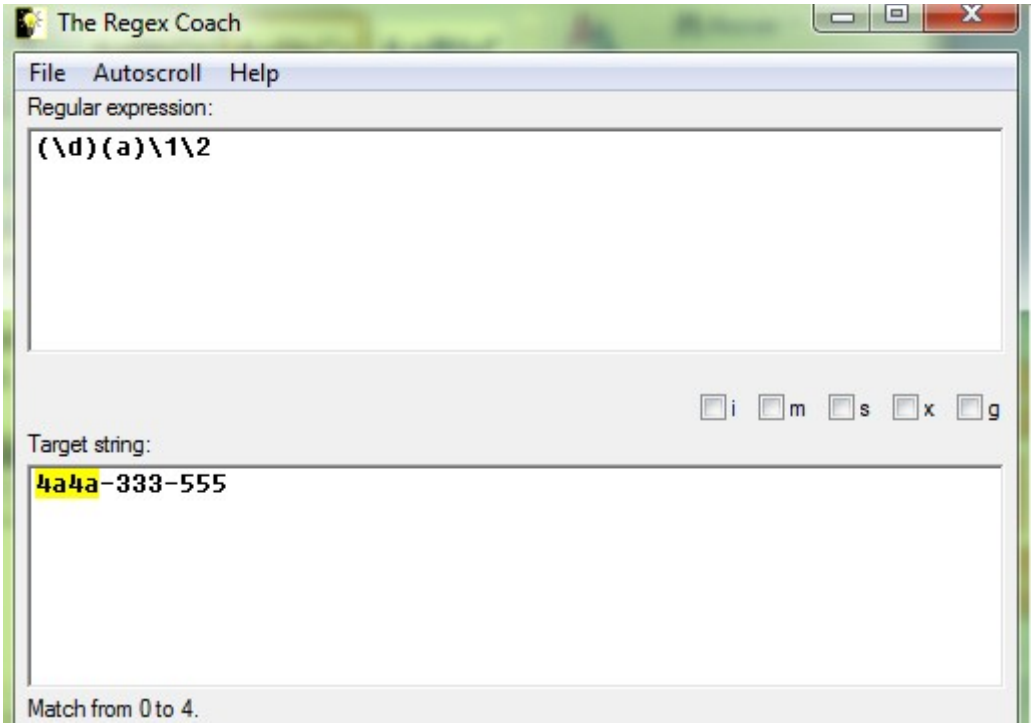
\d\d. \d\d\d. \d\d\d

Captura de grupo y referencia hacia atras

Ahora vamos a coincidir sólo una parte del número de teléfono utilizando lo que se conoce como un grupo de captura. A continuación nos referiremos al contenido del grupo con una referencia inversa. Para crear un grupo de captura, escriba a \ d en un par de paréntesis, a colocarlo en un grupo, y luego seguir con una referencia hacia atrás \ 1 a lo adquirido: (\ d) \ d \ 1
El \ 1 remite a lo captado en el grupo encerrado entre paréntesis. Como resultado, esta expresión regular coincide con el prefijo 444. He aquí un desglose de la misma:
• (\ d) coincide con el primer dígito y lo captura (el número 7)
• \ d coincide con el siguiente dígito (el número 0), pero no lo captura, ya que no esta entre paréntesis
• \ 1 hace referencia al dígitos capturado (el número 4) con lo cual el tercer númer debe ser igual que el primero.



Si escribimos (\d)(a)\1\2, podemos encontrar por ejemplo el string “4a4a” en donde \1 esta haciendo referencia al primer grupo capturado, en este caso el número “4” y el \2 hace referencia al segundo grupo, que en este caso representa solo el carácter “a”.



Uso de cuantificadores

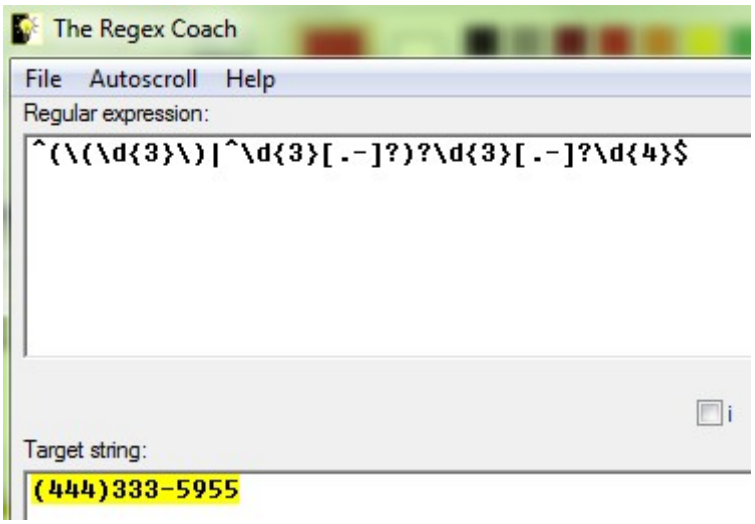
Podriamos usar cuantificadores, y por ejemplo utilizar la siguiente expresión regular para identificar todo el número:

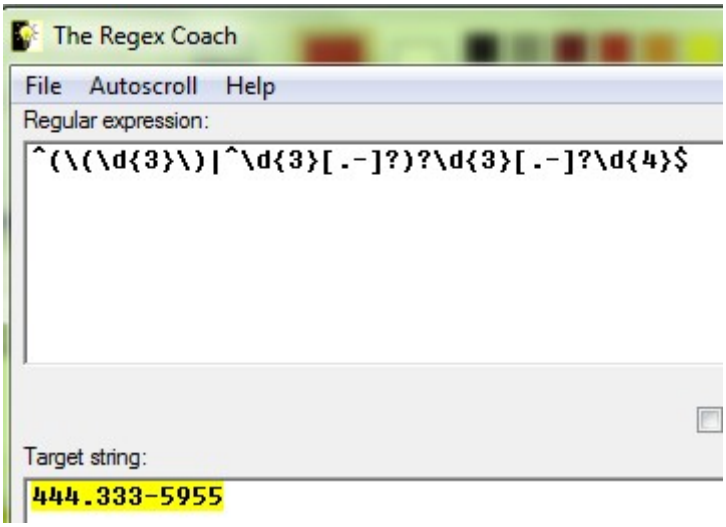
\d{3}-?\d{3}-?\d{3}

Un ejercicio un poco más complejo

Podemos ahora realizar una selección un poco más compleja, que nos permite seleccionar los primeros tres digitos entre parentesis, o separados por punto o guión. El simbolo **^** representa un comienzo de línea mientras que el simbolo de $ indica el final de la línea.

El signo de interrogación indica cero o una vez.





FUNCIONES PHP

preg_grep

(PHP 4, PHP 5)

preg_grep — Devuelve entradas de matriz que coinciden con el patrón

Descripción

array preg_grep (string `$pattern` , array `$input` [, int `$flags` = 0])

Devuelve la matriz consistente en los elementos de la matriz *`input`* que coinciden con *`pattern`*.

Ejemplo 1 – USO DEL PUNTO – ARCHIVO: regex-punto.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/.on/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/.on/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) .on debe llevar delimitdores por lo que se debe escribir como /.on/
2) El resultado nos retorna los elementos de la matriz que coinciden con la busqueda, por lo que en este caso nos retorna el primer elemento ya que tiene la palabra (c**on**) y el siguiente elemento ya que tiene la palabra (alf**on**sín)

Resultado:

La cadena buscada se encuentra en el stringArray (
[0] => con las manos en la mesa
[1] => la casa esta en orden dijo alfonsín, esa)

Ejemplo 2 – USO DE CLASE DE CARACTERES – ARCHIVO: regex-clase.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
```

```
alfonsín, esa");
if(preg_grep("/[a-z]on/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/[a-z]on/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) [a-z]on debe llevar delimitdores por lo que se debe escribir como /[a-z]on/
2) El resultado nos retorna los elementos de la matriz que coinciden con la busqueda, por lo que en este caso no va a retornar el primer elemento ya que tiene la palabra (C**on**) la cual comienza con mayúscula, perso si terona el siguiente elemento ya que tiene la palabra (alf**on**sín) en donde (on) esta presedida por la letra (f) minúscula.

Resultado:

La cadena buscada se encuentra en el stringArray (
[1] => la casa esta en orden dijo alfonsín, esa)

Ejemplo 3 – USO DE CLASE DE CARACTERES – ARCHIVO: regex-clase2.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/[aeiou]on/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/[aeiou]on/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) [aeiou]on debe llevar delimitdores por lo que se debe escribir como /[aeiou]on/
2) El resultado no retorna coincidencias

Resultado:

La cadena buscada no se encuentra en el string

Ejemplo 4 – USO DE CLASE DE CARACTERES – ARCHIVO: regex-clase3.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
//if(preg_match("[[:alnum:]]on",$ejemplo)){
if(preg_grep("/[a-zA-Z]on/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/[a-zA-Z]on/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) [a-zA-Z]on debe llevar delimitdores por lo que se debe escribir como /[a-zA-Z]on/

Resultado:

La cadena buscada se encuentra en el stringArray (
[0] => con las manos en la mesa
[1] => la casa esta en orden dijo alfonsín, esa)

Ejemplo 5 – NEGAR CLASE – ARCHIVO: regex-negar-clase.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/[^A-Z]on/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/[^A-Z]on/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) Usar ^ dentro de una clase, niega el contenido de la clase /[^A-Z]/

Resultado:

La cadena buscada se encuentra en el stringArray (
[1] => la casa esta en orden dijo alfonsín, esa)

Ejemplo 6 – SUBEXPRESIONES – ARCHIVO: regex-subexpresiones.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/(las)*manos/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/(las)*manos/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) El parentesis indica un subestring por lo que la busqueda va a coincidir con:
   manos
   las manos

2) Si usaramos el simbolo de + deberiamos agregar por lo menos una vez la palabra (las) para que retorne un resultado verdadero.

Resultado:

La cadena buscada se encuentra en el stringArray (
[0] => Con las manos en la mesa)

Ejemplo 7 – NÚMERO DE REPETICIONES – ARCHIVO: regex-rep.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/(las){2,3}/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/(las)*{2,3}/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) Con las llaves, indico el número de repeticiones. En el caso de poner {2,3} indicamos que (las) debe repetirse entre 2 y 3 veces, por lo que en este caso no existirán coincidencias.

Resultado:

La cadena buscada no se encuentra en el string

Ejemplo 8 – NÚMERO DE REPETICIONES – ARCHIVO: regex-inicio.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/^las/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/^las/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) Con ^ fuera de una clase carácter, indicamos que la palabra que sigue debe encontrarse al inicio. Por lo que **"/^las/"** no retorna coincidencia, pero **"/^Con/"** si retorna coincidencia.

Resultado:

La cadena buscada no se encuentra en el string

Ejemplo 9 – NÚMERO DE REPETICIONES – ARCHIVO: regex-fin.php

```
<?php
$ejemplo=array("Con las manos en la mesa","la casa esta en orden dijo
alfonsín, esa");
if(preg_grep("/mesa$/", $ejemplo)){
        echo "La cadena buscada se encuentra en el string";
        $output = preg_grep("/mesa$/", $ejemplo);
                print_r( $output );
}else
{
        echo "La cadena buscada no se encuentra en el string";
}
```

Observaciones:

1) Con $ indicamos que la palabra o letra que se encuentra antes, debe encontrarse al final.

Resultado:

La cadena buscada se encuentra en el stringArray (
[0] => Con las manos en la mesa)

Ejemplo: Validar código

```
/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
```

/ Delimitador de apertura

^ Inicio de expresión regular

\w cualquier carácter "palabra"
\. Es el punto escapado

[\w-\.] cualquier carácter "palabra", o guión
[\w-\.]+ cualquier carácter "palabra", o guión repetidos uno o más veces antes de
[\w-\.]+@ cualquier carácter "palabra", o guión repetidos uno o más veces antes de @

Ejemplos: pepe@ - pepe89@

##

([\w-]+\.) Es un subcaracter con cualquier carácter “palabra” o guión repetido 1 o más veces, seguido de punto

[\w-]{2,4} cualquier carácter "palabra", o guión repetido entre 2 a 4 veces

([\w-]+\.) + [\w-]{2,4}

Ejemplo: gmail.com – gmail-gmail.com – gmail3.c-m - etc

$ Fin de expresión regular

/ Cierre de delimitador

([\w-]+\.) + [\w-]{2,4}$/ Esta cadena debe encontrarse al final

##

/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;

Ejemplo completo: pepe@gmail.com – pepe89@g-mail.c-m