Materia: Tecnologías para la Web.
Tema: JavaScript: Expresiones Regulares.

Expresiones Regulares en JavaScript

¿Qué son las expresiones regulares?

Una expresión regular es una forma de definir una 'plantilla', con una sintaxis propia, para localizar un patrón dentro de un texto. El ejemplo más sencillo sería definir una palabra como plantilla:

Expresión regular: `/casa/`

Encontraría la palabra *casa* en el texto:

```
La casa de la montaña.
```

La utilidad real se muestra cuando utilizamos caracteres especiales para crear patrones de búsqueda mucho más complejos, del tipo:

Expresión regular: `/\W*\s(\S)*$/`

Parece mucho más complicado de lo que es en realidad. Sólo hay que aprender unas reglas básicas y unos cuantos caracteres especiales.

JavaScript tiene dos notaciones

En JavaScript hay dos formas de crear una expresión regular: mediante la **notación literal** o utilizando el constructor **RegExp()**:

```
//notación literal
var myRegExp = /casa/g; /* /reg. expression/flags(g|i|m) */

//Usando el constructor RegExp
var myRegExp=new RegExp('casa','g');/*RegExp('expression','flags(g|i|m)');*/
```

Las dos son equivalentes. La primera se compila cuando el script se carga y es más rápida. La segunda se compila en tiempo de ejecución por lo que es la única opción si la expresión contiene variables que tienen que ser interpoladas. Aquí se utilizará la **notación literal** por simplicidad. Todas las explicaciones son válidas también para el constructor.

Para entender mejor el funcionamiento de las expresiones regulares conviene pensar en una búsqueda **carácter a carácter**. Si tenemos el patrón `'casa'`, lo que estamos buscando no es la palabra *casa*, sino una `'c'` seguida de una `'a'` seguida de una `'s'`, etc. Es mucho más fácil entender el uso de los caracteres especiales de esta manera.

Por ejemplo, si se sabe que **\s** representa un espacio en blanco y que **?** significa que el carácter anterior es opcional, ¿qué se está seleccionando en la siguiente expresión regular?:

```
var myRegex = /\sglobos?/;
```

Se está buscando un espacio en blanco, seguido de una 'g', de una 'l', de una 'o', de una 'b', de una 'o' y de una 's' que es opcional, puede aparecer o no. Por tanto se selecciona:

```
Un globo, dos globos, tres Globos.
```

Se puede ver que sólo ha seleccionado el primer 'globo'. Por definición la búsqueda termina cuando se encuentra la primera coincidencia. Si se desea que se siga buscando se debe utilizar el *flag* **g**.

1.

Flags (g, i, m)

Los *flags* modifican el comportamiento predeterminado de la búsqueda. Aparecen justo al lado de la barra que marca el final de la expresión regular y pueden ser:

- **g** (global). Se busca siempre en el texto completo en vez de detenerse cuando encuentra la primera coincidencia con el patrón.
- **i** (ignora Mayúsculas). No diferencia mayúsculas y minúsculas.
- **m** (multilínea). El texto incluye saltos de línea. ^ y $ se aplican a 'comienzo de línea' y 'final de línea', en vez de 'comienzo de texto' y 'final de texto'.

Si en la cadena de nuestro ejemplo se desea seleccionar también otras ocurrencias de la palabra 'globo', se debe utilizar:

```
var myRegex = /globo/g;
```

Un globo, dos globos, tres Globos.

La expresión encontraría dos coincidencias. El tercer 'Globo' no coincide con el patrón porque la G es mayúscula. Si se añade el modificador **i** sí se seleccionará:

```
var myRegex = /globo/gi;
```

Un globo, dos globos, tres Globos.

La segunda y tercera aparición están en plural. Sería bueno que la expresión regular capturara tanto *globo* como *globos*. Como ya se revisó antes, se puede utilizar la **?** para indicar que el carácter anterior es opcional:

```
var myRegex = /globos?/gi;
```

Un globo, dos globos, tres Globos.

2.

?, + , * Cuantificadores sobre el carácter anterior

Con estos tres caracteres extraños se puede indicar que el carácter **inmediatamente anterior** puede ser opcional, o que tiene que aparecer como mínimo una vez o que puede aparecer muchas veces.

?	El carácter anterior es opcional. Aparece 0 o 1 veces.
+	El carácter anterior aparece una vez o más.
*****	El carácter anterior puede no aparecer, aparecer una vez o aparecer repetido muchas veces (aparece 0 o más veces).

Si se tiene el texto:

```
"Tengo un glooooooobo grande y tres globos pequeños."
```

¿Cómo hacer para capturar los dos globos? La expresión anterior no sirve porque la primera aparición quedaría fuera.

Se necesita una expresión regular que seleccione cualquier patrón con una 'g' seguida de una 'l', seguida de **una o más 'o'**, una 'b', una 'l', una 'o' y una 's' opcional.

```
var myRegex = /glo+bos?/gi;
```

Tengo un gloooooobo grande y tres globos pequeños.

En este caso, si se pone ***** en vez de **+** también funcionaría. La diferencia es que el asterisco permite que el carácter no aparezca (cero o más veces). Por lo tanto también seleccionaría **glbo**. Con **+** se obliga a que el carácter aparezca al menos una vez.

Para seleccionar todas las palabras en el siguiente texto:

```
"bo, boooo, b."
```

Sólo nos sirve el *****:

```
var myRegex = /bo*/g;
```

bo, boooo, b.

3.

'.' Un comodín para un solo car.cter

El punto '.' es un comodín que puede sustituir a **cualquier carácter excepto a un salto de línea**.
La siguiente expresión:

```
var myRegex = /gat./gi;
```

Capturará 'gat' seguido de cualquier carácter:

Gato, gata, gamo, gatuno, gatitos, gatooooos, gatk, gallego, gat , gat.

En las dos últimas coincidencias se puede ver que también captura el espacio en blanco y el punto final.

4.

{n,m} Cuantificadores más concretos

Con {n,m} se puede indicar exactamente el número de veces que aparece el carácter anterior o delimitarlo en un rango concreto (por ejemplo, que aparezca entre 3 y 5 veces).

{n}	El carácter anterior aparece exactamente *n* veces.
{n,}	El carácter anterior aparece *n* o más veces.
{n,m}	El carácter anterior aparece un mínimo de *n* y un máximo de *m* veces

```
var myRegex = /r{2}/gi;
```

b, brr, brrr, brrrr.

En el último caso tenemos *brrrr*. Se seleccionan las dos primeras y después las dos segundas.

```
var myRegex = /r{2,}/gi;
```

b, brr, brrr, brrrr.

```
var myRegex = /r{2,3}/gi;
```
b, brr, brrr, brrrr.

5.

[abc] Conjuntos de caracteres

Para indicar que en un lugar de nuestra expresión puede aparecer cualquier de los caracteres de un conjunto concreto podemos utilizar **[ ]**. Por ejemplo, si queremos seleccionar 'sal', 'cal' y 'mal', podemos indicar que el primer carácter puede ser [scm]. Es decir, puede ser **uno** de los caracteres del conjunto:

```
var myRegex = /[scm]al/gi;
```
tal, sal, perro, mal, sota, cal, tal, salir, ramal.

También podemos **negar el conjunto** con el carácter ^, indicando que puede aparecer cualquier carácter **menos los que están en el conjunto**.

```
var myRegex = /[^scm]al/gi;
```
tal, sal, perro, mal, sota, cal, tal, salir, ramal.

Se puede especificar un rango mediante un guion: **[0-7]** indica números del 0 al 7, **[a-z]** indica letras de la 'a' a la 'z'

6.

'^' y '$' Principio y final de línea

Muchas veces es necesario seleccionar una cadena solamente si está al principio o al final de un texto. Por ejemplo, es muy común eliminar los espacios en blanco que puedan aparecer al principio o al final. Sabemos que el símbolo **\s** selecciona un espacio en blanco. Con la siguiente expresión se seleccionarían todos los espacios al principio de un texto:

```
var myRegex = /^\s+/g;
```

Se está buscando un 'principio de texto' seguido de uno o más espacios en blanco:

  Este es mi texto.

Si se desea seleccionar los del final:

```
var myRegex = /\s+$/g;
```

Estamos buscando uno o más espacios en blanco justo antes de un final de texto:

Este es mi texto.

En el apartado de *flags* vimos que existe uno para modificar el comportamiento de estos dos caracteres. El flag **'m'**, cuando está presente indica que el texto es multilínea y queremos que ^ y **$** seleccionen principio y final **de cada línea**, en vez de principio y final de texto.

7.

\s, \d, \D, \w, \W .... Símbolos para hacer la vida más fácil

Algunos conjuntos de caracteres son de uso tan común que se han creado unos símbolos para incluirlos más fácilmente. Por ejemplo, para indicar que un carácter debe ser un número podemos escribir **[0-9]** o

podemos utilizar **\d** (digit) que es mucho más sencillo. Cuando el símbolo aparece en mayúscula significa exactamente lo contrario. **\D** sería cualquier carácter que NO sea un número.

Los más utilizados son:

\s	Cualquier tipo de espacio en blanco (espacio, tabulador, salto de línea, etc.)
\S	Todo carácter que NO sea un espacio en blanco
\t	Tabulador
\w	Cualquier carácter alfanumérico. Equivalente a [a-zA-Z0-9_]
\W	Cualquier carácter NO alfanumérico
\d	Un dígito. Equivale a [0-9]
\D	Cualquier carácter que NO sea una dígito
\b	'Word Boundary'. Marca el inicio o el fin de una palabra (similar a '^' y '$' para una línea)

Como ejemplo, se escribe una expresión para comprobar un nombre de usuario. Se supone que el nombre de usuario no puede contener espacios y sólo puede contener números y letras (incluimos también el guion bajo o subrayado). Tiene que tener un mínimo de 6 caracteres y un máximo de 12:

```
var myRegex = /^\w{6,12}$/g;
```

Sólo seleccionará un nombre de usuario cuando cumpla las condiciones. En caso contrario no seleccionará nada. Puede utilizarse en una condición de JavaScript para aceptarlo o rechazarlo.

8.

'|' OR

La barra vertical nos permite definir varias expresiones diferentes para buscar una o otra:

```
var myRegex = /perro|gato/gi;
```

perro, loro, gato, hamster, serpiente.

Se busca la coincidencia con la expresión de la derecha o la de la izquierda **completas**. Si se desea que el OR sólo afecte a una parte de la expresión, se debe encerrar esa parte entre paréntesis. Por ejemplo, si se desea buscar 'elefante' y 'elemento'

```
var myRegex = /ele(mento|fante)/gi;
```

elegante, elemento, testamento, elefante, chanante, elemental.

También se puede definir más de dos opciones:

```
var myRegex = /ele(mento|fante|gante)/gi;
```

elegante, elemento, testamento, elefante, chanante, elemental.

9.

() Creando subexpresiones en la regex

Los paréntesis se utilizan para marcar una parte de la expresión sobre la que actúa algún operador. En el ejemplo anterior se revisó que se puede delimitar el rango sobre el que actuaba **OR** para que no se aplicara a la expresión completa. También se puede extender el rango sobre el que actúan los cuantificadores.

```
var myRegex = /1(23)+/gi;
```

En la expresión anterior el '**+**' se aplica al grupo completo '23'.

123, 1232323, 1232222, 123333333, 12323232323.

Sin los paréntesis los cuantificadores se aplican sólo al carácter inmediatamente anterior:

```
var myRegex = /123+/gi;
```

123, 1232323, 1232222, 123333333, 12323232323.

Los paréntesis tienen también otra función: **memorizan todas las coincidencias** que se encuentren en el texto y pueden utilizarse después como **$1**, **$2**, **$3**, etc. Sólo se memoriza y guarda (en **$n**) la parte de la expresión marcada con los paréntesis. Por ejemplo, en una expresión que seleccione fechas, se puede encerrar entre paréntesis la parte que coge el día y el mes y luego cambiarlo de orden colocando $2-$1. Se puede utilizar **?:** para evitar que se memorice el contenido:

```
var myRegex = /1(?:23)+/gi;
```

Ejemplo:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8" />
		<title>Validación de una expresión regular</title>
		<script type="text/javascript">
			function literal() {
				var m = document.getElementById("matricula").value;
				var er = /^[A-Z]{1,2}\s\d{4}\s([B-D]|[F-H]|[J-N]|[P-T]|[V-Z]){3}$/;
				if(er.test(m))
					alert("La matricula es correcta");
				else
					alert("La matricula NO es correcta");
			}
		</script>
	</head>
	<body>
		<form id="miFormulario" action="" method="get">
			<p>
				Matr&iacute;cula: <input type="text" id="matricula" /> <br />
				<input type="button" value="Literal" onclick="literal()" />
				<input type="button" value="Objeto" onclick="objeto()" />
			</p>
		</form>
	</body>
</html>
```

10.

Ejercicio.

Diseñar un captcha textual utilizando expresiones regulares para validar el usuario y la contraseña.