# Tema 6

## Expresiones regulares

# Expresiones Regulares

Las expresiones regulares constituyen un mecanismo bastante potente para realizar manipulaciones de cadenas de texto. El proceso para el que se usan estas expresiones, presente en el mundo el UNIX y el lenguaje Perl, es el de buscar y/o substituir una subcadena de texto dentro de otra cadena. En principio esto puede hacerse usando los métodos del objeto string, pero el problema surge cuando no tenemos una subcadena fija y concreta sino que queremos buscar un texto que responda a un cierto esquema, como por ejemplo: buscar aquellas palabras que comienzan con http: y finalizan con una \, o buscar palabras que contengan una serie de números consecutivos, etc.; es en estos casos cuando tenemos que utilizar las expresiones regulares.

La subcadena que buscamos en el texto es lo que se llama un patrón y se construye encerrando entre dos barras inclinadas ( / ) una serie de caracteres normales y símbolos especiales llamados comodines o metacaracteres, (algo parecido a buscar archivos con nombre *.bat cuando queremos encontrar los ficheros con extensión bat). Este patrón es una descripción del texto que se está buscando y JavaScript encontrará las subcadenas que concuerdan con ese patrón o definición. Las expresiones regulares se usan con el objeto Regular Expresion y también dentro de los métodos String.match, String.replace, String.search y String.split.

En la tabla que sigue se muestran los caracteres comodín usados para crear los patrones y su significado, junto a un pequeño ejemplo de su utilización.

| | **Significado** | **Ejemplo** | **Resultado** |
|---|---|---|---|
| \ | Marca de carácter especial | /\$ftp/ | Busca la palabra $ftp |
| ^ | Comienzo de una línea | /^-/ | Líneas que comienzan por - |
| $ | Final de una línea | /s$/ | Líneas que terminan por s |
| . | Cualquier carácter (menos salto de línea) | /\b.\b/ | Palabras de una sóla letra |
| \| | Indica opciones | /(L\|l\|f\|)ocal/ | Busca Local, local, focal |
| ( ) | Agrupar caracteres | /(vocal)/ | Busca vocal |
| **[ ]** | Conjunto de caracteres opcionales | /escrib[aoe]/ | Vale escriba, escribo, escribe |
| [^ ] | No coincide con caracteres | /escrib[^aoe]/ | Vale escribu, escribi |

La tabla que sigue describe los modificadores que pueden usarse con los caracteres que forman el patrón. Cada modificador actúa sobre el carácter o el paréntesis inmediatamente anterior.

| | Descripción | Ejemplo | Resultado |
|---|---|---|---|
| * | Repetir 0 ó más veces | /l*234/ | Valen 234, 1234, 11234... |
| + | Repetir 1 ó más veces | /a+mar/ | Valen amar, aamar, aaamar... |
| ? | 1 ó 0 veces | /a?mar/ | Valen amar, mar. |
| {n} | Exactamente n veces | /p{2}sado/ | Vale ppsado |
| {n,} | Al menos n veces | /(m){2}ala/ | Vale mmala, mmmala.... |
| {m,n} | entre m y n veces | /tal{1,3}a/ | Vale tala, talla, tallla |

Los siguientes son caracteres especiales o metacaracteres para indicar caracteres de texto no imprimibles, como puedan ser el fin de línea o un tabulador, o grupos predefinidos de caracteres (alfabéticos, numéricos, etc...)

| | Significado | Ejemplos | Resultado |
|---|---|---|---|
| **\b** | Principio o final de palabra | /\bver\b/ | Encuentra ver en "*ver de*", pero no en "*verde*" |
| **\B** | Frontera entre no-palabras | /\Bver\B/ | Empareja ver con "*Valverde*" pero no con "*verde*" |
| **\d** | Un dígito | /[A-Z]\d/ | No falla en "A4" |
| **\D** | Alfabético (no dígito) | /[A-Z]\D/ | Fallaría en "A4" |
| **\O** | Carácter nulo | | |
| **\t** | Caracter ASCII 9 (tabulador) | | |
| **\f** | Salto de página | | |
| **\n** | Salto de línea | | |
| **\w** | Cualquier alfanumérico, [a-zA-Z0-9_ ] | /\w+/ | Encuentra *frase* en "*frase*.", pero no el . (punto). |
| **\W** | Opuesto a \w ([^a-zA-Z0-9_ ]) | /\W/ | Hallaría sólo el punto (.) |
| **\s** | Carácter tipo espacio (como tab) | /\sSi\s/ | Encuentra *Si* en "*Digo Si ",* pero no en "*Digo Sientate*" |
| **\S** | Opuesto a \s | | |
| **\cX** | Carácter de control X | \c9 | El tabulador |
| **\oNN** | Carácter octal NN | | |
| **\xhh** | El hexadecimal hh | /\x41/ | Encuentra la *A* (ASCII Hex41) en "*letra A*" |

# Objeto. Expresiones Regulares

JavaScript usa este objeto para trabajar con patrones de expresiones regulares, estos patrones se crean como cualquier otro objeto mediante su inicialización directa o bien mediante el constructor new RegExp(), como podemos ver en el ejemplo:

```
......
var mipatron = /^[aeiou]/gi;
var mipatron2 = new RegExp("^[aeiou]", "gi");
........
```

Ambas formas conducen al mismo patrón, en este ejemplo define palabras que comienzan con una vocal. El patrón puede llevar modificadores o flags para matizar la forma de buscar, en

el ejemplo se usan dos: g i Estos modificadores tienen los siguientes significados:

| **flags** | **Significado** |
|---|---|
| g | Explorar la cadena completa |
| i | No distinguir mayúsculas de minúsculas |
| m | Permite usar varios ^ y $ en el patrón |
| s | Incluye el salto de línea en el comodín punto . |
| x | Ignora los espacios en el patrón |

Los patrones son algo así como operadores y usan una serie de símbolos con significado especial, que hemos visto en el apartado anterior.

El objeto RegExp posee en total tres métodos exec(), test(), compile() además de los métodos ya citados del objeto String que también usan patrones como son: match(), replace(), search() y split(). La única propiedad que funciona en estos objetos es la source que refleja el contenido del patrón.

En el patrón pueden aparecer caracteres o grupos de caracteres encerrados entre paréntesis, posteriormente podemos usar un índice para referirnos individualmente al contenido de esos paréntesis.

Por ejemplo vamos a substituir por un - todas los letras situadas tras un dígito en la cadena a explorar.

```
var cadexp = "234879x089h9ys7";
var patron = /(\d)([a-z])/ig;       ///(|d) → $1 y ([a-z]) → $2
document.write(cadexp+'<br> ');
cadexp = cadexp.replace(patron, "$1-");
document.write(cadexp)
```

```
Resultado

234879x089h9ys7
234879-089-9-s7
```

Como ves donde antes existía un dígito seguido de una letra ahora hay un dígito seguido de un guión. Las coincidencias con el primer paréntesis del patrón están en $1 y con el segundo en $2. La primera coincidencia hallada es 9x, luego $1 contiene 9 y $2 contiene x, sustituyo lo hallado con $1-, o sea, quito $2 y pongo un guión y me quedará 9- en lugar de 9x. Como se ha usado el flag g (global) esta operación se realiza en toda la cadena.

```
exec(cadexplor)
```

Este método busca la primera concordancia del patrón con el contenido de la cadena de texto donde se busca, que se le pasa como argumento. Si no encuentra ninguna concordancia devuelve null, pero si encuentra una secuencia de caracteres que se adapte al patrón de búsqueda devuelve un array cuyo primer elemento indica la concordancia encontrada y las restantes indican los resultados de acuerdo a los paréntesis que aparezcan en la expresión regular. Además este array posee dos propiedades:

**index**, para indicar la posición de la subcadena encontrada,

**input**, que contiene la cadena de caracteres que se está explorando.

Además modifica las propiedades de una variable global RegExp con datos relativos a la búsqueda.

En el ejemplo que sigue buscamos cualquier letra seguida de un número y de un guión, el patrón de búsqueda será /[a..z]\d-/i, [a..z] representa todas las letras del abecedario, \d representa cualquier número y el modificador i se usa para no diferenciar mayúsculas de minúsculas.

```
patron = /[a-z]\d-/i;
var busca = new Array()
busca = patron.exec("3c491a-9d1d6-91br");
if (busca != null){
   document.write("Concuerda en: "+busca.index + '<br>');
   document.write("Explorando:" +busca.input + '<br>');
   document.write("Hallado: " + busca[0] + '<br>');
}
document.write("Resto " + RegExp.rightContext + '<br>');

Resultado
Concuerda en: 10
Explorando:3c491a-9d1d6-91br
Hallado: d6-
Resto 91br

test(cadexp)
```

Este es el método más simple del objeto expresión regular, tan sólo comprueba si existe alguna coincidencia en la cadena explorada pasada como argumento. Si existe tal coincidencia devuelve un valor booleano true y en caso contrario devuelve false.

```
var patron = new RegExp("Lunes","gi");
var cadexpl = "La reunion es el lunes o el martes.";
var eslunes = patron.test(cadexpl);
document.write("Es el lunes? "+eslunes+'<br>');
document.write("Hallado en "+patron.lastIndex+"<br />");
document.write("Cadena buscada "+patron.source);
Resultado
Es el lunes? true
Hallado en 22
Cadena buscada Lunes
```

En este sencillo ejemplo se comprueba si la cadena explorada, cadexpl, contiene la palabra "lunes", sin considerar la caja (mayúsculas o minúsculas). El resultado lo guarda en la variable eslunes.

```
compile(cadpatr)  Obsoleto - en desuso
```

Un patrón de búsqueda puede construirse mediante una simple asignación o mediante el constructor new RegExp y ser utilizada tal cual, pero se puede mejorar bastante la búsqueda usando este método que convierte el patrón en un formato interno para optimizar su uso. Utiliza como argumento una cadena que representa la expresión regular que se quiere compilar

```
var patron = new RegExp();
patron.compile("\\D-");
var busq = patron.exec("1234u90t-789");
document.write('Buscando '+patron.source+'<br>');
document.write(busq[0]+'  esta  en  la  posicion  '  +  busq.index  +'  de
'+busq.input);
Resultado
Buscando \D-
t- esta en la posicion 7 de 1234u90t-789
```

En este ejemplo se busca cualquier no numérico seguido de un guión en la cadena "1234u90t-789". Primero se declara la variable patrón y se compila con el patrón \D- que indica cualquier carácter no numérico seguido de guión. Por último muestra el patrón usado y los resultados de la búsqueda: coincidencia encontrada, posición y cadena explorada.

## String

### match( expreg )

Este es uno de los más potentes métodos para buscar subcadenas dentro de cadenas de texto. Permite usar patrones de búsqueda construidos con comodines y texto, lo que se denominan expresiones regulares. Este método usa como argumento una expresión regular y va buscando en el objeto alguna subcadena que concuerde con esa expresión. Esta subcadena la devuelve en un array. Si no encuentra ninguna devuelve null.

Por ejemplo:

```
frase="Busco palabras con menos de cinco letras";
result=frase.match(/\b\w{1,4}\b/g);
document.write("Hallados: "+result+"<br>");
document.write("En la frase: " + RegExp.input);
```

Si pruebas el ejemplo obtendrás el siguiente listado

```
Hallados: con,de
En la frase: Busco palabras con menos de cinco letras
```

El patrón de búsqueda está encerrado entre dos barras / , y busca caracteres alfanuméricos ( \w ) comprendidos entre límites de palabras ( \b ) además hace una búsqueda global (indicado por la g fuera de las barras).

### replace( expreg, nueva )

Con este método todas las cadenas que concuerden con la expreg del primer argumento son reemplazadas por la cadena especificada en el segundo argumento, nueva, que puede contener elementos del patrón mediante los símbolos $1 a $9. El resultado devuelto es la cadena con las sustituciones realizadas. Por ejemplo vamos a cambiar palabra por frase en la frase "Cada palabra dicha es una palabra falsa"

```
linea="Cada palabra dicha es una palabra falsa";
linea = linea.replace(/palabra/g, "frase");
document.write(linea);
```

Si pruebas el ejemplo obtendrás lo siguiente

```
Cada frase dicha es una frase falsa
```

En esta ocasión se ha usado un patrón con el modificador g de global por lo que cambia todas las coincidencias, si se omite sólo se cambia la primera. En la cadena nueva pueden usarse elementos del patrón, por ejemplo cambiemos las negritas a cursivas en la frase:

```
var patron = /(<b>)([^<]+)(<\/b>)/g;
var frase = "Cada <b>negrita</b> pasa a <b>italica</b>";
document.write(frase+"<br>");
newstr = frase.replace(patron, "<i>$2</i>");
document.write(newstr);
```

veras la frase antes y después del cambio:

```
Cada negrita pasa a itálica

Cada negrita pasa a itálica
```

El $2 es un índice referido a los paréntesis del patrón, así $1 indica lo contenido en el primer paréntesis (<b>) mientras que $3 es <\b>, el tercer paréntesis.

## Patrones especiales para String.replace

| Patrón | Propósito |
|---|---|
| $$ | Permite remplazar literalmente el signo dolar($) |
| $& | Inserta la subcadena coincidente |
| $` | Inserta la parte de la cadena antes de la coincidencia |
| $’ | Inserta la parte de la cadena después de la coincidencia |
| $n | Inserta todos los valores capturados por paréntesis en la expresión regular. |

### search( expreg )

Es un método similar al método match pero más rápido. Este método realiza una búsqueda en la cadena y devuelve el índice donde se produce la primera concordancia con el patrón o -1 si no existe ninguna. Por ejemplo buscamos las cadenas 'lunes' o 'martes' en la frase cita, la letra i del patrón indica que se debe ignorar el tipo mayúsculas o minúsculas en la búsqueda:

```
var patron = /lunes|martes/i;
var cita = "La reunion sera el lunes y el miercoles";
document.write(cita.search(patron)+"<br>");
```

Si pruebas el ejemplo obtendrás un 19, la posición de la palabra 'lunes'.

### split( separ)

Devuelve un array conteniendo las porciones en que queda separada la cadena por el separador indicado mediante el argumento separ, que será una expresión regular o una cadena literal. Si este separador es una cadena vacía el texto queda desglosado en todos sus caracteres. Si se omite el separador el resultado es un array de un elemento con la cadena completa.

```
var linea=new String("Título: El portero");
var lista = linea.split(/:\s*/);
```

La variable lista es un array con dos elementos "Título" y "El portero". También podríamos haberlo escrito como

```
var linea=new String("Título: El portero");
lista = linea.split(":");
document.write(lista);
```

en este caso el primer elemento de lista es "Título" y el segundo " El portero" con un espacio por delante. Por último si el separador es una cadena vacía:

```
var linea=new String("Título: El portero");
lista = linea.split("");
document.write(lista);
```

la variable lista contendrá T,í,t,u,l,o,:, ,E,l, ,p,o,r,t,e,r,o.

# EJERCICIO OPCIONAL

Ej 1 - Crear las expresiones regulares necesarias para resolver los siguientes puntos:

- Valida si una cadena introducida es un número entero.
- Validar DNI, 8 números y una letra al final.
- Validar una matrícula de un coche con formato 0000XXX
- Valida nombre de usuario en twitter, debe de empezar por @ y puede contener, números, letras mayúsculas y minúsculas, “-“ y “_”.
- Crear una expresión regular que valide una fecha en formato "XX/XX/XXXX", donde cada "X" es un dígito. Se puede probar con expresiones tipo: "El día 29/11/2019 tenemos examen.".
- Crear una expresión regular para la validación de direcciones de correo electrónico. Para simplificar, los valores anteriores a @ pueden contener cualquier carácter alfanumérico, y los caracteres “.” y “-”, mientras que los valores después de la @ pueden contener caracteres alfanuméricos, y el tipo de dominio tendrá una longitud de 2 o 3 caracteres.
- Crear una expresión regular que elimine las etiquetas potencialmente peligrosas (<script>...</script>) y todo su contenido de una cadena HTML.
- Crea una expresión regular que dado un número de cuenta IBAN en formato ESXXXXXXXXXXXXXXXXXXXXX nos lo devuelva en porciones de 4 dígitos separado por un “-“.

Ej 2 – Genera dos ejercicios donde te parezca interesante el uso de expresiones regulares.