RamónInvarato
8 - Feb - 2015

Categoría:

Lenguajes de Programación

Tags:

Busqueda de patrones

Coincidencia de patrones

Delimitadores

Búsqueda de patrones – Expresiones regulares

La búsqueda o coincidencia de patrones (Pattern matching) está ideada para ahorrarte mucho tiempo y realizar procesos de una manera muy optimizada. Con los patrones podremos buscar algo, remplazar por todas las coincidencias por otra cosa, contar lo que se ha encontrado, ubicar la posición de lo que buscamos, entre otras ventajas. Lo mejor, prácticamente todos los lenguajes de programación lo soportan, funcionando en todos igual. Veremos ejemplos de expresiones regulares tanto en PHP, como en JavaScript, así como en Java.

Vamos a empezar poniendo la novena definición de patrón de la RAE (ni la primera, ni la segunda, ni tampoco la octava. La novena definición es la que nos interesa en el contexto en el que trabajamos):

> Modelo que sirve de muestra para sacar otra cosa igual.

Un patrón es un modelo, una plantilla, un molde, un diseño, que nos permite obtener varias cosas idénticas; es decir, es donde nos fijamos para copiar (como quien pinta un bodegón en un lienzo; donde el bodegón copiado con los objetos reales –el plátano, la jarra, el pan, las uvas- es el patrón). La “cosa” que indica la definición no tiene por qué ser solo un objeto físico, sino que puede ser un texto, una imagen, una idea, un modo de vivir, etc.

Y la otra palabra que utilizamos es “búsqueda”. Juntamos las dos y nos sale “Búsqueda de patrones” del inglés “Pattern matching” (“matching” viene del verbo “to match”, que significa algo que coincide; es decir “Coincidir con un patrón”, sinónimo a "Búsqueda de patrones"). Imagínate un cajón lleno de juguetes, y queremos buscar los juguetes coincidentes con un patrón, que puede ser un juguete con forma de barco; por lo que del cajón solo buscaremos los barcos: un barco pirata, un transatlántico, una piragua, etc. Esto es

Diagramas de estados

Ejemplos

Expresiones Regulares

java javascript

Patrones

Pattern matching

php

Reconocimiento de patrones

Regex Regexp

"Busqueda de patrones".

Nos hemos entretenido mucho, lo que nos interesa. Ahora vamos a enfocarnos en la programación, en mayoría suelen ser textos a analizar. Un ejemplo, nos va llegando la siguiente lista de textos (pueden ser los que queramos):

- ¡Hola Mundo!
- miCorreo@gmail.com
- La teoría de "Pattern Machine" dice...
- correoFalso@yahoo.es
- En un lugar de la Mancha, cuyo nombre no quiero acordarme...
- +34 91 123 456 789
- estoNOesUnCorreoNoTieneArroba.com
- RaMoN@jarroba.com
- Calle Alcalá 12345 Madrid, Madrid

Y nuestro patrón es "las direcciones de correos electrónicos". Hay que buscar las direcciones de correos electrónicos en el listado anterior. Dicho esto, si vuelves al listado de textos, seguro que sacas todos los correos electrónicos que hay en unos segundos. Habrás descubierto que hay 3 direcciones de correos ("miCorreo@gmail.com", "correoFalso@yahoo.es", y "RaMoN@jarroba.com"); hay uno que parece una dirección de correo, pero no lo es, le falta la arroba ("estoNOesUnCorreoNoTieneArroba.com"). Tú mismo has aplicado "Búsqueda de patrones".

Parafraseando a Neil deGrasse Tyson –astrónomo espectacular- en el documental "Cosmos" en doblaje español:

> *El talento humano para reconocer patrones es un arma de doble filo:* ***somos especialmente buenos encontrando patrones****, aun cuando realmente no están ahí.*

Aunque el párrafo en sí tiene mucho más significado dentro del guión del documental, he marcado en

negrita lo que me interesa hacerte llegar “Somos especialmente buenos encontrando patrones”. Y quiero hacerte llegar lo fácil que es para nosotros utilizar esta herramienta. Se acabó eso de huir del mundo del “Pattern matching” (no sabes la cantidad de gente que evita su utilización alegando dificultad) , ya que es nuestro mundo, nuestro modo de aprendizaje, nuestro cerebro, nuestros pensamientos, lo que somos.

Tenemos que desmenuzar y decirle al ordenador que haga lo mismo que ha hecho nuestro cerebro. Le tenemos que decir que solo queremos las direcciones de correos electrónicos. Como tu cerebro es muy bueno encontrando patrones, seguro que te habrás dado cuenta de cuál es la estructura de un correo electrónico.

Habrás deducido que un correo electrónico está formado por un texto cualquiera, seguido de un símbolo de arroba, que le continua un texto cualquiera, luego un punto, y termina con otro texto cualquiera.

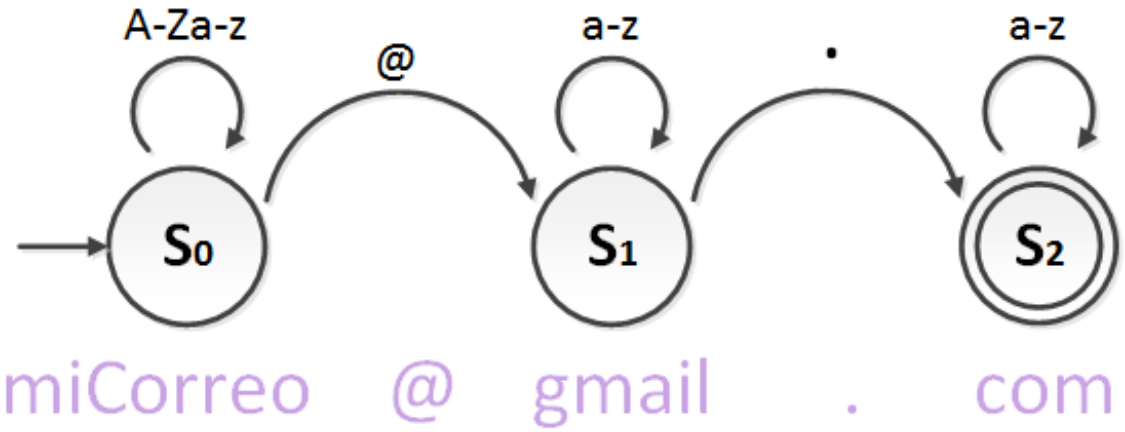
Nota para las urgencias: Si tienes prisa puedes ir directamente a la parte de “Expresiones Regulares”. Explico los “Diagramas de estados” porque es una herramienta muy útil, es muy sencillo; y en el caso que nos ocupa, es muy útil para desgranar una expresión regular. También explico la diferencia de “Reconocimiento de patrones versus Búsqueda de patrones”, tampoco es necesario para entender cómo se utilizan las expresiones regulares.

Diagrama de estados

Es fácil crear el **diagrama de estados o autómata finito** del patrón de correos electrónicos. No es

necesario en casos simples (cuando tengas un poco de práctica salen las expresiones regulares de cabeza), en casos complejos te aseguro que ayuda mucho.

El siguiente tiene muy buena pinta:



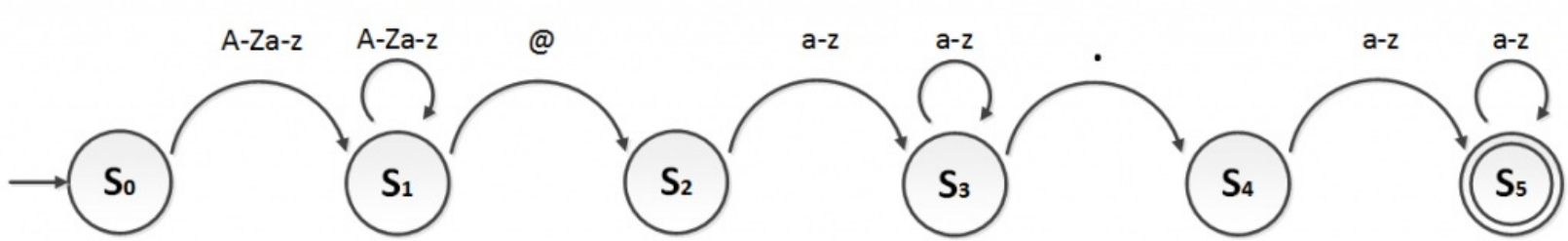
Puede que nunca hayas visto un diagrama de estados o autómata finito. En este pequeño párrafo te voy a explicar todo lo que hay que saber sobre diagramas de estados. La flecha de la izquierda indica el inicio. El círculo con la S simboliza el estado con su número (el número es para poner al estado un nombre único, un id; suelen ponerse en orden de lectura, aunque no es obligatorio, y cuando el diagrama de estados tiene varios caminos resulta imposible poner un orden). Las flechas indican qué es lo que se tiene que cumplir para pasar de un estado a otro, o permanecer en el mismo. El círculo con otro círculo dentro indica que ese estado será el final. Puede haber varios estados finales, así como varios inicios. El estado final es el objetivo, en caso de que lleguemos habremos reconocido lo que sea con éxito.

En el ejemplo, analizo la dirección de correo “miCorreo@gmail.com”. Empiezo por la flecha de entrada y llego al S0 por la primera letra “m”. Encima del estado S0 he puesto una flecha cíclica “A-Za-z” que indica que reconocerá cualquier letra una a una (“a-z” significa letras desde la “a” hasta la “z”, todo el abecedario), tanto en mayúsculas "A-Z" como en minúsculas "a-z". La flecha cíclica sobre el estado S0 siempre va a indicar que nos puede llegar entre cero letras e infinitas letras. Analizamos la letra “o” del final de la palabra “miCorreo” y nos

encontramos con una “@”. El símbolo arroba "@" no pertenece al conjunto “A-Za-z”, pero sí que tiene otro camino. En cuanto llegamos a la arroba “@”, saltaremos al siguiente estado S1. En S1 reconoceremos solo letras minúsculas del conjunto “a-z”. Cuando lleguemos al punto “.” saltaremos al estado S2. En S2 igualmente reconoceremos cualquier letra minúscula. Como S2 está marcado como estado final, si se nos han acabado los caracteres a comprobar en este estado final, significa que nuestra dirección de correo electrónico ha sido validada. Para ver que el diagrama de estados cumple su función de validar direcciones de correos electrónicos, hay que analizar varios. Por ejemplo “RaMoN@jarroba.com” comprobaremos que también cumple. Si intentamos analizar “estoNOesUnCorreoNoTieneArroba.com”, nunca llegaríamos al estado final S2 -pues no hay arroba- por lo que no pasaría la validación.

Aunque bien mirado el diagrama de estados anterior, permite unas salidas no deseadas 🙁

Por ejemplo, permite el correo “@.”. ¡Sí, vacío!, solo la arroba y el punto. El anterior diagrama lo hice simple para entender la idea de un vistazo rápido, y ver cómo serían los ciclos entre cero y muchos; ahora vamos a complicarlo un poco para hacer ciclos entre uno y muchos. Para solventar este problemilla, lo que se puede hacer es forzar a que al menos haya una letra o más en cada parte (antes de la arroba, después de la arroba y después del punto).



Ahora sí que sí 🙂

Nuestro diagrama de estados permite direcciones de correos del tipo “miCorreo@gmail.com” o “RaMoN@jarroba.com”. Y no permite direcciones de correos vacíos como el de “@.” o el de “@algo.”. Si te fijas en el diagrama de estados, obligamos en cada parte de la dirección del correo a que al menos haya una letra mínimo (por ejemplo, la flecha que pasa entre S0 y S1); luego todas las letras que se quieran (por ejemplo, el

bucle en S1; hasta encontrar la arroba, momento para saltar al estado S2).

Nota: para amenizar la lectura he descrito ejemplos de direcciones de correos electrónicos simples. Existen otras direcciones de correos válidas en los estándares de Internet, como "correo_Numero8&dolar$@inglaterra.co.uk". Es decir, podría tener números, guiones bajos, algunos caracteres más. E incluso podría tener varias extensiones en la parte final del dominio (normalmente el tipo de dominio de la actividad ".co", seguido del tipo de dominio geográfico ".uk"). Si quieres conocer la sintaxis exacta de una dirección de correo electrónico, la tienes descrita en http://en.wikipedia.org/wiki/Email_address, donde además tienes ejemplos de correos válidos e inválidos.

Nota técnica de porqué ayudarte de diagramas de estados con expresiones regulares: Dado que una expresión regular describe cadenas en lenguaje regular, las cuales entran dentro de los lenguajes formales (dicho rápidamente, un lenguaje formal es un texto que sigue unas normas; y en el caso que nos ocupa la norma la marca la expresión regular). Los autómatas finitos reconocen lenguajes regulares. Entonces cada expresión regular tiene asociado un autómata finito (tanto autómatas finitos deterministas o AFD, como autómatas finitos no deterministas o AFND).

Reconocimiento de patrones versus Búsqueda de patrones

Lo que acabas de hacer -desmenuzar la dirección de correo electrónico para encontrar el patrón- se llama "Reconocimiento de patrones" (Pattern recognition).

No hay que confundir "Búsqueda de patrones" con "Reconocimiento de patrones". Defino los dos rápidamente:

- **Reconocimiento de patrones (Pattern recognition):** extrae información de algo para obtener un patrón. El resultado es un patrón. En el ejemplo de la dirección de correo hemos obtenido el patrón de las

direcciones de correos electrónicos a base de estudiarlo mentalmente. Que decir tiene que es una ciencia de computación que intenta llevar a un ordenador a encontrar patrones. Por ejemplo el reconocimiento de caras, donde el ordenador dado una cara de frente de una persona, tiene que saber sacar el patrón de esa cara (Reconocimiento de patrones); para posteriormente reconocerla en diferentes posiciones (Búsqueda de patrones), y sin equivocarse con las caras de otras personas.

- **Búsqueda de patrones (Pattern matching):** busca algo que coincida con un patrón. El resultado son todas las coincidencias con un patrón determinado. En el ejemplo, es encontrar todas las direcciones de correos electrónicos conociendo la estructura básica de un correo electrónico (que tiene arroba en medio y un punto); es decir, sabiendo su patrón.

Necesitamos reconocer un patrón (en el caso de la dirección de correo electrónico es muy sencillo), y luego utilizar este patrón para buscar aquellos textos que coincidan con ese patrón (en el ejemplo queremos encontrar cualquier dirección de correo electrónico, entre las infinitas posibilidades de combinación que pueda haber).

Expresiones regulares

Para que nuestro ordenador sea capaz de reconocer direcciones de correo electrónico, lo tenemos que escribir en el formato que se llama "**expresión regular**" o "Regex" o "RegExp". Una expresión regular simplemente es una secuencia de caracteres. La expresión regular será nuestro patrón, que servirá para buscar algo en un determinado texto (en el caso del ejemplo, queremos encontrar las direcciones de correos electrónicos). En programación es muy útil utilizar expresiones regulares para buscar cadenas o sustituir textos que cumplan con cierta expresión regular (que cumpla con cierto patrón).

Las expresiones regulares siempre se escriben igual sea el lenguaje de programación que sea (por fortuna es un estándar). Al principio y al final siempre hay que poner delimitadores (en algunos lenguajes no es necesario, te lo explico un poco más abajo), que indiquen dónde empieza y dónde acaba la expresión regular; y en medio estará nuestro patrón:

```
Delimitador Patrón Delimitador
```

Sobre las tablas de símbolos que vienen a continuación: te voy a poner las tablas de símbolos antes de mostrarte algún ejemplo. Podrías pensar "ya me tengo que aprender otro montón de símbolos". Te aseguro que no son tantos, muchos son el contrario del anterior, y la verdad es que entendidos unos pocos el resto vienen solos por pura necesidad lógica. Simplemente ojea las siguientes tablas rápidamente para tener una idea del conjunto, detente en los símbolos que tengas curiosidad, y pasa al siguiente punto; la necesidad de utilizar estas tablas viene sola, te terminas aprendiendo los símbolos con el uso. Antes te tengo que decir como interpretar las tablas. Cada tabla consta de cuatro columnas: la primera empezando por la izquierda y sin leyenda podríamos llamarla "Símbolo de expresión regular"; la segunda columna es la descripción de qué es lo que hace el símbolo; la tercera y cuarta van juntas, la tercera es un ejemplo de la expresión regular, y la última es el texto de ejemplo donde he marcado en negrita lo que reconoce la expresión regular. Decir que los ejemplos los he recorrido completos, buscando todas las coincidencias. Ya estás listo, ahora empecemos; aunque comenzaremos por los delimitadores, algo necesario para construir expresiones regulares.

Delimitadores de la expresión regular

Es necesario para indicar dónde empieza y dónde acaba un patrón. Es obligatorio en algunos lenguajes de programación o en algunas funciones (en los lenguajes donde no hay que poner delimitadores, traen delimitadores predefinidos puestos de manera automática). Pero donde es obligatorio se permite cualquier símbolo siempre que coincida el primero con el último, uno muy usado es“/” o “~”. Por ejemplo

```
/patron/
```

Los únicos que no podremos usar son “\” y “ ” (espacio en blanco).

Otra opción, si nos gusta más, es utilizar pares de apertura y cierre como “()”, “{}”, “[]” y “<>”. Por ejemplo es válido:

```
{patron}
```

Puede que nos sea necesario utilizar el mismo símbolo que ya utilizamos de delimitador dentro del patrón. Para ello le ponemos delante la barra “\”, que indica que ese carácter no es especial. Si prevemos que vamos a utilizar mucho un símbolo, lo mejor es usar otro delimitador, que para eso tenemos libertad (por ejemplo, si vas a analizar HTML, donde analizaríamos elementos como “texto en negrita”; sería incómodo utilizar los delimitadores de “/” o de “<>”).

```
<cinco_es\<que_seis>
```

Al final, fuera de los delimitadores, pueden ir algunos modificadores de la expresión regular. Lo veremos en las tablas.

Indicar que los ejemplos de las siguientes tablas varío en el uso de los delimitadores para facilitar la lectura (utilizo "/" y "~"). Como es obvio, en los lenguajes que no necesiten delimitadores no habría que ponerlos.

Caracteres especiales de búsqueda

	Descripción	*Patrón de ejemplo*	*Ejemplo reconocido*
^	Encuentra el elemento siguiente al **principio de línea** **Nota**: así usado no es una negación. No confundir con la negación de conjuntos “[^patron]” que	/^A/	**A**sAdA

confundir con la negación de conjuntos "[^patrón]" ,que requiere los corchetes además del símbolo "^"

$	Encuentra el elemento anterior al **final de línea**	/A$/	AsAd**A**
*	Encuentra el elemento anterior **0 o más veces**. Es lo mismo que poner {0,}	/es*/ **Nota:** el "*" solo afecta a "s", no a "e". Si queremos que afecte a la "e" también, tendríamos que poner "/e*s*/". Lo mismo le pasa al "+" y a "?"	Es **es e**l **ess e** os oss **esss**
+	Encuentra el elemento anterior **1 o más veces**. Es lo mismo que poner {1,}	/es+/	Es **es** el **ess** e os oss **esss**
?	Encuentra el elemento anterior **0 o 1 vez**. Es lo mismo que poner {0,1}	/es?/	Es **es e**l **es**s **e** os oss **es**ss
.	Encuentra **cualquier carácter**, salvo el carácter de nueva línea. Es un comodín que sustituye a un carácter	/s.s/	Es es el **ses** e os **sos** esss
\	El carácter puesto a continuación será **convertido en carácter especial o**, si ya es especial, **deja de ser especial** **Nota**: se puede hacer que deje de ser especial a la misma barra "\". Bastará con poner "\\", donde la	/midominio\.com/	**midominio.com** **Nota**: el punto es un carácter especial que si lo queremos usar tiene que dejar de ser especial

la misma barra "\". Bastará con poner "\\", donde la primera barra es el carácter especial que quita a la segunda barra de ser especial. Y si queremos que haya dos barras seguidas que no sean especiales pondremos "\\\\"; cada barra que queramos que no sea especial tendrá que tener delante una barra que haga que la siguiente barra deje de ser especial

que dejar de ser especial. Si ponemos "/midominio.com/", sin la barra antes del punto, estamos diciendo que también es válido "midominioHcom", pues es el punto es un comodín

Paréntesis

	Descripción	*Patrón de ejemplo*	*Ejemplo reconocido*
[patron]	Encuentra **cualquier carácter de este conjunto**	/[eo]s/	Es **es** el **es**s e **os os**s **es**ss e**os** o**es** uuus
[^patron]	Encuentra **cualquier carácter que no esté en este conjunto**	/[^eo]s/	**Es** es el e**ss** e os o**ss** es**ss** eos oes uu**us**
(patron)	**Encuentra** lo que está entre paréntesis **y lo guarda**. **Nota**: Se guarda cada paréntesis en una variable numerada entre el 1 y el 99; siempre en incremento desde el 1 por cada par de paréntesis en el patrón. La variable 0 corresponde con el texto que ha encontrado el patrón. Por ejemplo: /(A)(B)\2\1/ es equivalente a /(A)(B)BA/. (Más ejemplos abajo)	/(guarda.)(guarda.)/	**guardaAguardaB** 0 = "guardaAguardaB" 1 = "guardaA" 2 = "guardaB"

			Nota: Se reconoce toda la palabra y además la guarda en una variable
(?:patron)	**Encuentra** lo que está entre paréntesis, **pero no lo guarda**.	/(?:guarda.)(guarda.)/	**guardaAguardaB** 0 = “guardaAguardaB” 1 = “guardaB”
{num}	Encuentra el elemento anterior **tantas veces como ponga en el número**.	/s{3}/	Es es el ess e os oss e**sss**
{min,max}	Encuentra el elemento anterior **tantas veces entre un mínimo** (primer número) **y un máximo** (segundo número), ambos incluidos. Si en máximo no se pone nada, significa infinito	/s{2,3}/	Es es el e**ss** e os o**ss** e**sss**

Caracteres

	Descripción	*Patrón de ejemplo*	*Ejemplo r*
\b	Encuentra la **palabra exacta**, siempre ubicado en el límite de la palabra, normalmente un espacio.	~\bPala~~bra\b ~	**Pala**braP

	Nota: podemos encontrar un espacio en blanco en un conjunto poniendo “[\b]”, aunque también el propio espacio en blanco “[ ]”.	**Nota**: está al principio o final de “Palabra”	
\B	Encuentra la **palabra exacta**, no tiene que estar ubicado en el límite de la palabra (Lo contrario a “\b”)	~Pala\B~~\Bbra~ **Nota**: está en medio de “Palabra”	**Pala**braP
\d	Encuentra un **dígito** del 0 al 9. Es lo mismo que poner [0-9]	~\d~	**5** del **12** d
\D	Encuentra un carácter que **no sea un dígito** del 0 al 9 (Lo contrario a “\d”). Es lo mismo que poner [^0-9]	~\D~	5 **del** 12 **d**
\f	Encuentra un carácter de **nueva página** (form feed)	~\f~	Nueva Pa
\n	Encuentra un carácter de **nueva línea**	~\n~	Nueva Lir
\t	Encuentra un carácter de **tabulación**	~\t~	Tabulacio
\v	Encuentra un carácter de **tabulación vertical** **Nota:** Utilizado para el movimiento vertical de las impresoras	~\v~	Tabulacio
\r	Encuentra un carácter de **retorno de carro**	~\r~	Retorno c

	Nota: Al pulsar la tecla “Enter”, antes de nueva línea		
\s	Encuentra un carácter de **espacio** (incluye espacio, tabular, nueva página, nueva línea, entre otros)	~\s~	Te **No** blanco
\S	Encuentra un carácter que **no sea espacio** (Lo contrario a “\s”)	~\S~	**Te** **No** espacios
\w	Encuentra un carácter **alfanumérico** (incluye letras, números y barra baja). Es lo mismo que poner [A-Za-z0-9_].	~\w~	**Ramon_1**
\W	Encuentra un carácter que **no sea alfanumérico** (Lo contrario a “\w”). Es lo mismo que poner [^A-Za-z0-9_].	~\W~	Ramon_1
\n	Es la **variable guardada** por los paréntesis “(patron)”. “n” es un número positivo entre 1 y 99.	~Numero capicua: (.).\1~	**Nu** **No** sobre est
\0	Encuentra un carácter **NUL**. El carácter null se representa como “\x00”	~Es un nul: \0~	**Es un nul**

\x{hhhh}\xhhhh	Encuentra un carácter **Unicode**. “hhhh” son cuatro dígitos hexadecimales. Algunos lenguajes de programación aceptan tanto “\x{hhhh}” o “\xhhhh”, o bien una sola **Nota**: Tienes todos los caracteres Unicode en (notar que en esta web aparecen escritos como “U+hhhh”, lo tenemos que transformar a “\x{hhhh}”): http://en.wikipedia.org/wiki/List_of_Unicode_characters	~Unicode del caracter Arroba: \x{0040}~	**Unicode**
\x{hh}\xhh	Encuentra un carácter **ASCII**. “hh” son dos dígitos hexadecimales. Algunos lenguajes de programación aceptan tanto “\x{hh}” o “\xhh”, o bien una sola **Nota**: Tienes todos los caracteres ASCII en (notar que en esta web aparecen escritos como dos dígitos hexadecimales, lo tenemos que transformar a “\x{hh}”): http://en.wikipedia.org/wiki/ASCII	~ASCII del caracer Arroba: \x{40}~	**ASCII del**

Operadores lógicos

	Descripción	*Patrón de ejemplo*	*Ejemplo reconocido*
patron1|patron2	patron1 **o** patron2	~ojo|a.a~	**ojo asa** oro **aca**
patron1 (?=patron2)	Encuentra patron1 **solo si le sigue** patron2. patron2 no formará parte del resultado encontrado	~ojo (?=cuidado)~	**ojo** cuidado, ojo chispeante
patron1 (?!patron2)	Encuentra patron1 **solo si no le sigue** patron2. patron2 no formará parte del resultado encontrado	~ojo (?!cuidado)~	ojo cuidado, **ojo** chispeante

Modificadores

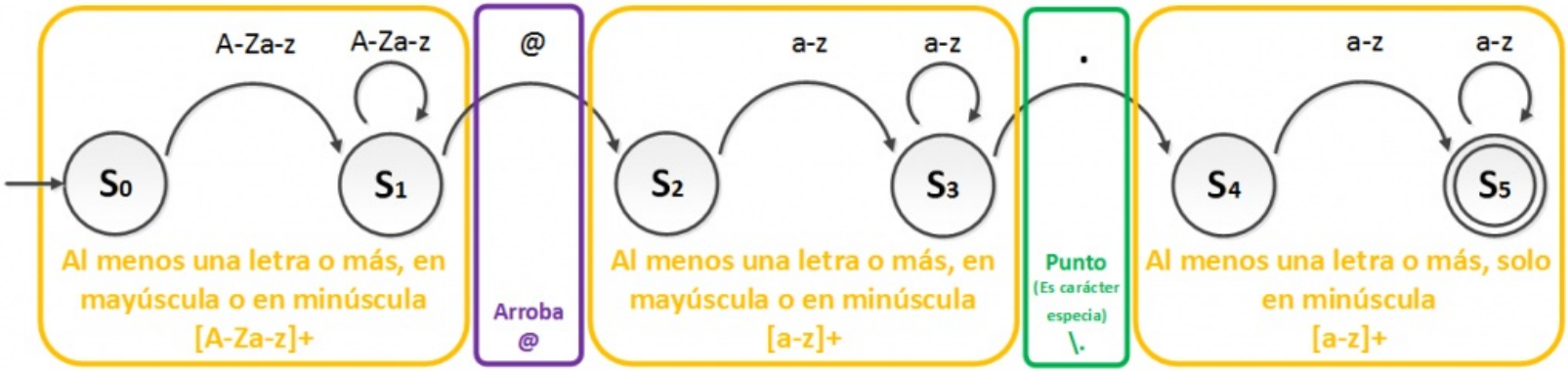
Estos modificadores de la expresión regular. Van siempre al final y fuera del delimiatador. Algunos lenguajes de programación tienen estos modificadores en forma de función, que afecta al reconocimiento del texto por la expresión regular.

	Descripción	*Patrón de ejemplo*	*Ejemplo reconocido*
g	Búsqueda global de **todas las coincidencias**.En PHP no existe, hay que utilizar el método “preg_match_all()”	/es/g	Es aqui, **es** el lugar donde **es** **Nota**: “Es” no se reconoce, por la letra mayúscula “E”
i	**No distinguir entre mayúsculas o minúsculas**	/es/i	**Es** aqui, **es** el lugar donde **es**
m	Encuentra los saltos de línea “\n”, para **separar cada línea de un texto**. Útil para utilizar con “^” o “$”	/^A/m	**A**rAdA\n**A**rAdA

Sobre las tablas: Para crear estas tablas me he basado en los trabajos de http://php.net/manual/es/book.pcre.php y de https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp. Donde puedes encontrar más información al respecto, aunque he intentado hacer una recopilación con todo lo necesario. Todos los ejemplos han sido desarrollados y probados por mí.

Crear una expresión regular

Ya tenemos toda la información necesaria, así que volvamos al último diagrama de estados. Vamos a crear nuestra expresión regular en base al diagrama de estados. Desgranémosla:



Apreciamos como el estado S0 y S1 nos indican que son letras mayúsculas y minúsculas, y que tienen que ser mínimo una letra o más letras. Revisando las tablas vemos que es lógico que sea "[A-Za-z]+" (El primer diagrama de estados, el que dije que permitía estados no deseados, indica ciclos entre cero y muchas letras, si miramos las tablas veremos que el asterisco nos apaña y valdría con poner "[A-Za-z]*"). Luego la arroba nos basta con poner "@". Los estados S2 y S3 es lo mismo que antes, salvo porque no se permiten mayúsculas, por lo que "[a-z]+". Luego llega un punto, pero cuidado con éste, si miramos las tablas aparece como carácter especial. Tenemos que hacer que deje de ser especial, es tan fácil como ponerle la barra "\" para que deje de ser especial. Y ya los estados S3 y S4 igual que antes "[a-z]+".

Por tanto, la expresión regular que buscamos sería la siguiente (he utilizado la barra "/" como delimitador):

```
/[A-Za-z]+@[a-z]+\.[a-z]+/
```

¿Sabías que los analizadores sintácticos de los compiladores (de Java, Python, etc) usan expresiones regulares para saber si el código está bien escrito? Ahora ya sabes quién tiene la culpa de cuando falta un "punto y coma" al final de una línea de código, que aparezca marcado en rojo en el IDE y entonces no compila el programa. Se debe a que la expresión regular que analiza el código tiene el "punto y coma" como obligatorio. Por tanto, sin "punto y coma" el código no pasa la validación. Sin embargo, otros lenguajes como JavaScript, sabrás que el "punto y coma" no es obligatorio; es razonable pensar que estarán usando el símbolo "?" detrás del "punto y coma" para que pueda aparecer cero o una vez, para que el "punto y

coma" sea opcional. No me extiendo más, tienes más información sobre compiladores en http://en.wikipedia.org/wiki/Compiler

Veamos unos ejemplos de uso de expresiones regulares tras responder a unas preguntas ¿Qué se puede hacer con las expresiones regulares? ¿Qué utilidad práctica tienen las expresiones regulares? ¿Para qué sirven las expresiones regulares? La respuesta la basaré en diferenciar el uso más típico de las expresiones regulares en programación con código funcional en Java, PHP y JavasCript:

- **Buscar** un patrón y decir si se ha encontrado o no
- **Remplazar** lo que ha encontrado un patrón por otra cosa
- **Separar** partes y guardarlas en variables
- **Contar** todas las coincidencias que encuentra un patrón
- **Posición** de cada coincidencia que haya encontrado el patrón

Ejemplos de Búsqueda o Coincidencia

Queremos encontrar las direcciones de correos del listado anteriormente citado. La salida por consola en todos los casos va a ser:

Correo reconocido: miCorreo@gmail.com
Correo reconocido: correoFalso@yahoo.es
Correo reconocido: RaMoN@jarroba.com

El código dependerá del lenguaje. Unos ejemplos rápidos de algunos lenguajes son:

PHP

En PHP utilizaremos la función **preg_match()**, a la que le pasaremos el patrón y un texto del listado. Nos dirá si texto coincide con el patrón o no.

```php
<?php

 $listaTextos = array(
  '¡Hola Mundo!',
  'miCorreo@gmail.com',
  'La teoría de "Pattern Machine" dice…',
  'correoFalso@yahoo.es',
  'En un lugar de la Mancha, cuyo nombre no quiero acordarme…',
  '+34 91 123 456 789',
  'estoNOesUnCorreoNoTieneArroba.com ',
  'RaMoN@jarroba.com',
  'Calle Alcalá 12345 Madrid, Madrid'
 );

 $patron = '/[A-Za-z]+@[a-z]+\.[a-z]+/';

 foreach ($listaTextos as $texto) {
  $esCoincidente = preg_match($patron, $texto);

  if ($esCoincidente) {
   echo '<br/>Correo reconocido: ' . $texto;
  }
 }
```

JavaScript

Lo mismo pero con JavaScript. Para hacer la validación utilizaremos la función **test()**, cuyo objeto es el propio patrón que pondremos sin comillas (aquí la expresión regular actúa de objeto, no de String). A la función test() le pasaremos cada texto del listado. Nos devolverá "true" si coincide con la expresión regular, o "false" en caso contrario.

```
var listaTextos = [
 '¡Hola Mundo!',
 'miCorreo@gmail.com',
 'La teoría de "Pattern Machine" dice…',
 'correoFalso@yahoo.es',
 'En un lugar de la Mancha, cuyo nombre no quiero acordarme…',
 '+34 91 123 456 789',
 'estoNOesUnCorreoNoTieneArroba.com ',
 'RaMoN@jarroba.com',
 'Calle Alcalá 12345 Madrid, Madrid'
];

var patron = /[A-Za-z]+@[a-z]+\.[a-z]+/;

for (i = 0; i < listaTextos.length; i++) {
 texto = listaTextos[i];
 var esCoincidente = patron.test(texto);

 if (esCoincidente) {
  console.log('Correo reconocido: '+ texto);
 }
}
```

Java

En Java es muy sencillo utilizar expresiones regulares, va todo por objetos. Primero tenemos que tener el patrón en un objeto "Pattern", donde introduciremos la expresión regular sin delimitadores (incido en "sin delimitadores") mediante el método compile(). Luego, solo tenemos que emparejar el texto con el patrón, nada más fácil que llamar al método matcher(), pasándole el texto que queramos comprobar; y nos devolverá un objeto de tipo "Matcher". Para validar si coincide o no con la expresión regular utilizamos **find()** del objeto "Matcher", donde se nos devolverá un "true" si coincide o un "false" si no (Una cosa muy cómoda es que si hay varias coincidencias con el patrón dentro del mismo texto, cada vez que llamemos a find() nos devolverá la siguiente coincidencia; un "while" nos descubriría todas).

Nota: si te fijas en la expresión regular, verás que el punto tiene doble barra "\\" delante. La primera barra es para Java (la barra "\" en los Stings de Java también convierte o quita de que sean caracteres especiales),

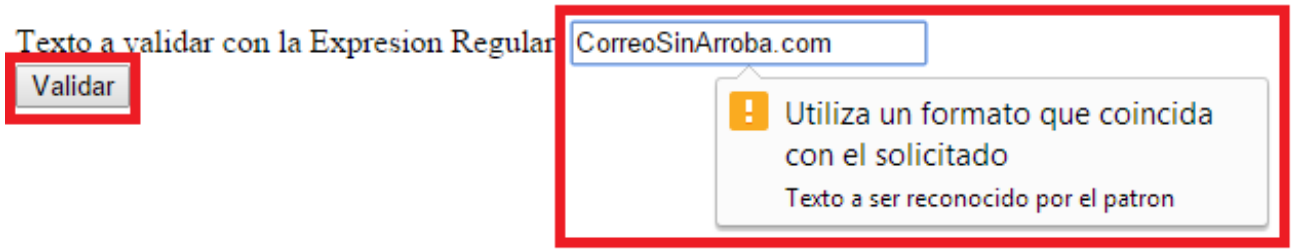
no tiene nada que ver con la expresión regular; le dice a Java que la barra “\” de después es parte del String y que no la tiene que evaluar.

```
package jarroba.com;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class expresionesRegulares {

 public static void main(String[] args) {

  String[] listaTextos = {
   "¡Hola Mundo!",
   "miCorreo@gmail.com",
   "La teoría de 'Pattern Machine' dice…",
   "correoFalso@yahoo.es",
   "En un lugar de la Mancha, cuyo nombre no quiero acordarme…",
   "+34 91 123 456 789",
   "estoNOesUnCorreoNoTieneArroba.com",
   "RaMoN@jarroba.com",
   "Calle Alcalá 12345 Madrid, Madrid"
  };

  String regex = "[A-Za-z]+@[a-z]+\\.[a-z]+";
  Pattern patron = Pattern.compile(regex);

  for (String texto : listaTextos) {
   Matcher emparejador = patron.matcher(texto);
   boolean esCoincidente = emparejador.find();

   if (esCoincidente) {
    System.out.println("Correo reconocido: " + texto);
   }
  }

 }
}
```

HTML

```
<form method="get">
 Texto a validar con la Expresion Regular: <input type="text" name="texto" pattern="[A-Za-z]
+@[a-z]+\.[a-z]+" title="Texto a ser reconocido por el patron"/>
 </br>
 <input type='submit' value="Validar"/>
</form>
```

Si introducimos un dato que sea una dirección de correo electrónico, esta pasará la validación (se envía el formulario al hacer el "submit"). Si no pasa la validación mostrará un cuadro avisando de ello. Veríamos en el navegador la siguiente imagen (los estilos varían de navegador a otro; así se ve si se utiliza el navegador Chrome).



Ejemplos de Sustitución o Remplazo

Vamos a hacer un ejemplo con paréntesis de guardado "(patron)".

Sabemos que cada par de paréntesis "()" guarda el contenido coincidente entre el paréntesis de apertura y cierre una variable empezando desde el 1. Si tenemos 4 pares de paréntesis, tendremos información en las variables 1, 2, 3 y 4, donde 1 es el par de paréntesis de más a la izquierda y 4 el de más a la derecha de la expresión regular.

Esto es muy útil en muchos casos. Por ejemplo, si queremos reconocer un código de cualquiera HTML (o XML), como:

```
<b>Texto en negrita</b>
```

Podríamos diseñar una expresión regular como la siguiente (cambio el delimitador "/" por "~", ya que el cierre del tag "" requiere el símbolo "/", por simplificar la lectura):

```
~<.+>.*</.+>~
```

Esta nos indica que buscaremos en los tags de apertura y cierre cualquier carácter 1 o más veces (pues los tags siempre han de tener algo, mínimo 1), y dentro del elemento buscaremos cualquier carácter 0 o más veces (podría ser vacío, por eso 0 o más). El problema está que las búsquedas requieren procesamiento. Podemos ahorrar cálculos si guardamos en variables. Sabemos que lo que nos encuentre el tag de apertura ha de ser igual al de cierre, por lo que podemos hacer lo siguiente:

```
~<(.+)>.*</\1>~
```

De este modo buscamos la coincidencia y el paréntesis nos lo guarda en la variable "1". Para llamar la variable dentro de la expresión regular, lo tenemos que tratar al "1" como símbolo especial y para ello se le pone "\" delante, por lo llamamos a la variable de la forma "\1". Sin salirnos de la expresión regular sirve para cualquier lenguaje.
Ahora veamos otro uso muy interesante que es el de sustitución. Tenemos la siguiente fecha del tipo "d de F de Y" (donde "d" representa el día con dos dígitos, "F" el mes escrito, e "Y" el año con cuatro dígitos):

```
31 de Enero de 2015
```

Queremos convertirlo al formato del tipo "d-F-Y" (es decir, que termine siendo "31-Enero-2015").
Haríamos la siguiente expresión regular para obtener cada parte que nos interesa y guardar cada parte en una

variable:

```
/(\d+) de (\w+) de (\d+)/
```

El primer paréntesis desde la apertura "(" hasta su cierre ")" guardará el día "31" en la variable $1, el segundo el mes "Enero" en la variable $2, y el tercero el año "2015" en la variable $3. Utilizamos una función de remplazo que nos permita usar expresiones regulares y pueda sustituir la variables.
Vamos a poner algunos ejemplos completos, cuya salida de consola de todos es:

Resultado despues de la sustitucion: 30-Enero-2015

PHP

Es tan fácil como llamar a la función **preg_replace()**, donde le pasaremos la expresión regular, el texto donde tenemos que realizar la búsqueda, y el cómo queremos que nos sustituya a las variables; devolverá un String con las variables remplazadas tal y como queríamos.

```
$texto = '30 de Enero de 2015';
$patron = '/(\d+) de (\w+) de (\d+)/';

$sustitucion = '$1-$2-$3';
$resultado = preg_replace($patron, $sustitucion, $texto);

echo '<br/>Resultado despues de la sustitucion: "'.$resultado;
```

JavaScript

Simplemente llamando a la función **replace()** del texto, donde le pasaremos el patrón y el cómo queremos la sustitución de las variables; nos devolverá el resultado con las variables sustituidas.

```
var patron = /(\d+) de (\w+) de (\d+)/;
var texto = '30 de Enero de 2015';

var sustitucion = '$1-$2-$3';
var resultado = texto.replace(patron, sustitucion);

console.log('Resultado despues de la sustitucion: '+ resultado);
```

Java

La búsqueda la explicamos en el anterior ejemplo de Java (recuerdo que la expresión regular ha de ir sin delimitadores). Para que nos busque el objeto “Matcher” tenemos que llamar a find(). Una vez encontrada la coincidencia, para realizar el remplazo podemos llamar al método **replaceAll()**, donde le pasaremos un String con el cómo queremos que remplace a las variables $1, $2 y $3; devolverá un String con las variables remplazadas.

```java
package jarroba.com;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class expresionesRegulares {

 public static void main(String[] args) {
  String regex = "(\\d+) de (\\w+) de (\\d+)";
  Pattern patron = Pattern.compile(regex);

  String texto = "30 de Enero de 2015";
  Matcher emparejador = patron.matcher(texto);

  emparejador.find();

  String sustitucion = "$1-$2-$3";
  String resultado = emparejador.replaceAll(sustitucion);

  System.out.println("Resultado despues de la sustitucion: " + resultado);
 }

}
```

Ejemplos de Separación o extracción

Los anteriores ejemplos están muy bien. Pero qué pasa si lo que quiero es, de un texto coincidente con la expresión regular, quedarme con las partes que me interesen en variables para poder utilizarlas más adelante en el código de mi programa.

Por ejemplo, quiero quedarme con el día por un lado, el mes por otro, y el año por otro en variables; para poder utilizar estos valores en mi programa (creo que no hace falta que te diga que imagines los cientos de usos que tiene esto: saber si es día par o impar, sumar uno al día, comprobar si el mes cae en verano, etc; y eso solo

con fechas, ¡Imagina con otros patrones y textos! Las posibilidades son infinitas). Supongamos que tenemos la misma fecha de antes:

```
31 de Enero de 2015
```

Con la misma expresión regular quiero extraer cada parte a variables de mi lenguaje de programación para poder utilizar dichos valores:

```
/(\d+) de (\w+) de (\d+)/
```

Vamos a poner algunos ejemplos completos, cuya salida por consola es la siguiente:

Contenido en la variable $1: '30', en la variable $2: 'Enero', en la variable $3: '2015'

PHP

Sencillamente usando **preg_match_all()**, donde le pasaremos la expresión regular, el texto, y un Array vacío (no hace falta inicializarlo) que nos llenará con las variables separadas. Luego podremos recorrer el Array, y así obtener los valores.

```php
$texto = '30 de Enero de 2015';
$patron = '/(\d+) de (\w+) de (\d+)/';

preg_match_all($patron, $texto, $coincidencias);

$el1 = $coincidencias[1][0];
$el2 = $coincidencias[2][0];
$el3 = $coincidencias[3][0];

echo 'Contenido en la variable $1: \''.$el1.'\', en la variable $2: \''.$el2.'\', en la vari
able $3: \''.$el3.'\'';
```

El Array bidimensional devuelto “$coincidencias” con el contenido (la posición 0 es el texto reconocido por la expresión regular; el resto de posiciones son las variables cada una y en orden):

```
array(4) {
 [0]=> array(1) { [0]=> string(19) "30 de Enero de 2015" }
 [1]=> array(1) { [0]=> string(2) "30" }
 [2]=> array(1) { [0]=> string(5) "Enero" }
 [3]=> array(1) { [0]=> string(4) "2015" }
}
```

JavaScript

Utilizando la función **match()**, cuyo objeto es el texto del que queremos extraer la información y pasándole la expresión regular como argumento. Obtendremos un array del cual obtener los valores.

```
var patron = /(\d+) de (\w+) de (\d+)/;
var texto = '30 de Enero de 2015';

var coincidencias = texto.match(patron);

var el1 = coincidencias[1];
var el2 = coincidencias[2];
var el3 = coincidencias[3];

console.log('Contenido en la variable $1: \''+el1+'\', en la variable $2: \''+el2+'\', en la
variable $3: \''+el3+'\'');
```

El array “coincidencias” que nos devuelve nos interesa de la posición 1 a la 3, siendo la posición 0 el texto reconocido por la expresión regular.

```
0: "30 de Enero de 2015"
1: "30"
2: "Enero"
3: "2015"
```

Java

Como hemos hecho en los otros ejemplos en Java, obtendremos el objeto Matcher del objeto Pattern. Después de encontrar las coincidencias con find() del objeto Matcher. Y aquí simplemente llamaremos al método **group()** del objeto Matcher, donde le pasaremos un número entero que corresponderá con la posición de la variable que queremos obtener (a diferencia de PHP o JavaScript, aquí no nos devuelve un Array, sino que directamente el objeto Matcher nos devuelve las variables separadas al llamar a group() a la posición que queramos; si llamamos a la posición “0” con group(0), nos devolverá el texto reconocido por la expresión regular).

```
package jarroba.com;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class expresionesRegulares {

 public static void main(String[] args) {
  String regex = "(\\d+) de (\\w+) de (\\d+)";
  Pattern patron = Pattern.compile(regex);

  String texto = "30 de Enero de 2015";
  Matcher emparejador = patron.matcher(texto);

  emparejador.find();

  String el1 = emparejador.group(1);
  String el2 = emparejador.group(2);
  String el3 = emparejador.group(3);

 System.out.println("Contenido en la variable $1: \'" + el1 + "\', en la variable $2: \'" +
el2 + "\', en la variable $3: \'" + el3 + "\'");
 }

}
```

Ejemplos de Contar coincidencias y la posición donde fue cada una encontrada

Aquí voy a aprovechas a hacer dos ejemplos muy útiles -ya que son bastante sencillos. Contaré el número de veces que aparece la coincidencia y mostraré la posición donde fue encontrada.

Por tanto, haré dos cosas con el texto:

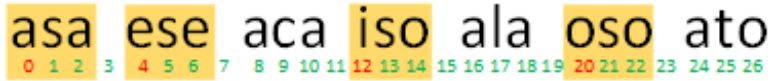
```
asa ese aca iso ala oso ato
```

Contaré todas las coincidencias con la siguiente expresión regular:

```
/.s./
```

Y diré en qué posición se han encontrado.

Veremos que hay 4 posibilidades, y donde está iniciando cada palabra.



Por lo que la salida de consola para los siguientes lenguajes de programación va a ser:

El numero de apariciones del patron .s. es 4
Palabra "asa" encontrada en la posicion 0
Palabra "ese" encontrada en la posicion 4
Palabra "iso" encontrada en la posicion 12
Palabra "oso" encontrada en la posicion 20

PHP

Una de las particularidades de la función **preg_match_all()** es que devuelve el número de veces que ha coincidido el texto con la expresión regular. Tenemos las veces que aparece, ahora nos falta obtener la posición. Para hacerlo, es tan simple como añadir el flag "**PREG_OFFSET_CAPTURE**" a la función preg_match_all(), para

pedir que nos devuelva además la posición dónde localiza a cada palabra.

```
$texto = 'asa ese aca iso ala oso ato';
$patron = ' /.s./ ';
$numApariciones = preg_match_all($patron, $texto, $coincidencias, PREG_OFFSET_CAPTURE);
echo '<br/>El número de apariciones del patron .s. es '.$numApariciones;

foreach ($coincidencias[0] as $palabraPosicion) {
 echo '<br/>Palabra "'.$palabraPosicion[0].'" encontrada en la posición '.$palabraPosicion[1
];
}
```

El array “concidencias” va a ser un array multidimensional, donde cada encuentro lo va a guardar en otro array de dos posiciones. La posición 0 del array será la palabra encontrada, y la posición 1 del array será la posición de dicha palabra en la frase. El resultante es:

```
array(1) {
 [0]=> array(4) {
  [0]=> array(2) {
   [0]=> string(3) "asa"
   [1]=> int(0)
  }
  [1]=> array(2) {
   [0]=> string(3) "ese"
   [1]=> int(4)
  }
  [2]=> array(2) {
   [0]=> string(3) "iso"
   [1]=> int(12)
  }
  [3]=> array(2) {
   [0]=> string(3) "oso"
   [1]=> int(20)
  }
 }
}
```

JavaScript

En JavaScript tenemos que indicar que queremos encontrar todas las coincidencias con la expresión regular con el **modificador "g" al final de la expresión regular**, para que busque todas las coincidencias de toda la frase (si no lo ponemos solo encuentra "asa"; es decir, la primera que encuentra y para). Luego, para contar las coincidencias, usaremos la función **match()** que nos devuelve un array con todos los encuentros, por lo que nos bastará con contar el array para saber el total. Para conseguir la posición de cada coincidencia utilizaremos el método **exec()**, que nos irá encontrando de una en una todas las coincidencias cada vez que lo llamemos dentro de un bucle; luego la posición es el atributo "index" que devuelve esta función.

```
var texto = 'asa ese aca iso ala oso ato';
var patron = /.s./g;

var coincidencias = texto.match(patron);
var numApariciones = coincidencias.length;
console.log("El numero de apariciones del patron .s. es " + numApariciones);

while ((coincidencia = patron.exec(texto)) != null) {
 console.log('Palabra "'+coincidencia+'" encontrada en la posicion '+coincidencia.index);
}
```

El array “coincidencias” tiene lo siguiente:

```
0: "asa"
1: "ese"
2: "iso"
3: "oso"
length: 4
```

Y un ejemplo de lo que contiene el array “coincidencia”

```
0: "asa"
index: 0
input: "asa ese aca iso ala oso ato"
```

Java

Similar a los otros ejemplos en Java de más arriba, simplemente utilizamos el objeto **Matcher** para ir encontrando cada coincidencia con el método **find()** dentro de un bucle while. Luego, simplemente el objeto Matcher ubicará internamente su cursor en cada encuentro, por lo que si llamamos a la función **group()** nos devolverá la palabra en ese momento, y para saber en qué posición se encuentra llamaremos al método **start()**

(y si queremos saber dónde acaba la palabra podemos llamar a end()). Para contar las apariciones no tenemos más opción que ir contándolas una a una. Para que coincida la salida de consola con la del ejemplo, he utilizado StringBuilder, que lo único que hace es concatenar los Strings con las funciones append() (es igual que hacer "string unido a"+"otro string"; pero con más opciones, como la posibilidad de insertar al principio con insert()).

```
package jarroba.com;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class expresionesRegulares {

 public static void main(String[] args) {
  String regex = ".s.";
  Pattern patron = Pattern.compile(regex);

  String texto = "asa ese aca iso ala oso ato";
  Matcher emparejador = patron.matcher(texto);

  StringBuilder devolver = new StringBuilder();

  int numApariciones = 0;

  while (emparejador.find()) {
   devolver.append("\nPalabra \"" + emparejador.group() + "\" encontrada en la posicion " +
emparejador.start());
   numApariciones++;
  }

  devolver.insert(0, "El numero de apariciones del patron .s. es " + numApariciones);

  System.out.append(devolver.toString());
 }

}
```

Varios ejemplos de expresiones regulares

Para solventar dudas en esta sección voy a ir poniendo varios ejemplos de expresiones regulares. Puedes utilizarlos como ejercicios para coger un poco de soltura. Casi todo se basa en pensar un poco, hacer el diagrama de estados (si tienes soltura los harás de cabeza), y mirar las tablas de símbolos de las expresiones regulares antes citadas.

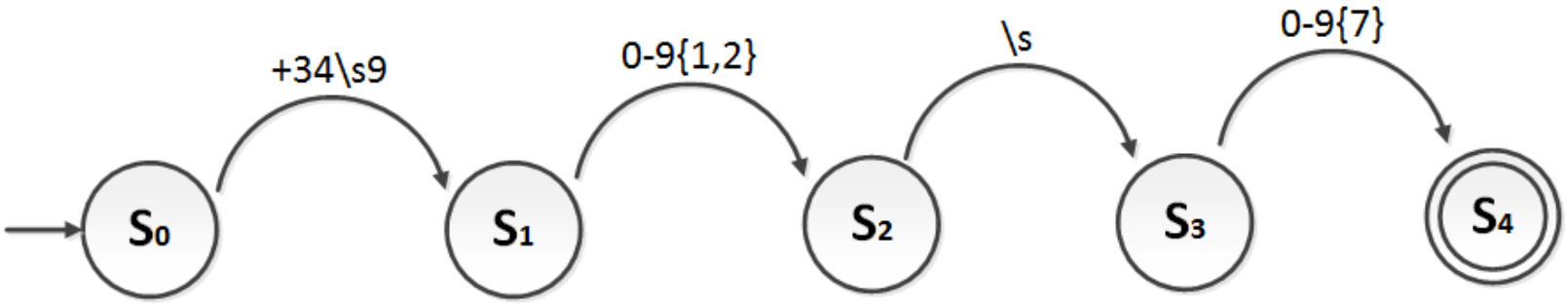
Expresión regular de teléfono fijo en España

para otros países cambian las normas, pero son parecidos, ya tienes ejercicio 😉

- El prefijo del país España es siempre “+34”
- Luego va el prefijo de la comunidad autónoma, son entre 2 y 3 cifras que comienzan siempre por “9”
- Y luego va el número, que son siempre 7 cifras numéricas
- Cada uno de los grupos anteriores está separado por un espacio en blanco

Un ejemplo de teléfono válido en España es: +34 91 1234567

Su diagrama de estados podría ser (utilizando la simbología de expresiones regulares se puede simplificar lo suficiente como para entender los pasos de un vistazo):



Indicar que a este diagrama de estados le he añadido un nuevo símbolo entre llaves “{}”. Indica lo mismo que en las expresiones regulares, el símbolo “{1,2}” indica que puede haber desde 1 hasta 2 caracteres (en este caso caracteres del 0 al 9), y “{7}” indica que tienen que ser justos 7 caracteres (en este caso del 0 al 9).

También recuerdo que “\s” simplemente significa que se espera un espacio en blanco “ ” (“\s” está también en la tabla de expresiones regulares). En la expresión regular da igual poner “\s” o “ ”.
La expresión regular resultante:

```
/\+34\s9[0-9]{1,2}\s[0-9]{7}/
```

O lo que es lo mismo:

```
/\+34 9[0-9]{1,2} [0-9]{7}/
```

Probar online

Puedes practicar y probar online las expresiones regulares desde:

- http://www.regexr.com/
- https://regex101.com/

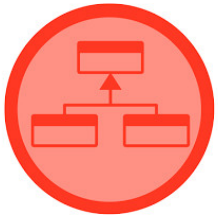
Bibliografía

- Experiencia propia (si tienes dudas pregunta)
- http://en.wikipedia.org/wiki/Pattern_matching

- http://en.wikipedia.org/wiki/Pattern_recognition
- http://en.wikipedia.org/wiki/Regular_expression
- http://php.net/manual/es/reference.pcre.pattern.syntax.php
- http://php.net/manual/es/book.pcre.php
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp
- http://docs.oracle.com/javase/tutorial/essential/regex/index.html

MEAN (Mongo-Express-Angular-Node)...

Aprender a programar conociendo lo que es...



Paradigmas de Programación

Patrón Singleton en Java, con ejemplos

Cliente-Servidor: Petición del Cliente

Multitarea e Hilos en Java con ejemplos...

ArrayList en Java, con ejemplos

Autentificarme en una web y hacer Scraping

74 comentarios en “Búsqueda de patrones – Expresiones regulares”

juce montenegro dice:

13/03/2017 a las 15:35

Excelente artículo. Realmente es admirable la pasión que pones al desarrollar la explicación. Si duda, mas allá de tus logros, esto denota vocación. Un saludo y gracias por compartir.

Responder

mark dice:

10/03/2017 a las 19:46

Estupendo articulo

Responder

Dani dice:

25/02/2017 a las 18:24

Hola Ramón.
Muchas gracias por tu explicación! 😀 Me surge una duda ante la siguiente situación:
Tengo que buscar dos palabras en un texto.
Caso1:
El texto es:
“texto de prueba Coche es muy bonito mas texto”
Tengo que buscar: “coche” y “bonito”
y las palabras que hay en medio.

En PHP he programado algo así:

$stringFound = “texto de prueba Coche es muy bonito mas texto”;
$patt = “/(coche).*(bonito)/i”;
$match1 = preg_match($patt, $stringFound, $coincidencias, PREG_OFFSET_CAPTURE);

El array $coincidencias tiene lo que necesito(OK):
$coincidencias array[3]
[0]
[0] string “Coche es muy bonito”
[1] integer 16
[2]
[0] string “bonito”
[1] integer 29
[1] array[2]
[0] string “Coche”
[1] integer 16

Mi expresión regular (coche).*(bonito) solo sirve si “coche” precede a “bonito”.
En el siguiente texto no lo encontraria:
“texto de prueba El bonito bla bla coche mas texto”
Por ello modifiqué mi expresión regular para que contemple los dos casos:
$patt = “/(bonito.*coche)|(coche).*(bonito)/i”;
Sin embargo el array tiene un resultado no esperado:
$coincidencias2
[0]
[0] string “bonito bla bla coche”
[1] integer 19
[1]
[0] string “”
[1] integer -1
[2]
[0] string “”
[1] integer -1

[3]
[0] string “bonito”
[1] integer 19
[4]
[0] string “coche”
[1] integer 34

No entiendo por qué me devuelve 2 posiciones vacías “” con “-1” ¿Cómo podría solventar este problema? Cómo sería la expresión regular. No quiero usar dos patrones y llamar dos veces a preg_match(...).

Un Saludo y gracias de antemano!

Responder

Iván dice:

16/01/2017 a las 07:57

Lo he entendido todo a la perfeccion, quisiera saber si me podeis ayudar a sacar este texto:

>
>
> Class Central Course Rank
>
>
>
> #3 in <a href="/subjects" onmousedown="ga('send','event', 'Inbound Click – Course Ranking Subject Click', 'Survey of Music Technology',

'Subjects')">Subjects

> Art & Design

> Digital Media & Video Games

extraer texto

Como es

```
<h2>texto</h2>

<b>palabra<br><br><b>casi</b><br>otra parte<br><br><b>modulo3</b>
<br>modulo<br><br><b>otra pearte</b><br>hgjgf<br><br><b>parte43</b>
<br>parte4<br><br><b>parte5</b><br>Parte6<br><br>
```

Quisiera sacar a partir de extraer texto, con todas las etiquetas y tal, y necesito una expresion para sacar todo esto, y no encuentro la manera, puedo conseguir que saque parte, pero no todo

Responder

Ramón [Admin Jarroba] dice:

11/02/2017 a las 17:36

Puedes utilizar los paréntesis en el texto que necesites extraer dentro de la expresión regular. Por ejemplo, podrías poner algo así:

```
<br>(.*)<br>
```

Responder

Luis Garzon dice:

27/12/2016 a las 23:34

Antes que nada,gracias por tomar el tiempo de leer :D.
Bien ya pasando a la pregunta mira es que tengo una base de datos digamoslo asi de noticias en donde se

guarda un campo que es descripcion es decir el contenido de la Noticia.
Y en este los que publiquen pueden meter comandos es decir si quiero poner un link seria:
[link=aquivaellink.com]aqui el texto que tendra el link ingresado[/link]
Algo asi seria, pero he visto que hay algo con retroreferencias usando preg_replace() como haria para por ejemplo cambiar la parte de [link=link] por la etiqueta normal de link <a href=""> y que dentro de las comillas quede el link.

Creo que me diras la expresion regular para el link como tal, pero no es la unica etiqueta que se puede ingresar, tambien se puede digamos una de imagen:

[img link=http://enlacedelaimg.com/foto1.jpg]

Y este deberia reemplazarse por <img src="http://enlacedelaimg.com/foto1.jpg">
No se si me explique bien, pero estare pendiente a cualquier respues, Muchisimas gracias.
Luis Garzon

Responder

Ramón [Admin Jarroba] dice:

28/12/2016 a las 20:19

Hola Luis. Para lo que necesitas utilizaría dos grupos de captura, uno para detectar el “img” si hay, y otro para conseguir la url después de “link=”, algo así:

\[(.)link=(.*)\]*

Podrías hacer algo así (para lo que necesitas no podemos hacer un remplazo directo, sino por extracción y contrucción sobre código):

$texto = ‘[img link=http://enlacedelaimg.com/foto1.jpg]’;

$patron = '/\[(.)link=(.*)\]/';*

preg_match_all($patron, $texto, $coincidencias);

$prefijo = $coincidencias[1][0]; //El "img" si hay, sino será String vacío ""

$url = $coincidencias[2][0]; //La url

Luego ya puedes construir lo que necesites conociendo el prefijo, de una forma o de otra.

Responder

dryant dice:

23/12/2016 a las 14:16

Guau!!!!
Increíble!!! La mejor explicación que he encontrado de expresiones regulares! Es una pasada!!! Mas que un artículo parece una clase magistral!

Enhorabuena y gracias! Y Feliz Navidad!

Responder

Ramón [Admin Jarroba] dice:

27/12/2016 a las 20:49

Gracias Dryant, me alegro que te guste 🙂

¡Feliz Navidad y prospero año nuevo!

Responder

www.trucos-para.com dice:

06/12/2016 a las 19:40

Gracias por la información, un salud y suerte con todo.

Responder

Lore dice:

28/11/2016 a las 02:07

Que buena página tienes la verdad, gracias por tu aporte!! oye tengo una duda, en regex java ¿cómo se valida en una cadena los símbolos de las llaves " { } "???

Responder

Ramón [Admin Jarroba] dice:

27/12/2016 a las 20:21

Depende de la complejidad, pero para cualquier cosa puedes utilizar:

{.*}

Un ejemplo que valida:

{valida}

Y ejemplos que NO valida:
{no valida
no valida}
no valida

Responder

Leonardo dice:

08/11/2016 a las 22:07

Hola,

Tengo el siguiente patron que necesito validar, me podrias dar una mano. gracias

1

1.1

1.1.1

1.1.1.11

([1-9]{1})(?:(\.{1})([1-9]{1}))?(?:(\.{1})([1-9]{1}))?

Responder

Ramón [Admin Jarroba] dice:

27/11/2016 a las 20:21

Para validar una IP puedes simplemente hacer: *(?:[0-9]{1,3}\.){3}[0-9]{1,3}*

Responder

Miguel Garces dice:

27/10/2016 a las 23:27

Ramon la segunda pregunta ya la solucione con el simbolo (patron2) | (patron1) solo me falta el primer punto, saludos cordiales

Responder

Miguel Garces dice:

27/10/2016 a las 22:50

hola ramon, genial tu post;

tengo dos preguntas

1. tengo un patron1 q recoge la URL patron= /**(href=")**((\w+)[^>]*(\pdf)(?=">))/gi; pero no quiero q guarde href=" aunque es impresindible para la busqueda por q de lo contrario coincide con otros, pero no lo quiero guardar, solo utilizar http//dominio/fichero.pdf

2. tengo un patron que busca las etiquetas que engloban todos los pdf q recojo en el patron1, pero

no se como hacer con los dos un solo patron q haga lo q este esquema.

esquema:

pdf tipo: <arte> (leonardo.pdf, picasso.pdf, botero.pdf....)

pdf tipo: <ciencia> (arquimedes.pdf, einstein.pdf....)

Array(arte,leonardo.pdf, picasso.pdf, botero.pdf,ciencia, arquimedes.pdf, einstein.pdf)

Gracias anticipadas

Responder

Ramón [Admin Jarroba] dice:

05/11/2016 a las 17:59

Hola Miguil. Para que no guarde utliza **(?:href=")**. Lo segundo vas a necesitar un Array con con los nombres para cada tipo, las expresiones regulares no diferencian temas abstractos (no pueden diferenciar que si le llega la palabra "picasso" quiere decir que es un artista y no un músico, por ejemplo) solo datos como tal.

Responder

Luis dice:

28/09/2016 a las 20:43

puedes explicar que lo que hace exactemente ?=

Responder

Daniel dice:

22/08/2016 a las 11:32

Hola Ramón,

Gracias por tu ayuda, he podido hacer el patron en java que quería gracias a toda la información que has puesto, pero tengo una pequeña duda: ¿Cómo puedo hacer que todo el pattern sea opcional?

Responder

Ramón [Admin Jarroba] dice:

03/09/2016 a las 16:05

Hola Daniel. La opcionalidad la puedes poner con interrogación de cierre “?” que significa o 1 o 0, es decir que puede no estar o estar una única vez (si quieres otro menos restrictivo puedes utilizar asterisco “*”, que significa o 0 o muchos). Por ejemplo, puedes tener

(hola)? a todos

que concidirá si existe o no la palabra “hola”. Por lo que coincidiría tanto “hola a todos” como ” a todos”.

Responder

Alejandro sanchez dice:

17/07/2016 a las 03:46

Hola Ramon, exelente post, necesito ayuda con una expresion regular que haga lo siguiente : procesar el archivo de texto contar las palabras del alfabeto latino incluyendo mayusculas, minusculas,tildes,digitos,simbolos aritmeticos,de relacion y todos los tipos de parentesis.El L(E) es toda cadena (palabra) que contiene las cinco vocales {a,e,i,o,u} en cualquier orden y al menos una vez cada una de ellas. Es importante hacer énfasis en que ω contiene todas las vocales, las cinco, aunque en cualquier orden (este no importa) y se pueden repetir las vocales.

Espero me puedas ayudar porfavor

saludos

Responder

Ramon dice:

19/07/2016 a las 21:14

Hola Ramon.

Necesito algo de ayuda con las expresiones regulares.

Estoy intentando que en mi web si escribo **miWeb.com/carpeta/es/** sea equivalente a **miWeb.com/carpeta/index.php?lang=es**

para ello he puesto lo siguente en el htacccess

RewriteRule **^(.*)/([a-zA-Z]{2})/$** $1/index.php?lang=$2 [L,QSA]

pero lo que consigo con esta expresion regular es miWeb.com/carpeta/index.php/es/

Como podria expluir de una cadena de caracteres un array, en este caso de la candena que saca de (.*) quiero excluir "index.php"

Gracias de antemano

Responder

Ramón [Admin Jarroba] dice:

25/07/2016 a las 14:58

El primer (.*) te debe estár eligiendo todo y luego añade el resto. Prueba con la expresión regular:

^([^\/]*)\/(([^\/]*)\/?)*$

Responder

Ramón [Admin Jarroba] dice:

25/07/2016 a las 14:52

Hola Alejandro, no podemos hacer enunciados completos, solo respondemos a dudas puntuales. Además, dicho enunciado no solo pide expresiones regulares sino una programación completa.

Responder

Jhon Sotto dice:

11/06/2016 a las 04:16

Hola Ramón, espero que todo ande muy bien para ti. Es increíble todo este buen contenido que nos compartes pero creo que es demasiado entenderlo a la primera y más con todo la presión que llevo encima en este momento por este problemilla que, si me pudieras ayudar a resolver, te agradecería toda la vida.

De la siguiente URL necesito sacar únicamente los valores de la variable origin y destination, no más. ¿Podrías ayudarme con esto por favor?
index.php?pagename=bus&origin=Cartagena&destination=Santa-Marta

Yo logré hacer este ([^=]+)$ pero sólo me saca el valor de destination.

Agradezco mucho tu ayuda.

Saludos.

Responder

Ramón [Admin Jarroba] dice:

22/06/2016 a las 17:53

Para extraer partes puedes hacer uso de paréntesis. Para tu ejemplo te bastaría con:

/origin=([^&]*)&destination=([^&]*)/g

Responder

Juan Carlos dice:

23/05/2016 a las 19:24

Hola Ramon, por favor tu apoyo, quiero obtener dentro de una cadena la primera palabra que tenga de longitud 4... por ejemplo de la cadena " de la vega puente" quiero obtener vega porque la primera es la primera palabra que tiene longitud 4 y es alfabetica.... gracias.

Responder

Ramón [Admin Jarroba] dice:

28/05/2016 a las 20:32

Puedes hacer una expresión regular que solo encuentre la primera coincidencia, como:

/ ([a-zA-Z]){4} /

Responder

Enrique Vazquez dice:

17/04/2016 a las 06:32

Hola disculpa me podrias ayudar:

Tengo esta expresion (a|b)(a|b)(a|b)*c la cual fue extraida de un automata finito, lo que quiero saber es como hacer un programa en java que me valide las diferentes cadenas validas que se pueden obtener de dicha expresion, x ejemplo: aac,aaaaac,abbac,aabbc. Estas cadenas las meteria el usuario a traves de un texfield y al darle clic en el boton validar me tendria que salir un mensaje de q si es cadena

valida, en caso contrario si no es cadena valida pues me tendria q aparecer un mensaje confirmandolo.

Apreciaria mucho q me ayudaras ya que eh estado buscando y no encuentro una solucion. Muchas gracias y saludos.

Responder

Ramón [Admin Jarroba] dice:

17/04/2016 a las 19:42

Hola Enrique, en este mismo artículo tienes ejemplos de como se hace lo que necesitas en Java. Para la entrada del usuario, dependerá de como la quieres (ventana, consola, fichero, etc). Para interfaces gráficas puedes echar un vistazo a JavaFX en http://jarroba.com/introduccion-a-interfaces-graficas-con-javafx-y-scenebuilder-video/

Responder

Andres Camilo Guzman Isaza dice:

11/04/2016 a las 22:03

Buena tarde lo que pasa es que tengo el siguiente texto:

string1
{$REGION 'mi region'}** **string2$$$4 {$ENDREGION} string4
String5
{$REGION 'mi region'} string6 {$ENDREGION}
string7
{$REGION 'mi region 2'}
nombre

edad
telefono
celular
$$$$$4
{$ENDREGION}
direccion
barrio

Y necesito porder obtener los textos que se encuentran en negrilla, he intentado de todo pero nada me funciona..

Muchas gracias...

Responder

Ramón [Admin Jarroba] dice:
17/04/2016 a las 19:15

Para capturar solo el texot que está entre caracteres de negrita () puedes utilizar una expresión regular como:

/(.*)<\/b>/g

Responder

Elesban Landero dice:

06/04/2016 a las 00:40

Hola, ¿se pueden utilizar expresiones regulares para separar en sílabas una frase?

Saludos

Responder

Ramón [Admin Jarroba] dice:

09/04/2016 a las 13:44

Por poderse se puede, solo que tendrás que trabajar en una expresión regular que será bastante grande con las excepciones, detectando las consonantes cuando van seguidas de vocal y que consonantes permiten otra consonante adyacente, etc.

Responder

Elesban Landero dice:

11/04/2016 a las 00:48

Lo intentaré, tiene que ser capaz de separarlas y además saber en que número de sílabas a sido separada. Supongo que para implementarlo en tiempo real se tendrá que hacer de alguna otra forma. Mientras se vaya escribiendo vaya separando y contado el número de sílabas escritas.

Responder

Ramón [Admin Jarroba] dice:

17/04/2016 a las 19:10

Mientras te funcione la expresión regular, luego hacerlo en tiempo real es simplemente ejecutando la función que evalua la expresión regular cada vez que lo necesites. Aunque si vas a tener mucha carga ten cuidado con las expresiones regulares, tardan bastante en procesarse (bastante carga me refiero a que en tiempo real te lleguen miles de peticiones por segundo a la función de la expresión regular).

Responder

Mar dice:

16/03/2016 a las 05:21

Exelente articulo, me sirvio demaciado

Responder

Pablo dice:

01/03/2016 a las 18:37

Hola Ramon he leido todo y se comprende bien, se agradece tu ayuda en el tema de las expresiones regulares pero me podrias ayudar con extraer datos de una direccion/domicilio?

tengo "CALLE UNO 717 SAN JOSE DE LAS CLARAS" y necesito que me entregue solo "CALLE UNO 717" en realidad las direcciones en mi pais siempre son el nombre de la calle mas un numero, se puede generar algo de este modo? Nombre de calle + numero y lo que sigue despues no me interesa solo rescatar nombre calle y numero. Seria de mucha ayuda!!!!!

Responder

Ramón [Admin Jarroba] dice:

02/03/2016 a las 10:36

Hola Pablo,

Podrías capturar la calle y el número con una expresión regular como:

\b([A-Za-z])(\d*)*

Responder

eduardo dice:

23/02/2016 a las 03:29

Buena tardes, como prodia validar que en una cadena de numeros no se permita una secuencia de 4 numeros repetidos o consecutivos por ejemplo 1234 o 1111,5555,3333,6666? agradesco su atencion

Responder

Ramón [Admin Jarroba] dice:

28/02/2016 a las 15:11

Para la detectar las secuencias de números iguales (como “2222”) podrías hacer algo como:

(\d)\1\1\1

Pero para lo que son secuencias como “1234” no te queda otra que poner todas las posibilidades que quieras reconocer como:

(1234|2345|3456 ...)

Responder

Isnel dice:

10/02/2016 a las 22:57

Buenas tardes

Con esta expresión regular ^(<(\\w+)>\\w*</(\\2)>)$ puedo validar las cadenas del tipo <tag>cadena</tag> pero no se me ocurre como hacerlo para que dentro de tag pueda tener otros tag html. Intente de la sigueinte forma ^(<(\\w+)>\\w*(\\1)*</(\\2)>)$ pero no funciono. Si se le ocurre como podria hacerse.

Saludos

Responder

Ramón [Admin Jarroba] dice:

15/02/2016 a las 10:57

Por lo que comentas necesitas validar algo como:

<a>abc<c>abc</c>

Te valdría con:

^<(\w+)>(<(\w+)>.*<\/\3>)*<\/\1>$

Responder

Hector Tarazona dice:

01/02/2016 a las 16:47

Hola Ramón un gusto., quiero validar si me puedes ayudar con esto
Muy interesante todo lo relacionado con el tema de las expresiones regulares. He tratado de resolverlo viendo distintos portales WEB que hablan sobre el tema, pero ésta no es mi area.
Tengo lo siguiente: Utilizo un portal de atención al cliente en el que cada vez que reporta algun caso referente a los productos que vendemos, debe llenar un formulario, entre otras cosas el formulario incluye el serial del articulo. dependiendo del producto hay un patron de serial que se debe colocar. Por ejemplo: Tabletas Modelos QD100BT, tienes el patron de serial **QTXX–XXXXXX** (Las x son numeros) Las cornetas tienes éste patron: **CX–XXXXXXXXXXX**, y asi, unos cuantos productos mas**.**
La expresion regular creada para todas las alternativas de seriales es ésta: ^[A-Za-z]{2}(\-\d{11}|[A-Za-z]\d{2}\-\d{6}|\d{7}|\d{2}\-\d{6})$. y hasta ahora funciona.
El problema que tengo es que me necesito agregarle a la expresión expuesta un patron de serial nuevo. El patron sería QXXXXXXXXXXXX (1 letra y 13 numeros) y a su ves agregarla a la expresion regular ya existente para que funcionen tanto en los seriales antiguos como en el nuevo patron.
Si hay alguna forma de que me ayudes te lo agradecere.
Atte.-
Hector Tarazona
Caracas, Venezuela

Responder

Ramón [Admin Jarroba] dice:

07/02/2016 a las 21:50

Buenas Hector.

El patrón que pones puede validar:

AB-12345678901
ABC12-123456
AB1234567
AB12-123456

Pues bien una modificación (más bien una extensión a tu expresión regular) simple para que también te incluya por ejemplo “A1234567890123” la que necesitas podría ser:

([A-Za-z]{2}(\-\d{11}|[A-Za-z]\d{2}\-\d{6}|\d{7}|\d{2}\-\d{6}))|([A-Za-z]\d{13})

Responder

Damian dice:

22/12/2015 a las 18:43

Hola, necesito lo siguiente:

Escribir una expresión regular para testear si una oración tiene una palabra repetida (por ejemplo

“Una casa blanca” no pasa el test pero “Una casa y otra casa más” si pasa el test.

Responder

Ramón [Admin Jarroba] dice:

22/12/2015 a las 21:15

Podrías hacer algo como:

/\b(\w+)\b(?=.\b\1)\b/g*

Que encuentras una palabra siempre que le sigua otra igual (es decir te va a contar todas las palabras repetidas menos la última que es la que se utiliza para comparar).

Responder

Carlos dice:

18/12/2015 a las 04:25

Hola Ramon me parece excelente tu ayuda

Estoy tratando de extraer (en PCRE) el tercer octecto de una direccion IP por ejemplo

10.20.6.8 -> 6

10.20.11.9 -> 11

logre extraer los dos ultimos octetos con la siguiente expresion \d{1,3}\.(\d{1,3})$ y me parece que

podria usar ^\d{1,3} para extraer el primer octecto de manera recursiva, lo que pasa es que no me queda claro como anidar ambas expresiones.

si tienes alguna forma mas eficiente te lo agradeceria

Saludos.

Responder

Ramón [Admin Jarroba] dice:

20/12/2015 a las 12:14

Tienes varias maneras de obtener el tercer octeto, por ejemplo:

^(?:\d{1,3}\.){2}(\d{1,3})

Reconoce hasta el tercer octeto, pero solo guarda el tercer octeto, por lo que solo te queda usarlo con el lenguaje que lo estés utilizando.

O encontrar lo que quieras diciéndole que solo si le sigue el resto, es decir, empezar por el final:

(\d{1,3})(?=(?:\.\d{1,3}){1}$)

Con esta última te encuentra justo el tercer octeto.

Aunque yo preferiría el primero modificándolo para que extraiga los 4 octetos, y con el lenguaje de programación utilizar el octeto que necesite, es más reutilizable para el futuro 🙂

Responder

Rodrigo dice:

13/12/2015 a las 05:02

Saludos Ramón. Tengo un problema para remplaza una cadena. Necesito reemplazar todas los saltos de línea que no terminen en punto y sustituirlos por **"/"**. Para ello he usado el patrón

> **[^\.]\r\n**

y lo sustituyo con

> **\/**

pero al remplazar la cadena sustituye el último carácter.

La cadena de ejemplo sería ésta:

> *Lo cual una vez impuso a las mesas, yo con mi justiciera llama*

> *sobre unos penates dignos de su dueño torné sus techos.*

Al sustituir elimina la a de "llama" y o que obtengo es:

> *Lo cual una vez impuso a las mesas, yo con mi justiciera llam / sobre unos penates dignos de su dueño torné sus techos*

Pero lo que necesito es:

> *Lo cual una vez impuso a las mesas, yo con mi justiciera llama / sobre unos penates dignos de su dueño torné sus techos.*

Estoy usando Notepad++ . Espero que quedas ayudarme, gracias.

Responder

Ramón [Admin Jarroba] dice:

13/12/2015 a las 19:21

He probado

[^\.]\n

y funciona correctamente detectando los saltos que no terminen en punto 🙂

Responder

Rodrigo dice:

14/12/2015 a las 20:38

Mil gracias Ramón, que sencullo era :). Me funcionó a la perfección. Me encanta cuando la solución usa menos código.

Responder

Freddy dice:

04/12/2015 a las 19:54

Ramon, estoy un poco enredado, como encuentro varios patrones cuando se repiten en un archivo de texto?: Ej:

Data

Region 1
2
-78.78381 -3.782565
-77.663121 0.663736
Pen (1,2,0)
Brush (2,16777215,16777215)
Center -77.031933 -1.696014
Region 1
3
-78.78381 -3.782565
-78.78381 -0.886686
-77.727308 0.880292
Pen (1,2,0)

Brush (2,16777215,16777215)
Center -78.219621 -1.447138
Region 1
1
-78.78381 -0.886686
Pen (1,2,0)
Brush (2,16777215,16777215)
Center -78.255559 0.202941

He intentado con la siguiente expresión regular, pero obtengo todo el conjunto, y realmente lo que necesito es agrupar por "Region y coordenadas"

"Region[\s\S]*Region"

"(Region[\s\S]*Region)"

Responder

Ramón [Admin Jarroba] dice:

04/12/2015 a las 21:23

Para encontrar las coordenadas puedes utilizar:

((\-)?\d+\.\d+)

Para la Region no entiendo que necesitas, solo veo “Region 1” en el ejemplo :S

Responder

Pepe dice:

28/11/2015 a las 17:48

Estoy estudiando el tema de validaciones de formularios con patrones regulares, y es el mejor post que he leido sobre este tema. Mi más sincera enhorabuena por lo bien explicado que está, y completo!!

Ya el remate sería tener ya las típicas validaciones de campos de registro de usuarios. [nombre, apellidos, direccion, codigo postal, poblacion, provincia, teléfono movil, fecha de nacimiento, hora, etc, etc 🙂]

De nuevo, muchas gracias por el post!!. Ya tienes un seguidor más!!

Responder

Ramón [Admin Jarroba] dice:

29/11/2015 a las 19:39

Hola Pepe. Muchas gracias por los ánimos se agradecen un montón 🙂

Al final del artículo hay un ejemplo de validación de números de teléfono, el resto de validaciones las dejamos como ejercicio 😉 aunque si tienes cualquier duda o alguna que no te salga me puedes preguntar y te echo una mano.

También, si vas a hacer una web, muchas de las validaciones (expresiones regulares más comunes) con poner el tipo (type) de la etiqueta “input” como *<input type=”email” name=”correo electrónico”>* ya valida automáticamente por lo que nos podemos ahorrar diseñar algunas expresiones regulares.

Responder

Alberto dice:

27/09/2015 a las 17:02

Hola muy buena tu web, me podrias ayudar en construir una expresion regular en javascript para validar apellido,nombre
Pero eso lo puedo escribir varias veces separados por ;
ejemplo Ruiz,Cardoso;Ruiz,González;Perera,Ramos

Responder

Ramón [Admin Jarroba] dice:

28/09/2015 a las 17:50

Supongo que solo quieres validar letras. Con la siguiente expresión regular te encuentra las palabras de todo el String:

/([A-Za-záéíóú]*,[A-Za-záéíóú]*)/g

Responder

Frank dice:

16/09/2015 a las 21:30

para validar solo numeros:

Pattern patm = Pattern.compile("[^A-Za-z.@_~#\\-\\|\\¬\\°\\!\"\\%\\$\\/\\(\\)\\=\\?\\¿\\'\\¡\\¨\\´*\\+\\{\\}\\[\\]\\^\\`\\,\\;\\:\\+\\á\\é\\í\\ó\\ú\\Á\\É\\Í\\Ó\\Ú\\Ä\\Ë\\Ï\\Ö\\Ü\\ä\\ë\\ï\\ö\\ü\\à\\è\\ì\\ò\\ù\\<\\>]+");

Ya lo probe y funciona pero, ¿¿Como se puede optimizar esta linea de codigo??

Responder

Ramón [Admin Jarroba] dice:

16/09/2015 a las 21:33

Hola Frank. Si es para validar números me parece un poco complicado, ¿Has probado con \d , que valida números?

Responder

Hugo dice:

08/09/2015 a las 17:31

Hola Ramon estoy validando cajas con expresiones regulares, y necesito validar un rango de 5 a 40 q ya lo resolvi y una excepcion que es 99
pero no puedo hacer andar la excepcion... please ayuda! te dejo la linea de codigo.... ^[5-9]|[1-3][0-9]|40|99$ GRACIAS!

Responder

Ramón [Admin Jarroba] dice:

12/09/2015 a las 17:36

Te faltan los paréntesis: **^([5-9]|[1-3][0-9]|40|99)$**

Responder

Taner dice:

28/08/2015 a las 08:42

Hola Ramon, tu pagina es muy buena, gracias. Tengo un objeto JSON asi;
"text": "Esto es "un" ejemplo.",

pero tengo que quitar o escapar las comillas de "un". quiero que sea asi; "text": "Esto es un ejemplo",

Como puedo hacer lo ? Por favor ayuda.

Un saludo cordial desde Turqia,

Responder

Ramón [Admin Jarroba] dice:

28/08/2015 a las 20:13

Tienes de pista delimitadora los conjuntos de caractéres que te pongo entre paréntesis: para abrir (**: “**) y para cerrar (**“,**). Por tanto, solo tienes que buscar todas las comillas que están en el interior de estos dos conjuntos que no van a cambiar (siendo un JSON podría cambiar la última coma, sería buscar las comillas del final).

Te he creado esta expresión regular:

/"\b([^"]*)\b"(?!:)/

que lo que hace es encontrar todo par de comillas internos a comillas que no sigan a dos puntos; es decir, que de tu ejemplo:

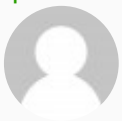
"text": "Esto es "un" ejemplo.",

encontraría

"un"

ya solo te queda aplicar sustitución para quitar las comillas, vamos que te lo he dado hecho 😉

Responder



Taner dice:

31/08/2015 a las 09:11

Grande Ramon ! mil gracias !! Un saludo cordial, 🙂

Responder

Taner dice:

31/08/2015 a las 11:08

Hola de nuevo Ramon, lo he comprobado asi, pero no encuentra. O ne se puede utilizar asi en java ? Gracias

```
String input = "\"text\": \"esto es \"un\" ejemplo\",";

if (input.contains("/"\\b([^"]*)\\b"(?!:)/")) {
   System.out.println("contains");
}else System.out.println("no contains");
```

Responder

Ramón [Admin Jarroba] dice:

12/09/2015 a las 17:21

El método contains solo admite textos, no expresiones regulares: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#contains(java.lang.CharSequence)

Ale dice:

24/06/2015 a las 05:33

Post bestial! Tengo que leerlo con tiempo 😀

Responder

Ramón [Admin Jarroba] dice:

24/06/2015 a las 21:11

Con calma y motivación. En vez de resolver sudokus serán expresiones regulares 😀

Responder

JOSE ANTONIO GAMBOA AGUIRRE dice:

23/05/2015 a las 16:28

que buena entrada, sigo sus tutoriales casi siempre, me surge la duda de como podria modificar el limite de ocurrencias de un patron es decir
"abbb" necesito corraborar que este patron aparezca almenor 5 veces
Muchas gracias por su pagina.

Responder

Ramón [Admin Jarroba] dice:

23/05/2015 a las 18:28

Hola José, gracias 🙂

Tienes varias maneras para hacerlo. Para que se encuentre 5 veces seguidas puedes o bien poner {5}. O si lo que necesitas es que aparezca en varias partes almenos 5 veces con .* entre medias de las 5 veces repetidas. También puedes encontralo 1 vez y por código contar las coincidencias.

Responder

Javier dice:

06/04/2015 a las 21:40

Un artículo muy elaborado, si señor. Actualmente estoy intentando familiarizarme con todo esto de

las expresiones regulares para la validación de formularios, por ejemplo, y tengo una duda que no consigo averiguar. Usando un caso concreto, validar una contraseña, no comprendo como hacer que los parámetros que ponga en la expresión regular no se tenga que cumplir en ese orden exactamente. Por ejemplo, quiero que la contraseña tenga letras mayúsculas, minúsculas, números y también alguno de estos caracteres(;,.-). Solo consigo que .match me la de por válida cuando la contraseña tiene el mismo orden, por ejemplo, (Aa1;) pero no es válidad con (aA;1). No se si me llegas a entender y si puedes ayudarme a comprender esta situación.

Un saludo y enhorabuena por tu post.

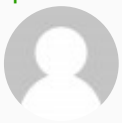
Responder

Ramón [Admin Jarroba] dice:

07/04/2015 a las 09:17

Hola Javier. Para lo que necesitas te valdría con: [A-Za-z0-9;,\.-]+

Responder



Javier dice:

07/04/2015 a las 11:30

Con esa regex no lo he conseguido, pero con esta si:

/^.*(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[;,\.-]).*$/

Responder

Deja un comentario

Tu dirección de correo electrónico no será publicada. Los campos obligatorios están marcados con *

Comentario

Nombre *

Correo electrónico *

Web

Publicar comentario

Siguenos en:

FAQ

VISITAS

GASTOS

SITE-MAP

Contacto:
jarrobaweb@gmail.com

By Reimon & Richard