TUTORIAL SOBRE EXPRESIONES REGULARES EN JAVASCRIPT PARA VALIDACIÓN DE CAMPOS EN FORMULARIOS HTML:

Ejemplos de expresiones regulares (http://www.lsi.us.es/docencia/get.php?id=1795):

NOTA: Las expresiones regulares se encuentran delimitadas por el carácter barra (/).

– **Número de teléfono nacional (sin espacios)**
- Ejemplo: 954556817
- Exp. Reg.: /^\d{9}$/ o también /^[0-9]{9}$/

Comienza (^) por una cifra numérica (\d) de la que habrá 9 ocurrencias ({9}) y aquí acabará la cadena ($).
NOTA: La expresión "\d" equivale a la expresión "[0-9]", y representa a un carácter de una cifra numérica, es decir, '0' o'1' o '2' o '3' … o '9'.

– **Número de teléfono internacional**
- Ejemplo: (+34)954556817
- Exp. Reg.: /^\(\+\d{2,3}\)\d{9}$/

Comienza (^) por un paréntesis (\(), le sigue un carácter + (\+), después una cifra numérica (\d) de la que habrá 2 o 3 ocurrencias ({2,3}), después le sigue un paréntesis de cierre (\)), luego viene una cifra numérica de la que habrá 9 ocurrencias ({9}), y aquí acabará la cadena ($).
NOTA: Puesto que los caracteres: (,), +, *, -, \, {, }, |, etc… tienen significados especiales dentro de una expresión regular, para considerarlos como caracteres normales que debe incluir una cadena deben de ir precedidos del carácter de barra invertida \.

– **Fecha con formato DD/MM/AAAA**
- Ejemplo: 09/01/2006
- Exp. Reg.: /^\d{2}\/\d{2}\/\d{4}$/

Comienza (^) por una cifra numérica (\d) de la que habrá 2 ocurrencias ({2}), después una barra (\/), seguida de 2 cifras numéricas, otra barra, 4 cifras numéricas, y aquí acabará la cadena ($).

– **Código postal**
- Ejemplo: 41012
- Exp. Reg.: /^\d{5}$/

Únicamente contiene 5 cifras numéricas.

– **Email**
- Ejemplo: usuario@servidor.com
- Exp. Reg.: /^(.+\@.+\..+)$/

Comienza (^) por caracteres cualesquiera que no sean salto de línea (.) de los que habrá al menos una ocurrencia (+), después el carácter arroba (\@), seguido de al menos un carácter que no podrá ser el salto de línea (.+), después viene el carácter punto (\.), seguido de al menos un carácter donde ninguno podrá ser el salto de línea (.+), y aquí acabará la cadena ($).

– **Número entero**
- Ejemplo: -123
- Exp. Reg.: /^(\+|\-)?\d+$/ o también /^[+-]?\d+$/ o también /^[+-]?[0-9]+$/

Comienza (^) opcionalmente (?) por el carácter + o por el carácter -, por lo que puede que incluso no aparezcan ninguno de los 2; seguidamente vienen caracteres de cifras numéricas (\d) de los que al menos debe introducirse uno (+), y aquí acabará la cadena ($).

– **Número real**
- Ejemplo: -123.35 o 7,4 o 8
- Exp. Reg.: /^[+-]?\d+([,.]\d+)?$/

Comienza (^) opcionalmente (?) por el carácter + o por el carácter -, por lo que puede que incluso no aparezcan ninguno de los 2; seguidamente vienen caracteres de cifras numéricas (\d) de los que al menos debe introducirse uno (+), y, opcionalmente, aparecerá un punto o coma decimal seguido de al menos una cifra numérica, y aquí acabará la cadena ($).

Ejemplo de uso en un formulario:

```
<html>
<head>
<title>Ejemplo de validación de un formulario con campos tipo teléfono y tipo dni usando expresiones regulares</title>
<script>
function ValidaCampos(formulario) {
        var expresion_regular_telefono = /^\d{9}$/; // 9 cifras numéricas.
        var expresion_regular_dni = /^\d{8}[a-zA-Z]$/; // 8 cifras numéricas más un carácter alfabético.
        // Usaremos el método "test" de las expresiones regulares:
        if(expresion_regular_telefono.test(formulario.telefono.value)==false) {
                alert('Campo TELEFONO no válido.');
                return false; // sale de la función y NO envía el formulario
        }
        if(expresion_regular_dni.test(formulario.dni.value)==false) {
                alert('Campo DNI no válido.');
                return false; // sale de la función y NO envía el formulario
        }
        alert('Gracias por rellenar nuestro formulario correctamente.');
        return true; // sale de la función y SÍ envía el formulario
}
</script>
</head>

<body>
<form name="formulario" action="recoger_datos.php" onSubmit="return ValidaCampos(this)">
DNI:<input type="text" name="dni" size="9" maxlength="9"><br>
Teléfono: <input type="text" name="telefono" size="9" maxlength="9"><br>
<input type="submit" value="Enviar" name="enviar">
</form>
</body>
</html>
```

http://www.elcodigo.net/tutoriales/jsavanzado/jsavanzado5.html

1. Introducción

Las **Expresiones Regulares** son patrones que permiten buscar coincidencias con combinaciones de caracteres dentro de cadenas de texto. Estos patrones pueden utilizarse con los metodos **exec** y **text** del objeto **RegExp**, y con los metodos **match**, **replace**, **search** y **split** del objeto **String**.

Las expresiones regulares están disponibles a partir de la versión 1.2 de JavaScript (Netscape Navigator 4.x y Microsoft Internet Explorer 4.x).

2. Creación de una expresión regular

Par crear una expresión regular, puede utilizarse dos métodos:

1) La primera opción compila la expresión regular cuando se evalúa el script, por lo que **es mejor cuando la expresión regular es una constante** (**delimitada por barras**) y no va a variar a lo largo de la ejecución del programa.

exp_reg1 = /^[0-9]+/;

La variable se convierte en una variable del tipo expresión regular, por tanto, puede usarse con ella el método **test** para validar la cadena.

if(exp_reg1.**test**(“123”)==false)

2) La segunda opción compila la expresión regular en tiempo de ejecución (guardada en una variable de tipo cadena o en un campo de un formulario). Aquí **los delimitadores son las comillas dobles**, no las barras.

exp_reg2 = **new RegExp**("^[0-9]+"); // Ahora exp_reg2 es una variable que contiene una expresión regular.

exp_reg3 = **new RegExp**(formu.campo1.value);
// exp_reg3 tendrá como expresión regular el contenido del campo campo1 del formulario formu.

exp_reg4 = **new RegExp**(cadena1);
// exp_reg4 tendrá como expresión regular el contenido de la variable de cadena cadena1.

if(exp_reg3.**test**("123")==false) // Ahora podrá usarse el método test en las variables.

3. Creación de los patrones

Una expresión regular es una combinación de caracteres normales con caracteres especiales. Por ejemplo, la expresión regular **/ejemplo/** encontrará la subcadena "ejemplo" dentro de la cadena "Esto es un ejemplo.".

Con la utilización de caracteres especiales se consigue encontrar coincidencias con los retornos de carro, los tabuladores, el inicio o el final de las palabras, las repeticiones de caracteres...

La siguiente tabla muestra una lista de los caracteres especiales más importantes, así como su significado y un ejemplo de aplicación:

Carácter	Significado	Ejemplo
\	Indica que el siguiente carácter normal debe ser considerado como especial. También se utiliza como carácter de escape para los caracteres especiales.	/\n/ encuentra un salto de línea. Si se desea buscar el carácter '\', habrá que utilizar /\\/
^	Se utiliza para encontrar el comienzo de una línea.	/a/ encuentra una 'a' en cualquier lugar, pero /^a/ haya una coincidencia con el carácter 'a' si se encuentra al comienzo de una línea.
$	Se utiliza para encontrar el final de una línea.	
*	Encuentra coincidencia del carácter que le precede cuando aparece 0 o más veces en la cadena.	/hola*/ encuentra coincidencias en las cadena "hol", "hola" y "holaaaa".
+	Igual que el '*', pero cuando el carácter aparece 1 o más veces.	
?	Igual que el '*', pero cuando el carácter aparece 0 o 1 vez.	
.	Se utiliza para encontrar cualquier carácter que no sea un carácter de nueva línea.	/.sa/ encontrará coincidencias en las cadenas "casa", "cosa" y "cesa", pero no en "asa".
(x)	Encuentra coincidencias con 'x', y recuerda el patrón para su posterior utilización.	
x|y	Encuentra coincidencia si aparece el carácter 'x' o el carácter 'y'.	
{n}	Encuentra coincidencia si hay exactamente n apariciones del carácter que precede.	Por ejemplo, /a{3}/ encuentra coincidencia en "holaaa", pero no en "holaa".
{n,}	Encuentra coincidencia si hay al menos n apariciones del carácter que precede.	
{n,m}	Encuentra coincidencia si hay como mínimo n y como máximo m apariciones del carácter que precede.	
[xyz]	Representa un conjunto de caracteres individuales.	/[aeiou]/ encuentra coincidencias con cualquier vocal. Si se utiliza el carácter '-' se pueden definir rangos. Por ejemplo, /[0-3]/ encuentra coincidencias si aparecen en la cadena los caracteres '0', '1', '2' o '3'.
[^xyz]	Representa un conjunto de caracteres individuales complementario. Encuentra coincidencias con aquellos	

	caracteres que NO aparezcan en el conjunto. Al igual que en el caso anterior, con el '-' se pueden definir rangos.	
`[\b]`	Encuentra coincidencia con el carácter de retroceso.	
`\b`	Encuentra coincidencias con los límites de las palabras.	Por ejemplo, `/\bola/` encuentra la cadena "ola" en "Viene una ola", pero no en "Viene una cola".

4. Aplicación a la validación de campos de formularios

Una de las aplicaciones más habituales de las expresiones regulares es la validación de campos de formularios. Para ello, se crea una función de validación que contiene una expresión regular por cada tipo de campo que se desea validar.

Por ejemplo, podemos crear una expresión regular para campos de teléfono, que compruebe que se han introducido sólo números, espacios o el carácter '-' en el campo correspondiente.

Para validar los campos se utiliza el método **test** de la expresión regular correspondiente. Este método compara la cadena que se le pasa como argumento con el patrón de la expresión regular.

El siguiente ejemplo valida campos de teléfono y direcciones de correo electrónico:

```
function ValidaCampos(formu) {
   //expresión regular para teléfonos
   //permite campos vacíos y guiones
   var er_tlfono = /(^[0-9\s\+\-])+$/;

   //expresión regular para emails
   var er_email = /^(.+\@.+\..+)$/;

   //comprueba campo tlfono de formu
   //usa el método test de expresión regular
   if(!er_tlfono.test(formu.tlfono.value)) {
      alert('Campo TELEFONO no válido.');
      return false;    //no submit
   }

   //comprueba el campo email de formu
   //usa método test de la expresión regular
   if(!er_email.test(formu.email.value)) {
      alert('Campo E-MAIL no válido.');
      return false;    //no submit
   }

   return trae;        //pasa al submit
}
```

La función de validación se invoca utilizando el evento **onSubmit** del formulario. Cuando la validación no da positivo, la función de validación devuelve **false**. Esto cancela el *submit*, de modo que el usuario pueda corregir la entrada incorrecta. En caso contrario, se devuelve **true**. El *tag form* quedaría así:

```
<form name="formu" action="datos.php" onSubmit="return ValidaCampos(this)">
```

El gran inconveniente de este procedimiento es que sólo funciona a partir de las versiones 4.0 de los navegadores de Microsoft y Netscape.

Para comprobar expresiones regulares, puede usarse la siguiente página:

http://gollum.inforg.uniovi.es/aii/valida_regexp.php

Donde en un cuadro de texto se escribe la expresión regular, en el siguiente se escribe la cadena a validar, y al pulsar el botón “Verificar cadena de entrada” se muestra si dicha cadena cumple el patrón que especifica la expresión regular.

Según otro autor:

http://www.webintenta.com/validacion-con-expresiones-regulares-y-javascript.html

Validación con expresiones regulares y Javascript

Las expresiones regulares son modelos que describen las combinaciones de caracteres en el texto. Se podrían definir como una serie de caracteres que forman un patrón, que representan a otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otros conjuntos de caracteres para ver las coincidencias. Las expresiones regulares pueden utilizarse en múltiples lenguajes de programación pero en esta entrada vamos a ver un ejemplo de validación de formularios mediante Javascript y haciendo uso de expresiones regulares.

La tabla siguiente contiene los caracteres especiales de las expresiones regulares:

Carácter	Texto buscado
^	Principio de entrada o línea.
$	Fin de entrada o línea.
*	El carácter anterior 0 o más veces.
+	El carácter anterior 1 o más veces.
?	El carácter anterior una vez como máximo (es decir, indica que el carácter anterior es opcional).
.	Cualquier carácter individual, salvo el de salto de línea.
x|y	x o y.
{n}	Exactamente n apariciones del carácter anterior.
{n,m}	Como mínimo n y como máximo m apariciones del carácter anterior.
[abc]	Cualquiera de los caracteres entre corchetes. Especifique un rango de caracteres con un guión (por ejemplo, [a-f] es equivalente a [abcdef]).
[^abc]	Cualquier carácter que no esté entre corchetes. Especifique un rango de caracteres con un guión (por ejemplo, [^a-f] es equivalente a [^abcdef]).
\b	Límite de palabra (como un espacio o un retorno de carro).
\B	Cualquiera que no sea un límite de palabra.
\d	Cualquier carácter de dígito. Equivalente a [0-9].
\D	Cualquier carácter que no sea de dígito. Equivalente a [^0-9].
\f	Salto de página.
\n	Salto de línea.
\r	Retorno de carro.
\s	Cualquier carácter individual de espacio en blanco (espacios, tabulaciones, saltos de página o saltos de línea).
\S	Cualquier carácter individual que no sea un espacio en blanco.
\t	Tabulación.
\w	Cualquier carácter alfanumérico, incluido el de subrayado. Equivalente a [A-Za-z0-9_].
\W	Cualquier carácter que no sea alfanumérico. Equivalente a [^A-Za-z0-9_].

La tabla siguiente contiene algunos de los patrones más utilizados a la hora de validar formularios:

Letra minúscula	[a-z]
Correo electrónico	/[\w-\.]{3,}@([\w-]{2,}\.)*([\w-]{2,}\.)[\w-]{2,4}/
URL	^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$

Fecha	^\d{1,2}\/\d{1,2}\/\d{2,4}$
Hora	^(0[1-9]|1\d|2[0-3]):([0-5]\d):([0-5]\d)$
Tarjeta de crédito	^((67\d{2})|(4\d{3})|(5[1-5]\d{2})|(6011))(-?\s?\d{4}){3}|(3[4,7])\ d{2}-?\s?\d{6}-?\s?\d{5}$
Número teléfono	^[0-9]{2,3}-? ?[0-9]{6,7}$
Código postal	^([1-9]{2}|[0-9][1-9]|[1-9][0-9])[0-9]{3}$

Una página web de ejemplo llamada index.php que permite comprobar si una cadena es validada por una expresión regular de JavaScript podría ser la siguiente (si la cadena no es válida muestra un mensaje de error y no envía los datos del formulario):

```
<html>
<head>
<title>Comprobación de si una cadena cumple el patrón de una expresión regular</title>

<?
// Si recibe los campos del formulario:
if (isset($_GET['expresion']) && isset($_GET['cadena']) && isset($_GET['comprobar'])) {
	echo "</head><body>\n";
	echo "DATOS OBTENIDOS:<BR>\n";
	echo "Expresión Regular: <b>" . $_GET['expresion'] . "</b><BR>\n";
	echo "Cadena válida: <b>" . $_GET['cadena'] . "</b><BR>\n";
	echo "<br><a href='index.php'>Volver</a>\n";
	echo "</body></html>";
}
// Si no recibe los campos del formulario:
else{
?>

<script>
function valida(formulario) {
	expresion_regular = new RegExp(formulario.expresion.value);
	// Usamos el método "test" de la expresión regular
	if(expresion_regular.test(formulario.cadena.value)==false) {
		alert('La cadena no cumple el patrón de la expresión regular.');
		return false; //no envía el formulario
	}
	return true; // Envía el formulario
}
</script>
</head>

<body>
Introduzca una expresión regular (por ejemplo: <b>^[0-9]{9}$</b>) y una cadena
(por ejemplo: <b>123456789</b>), para comprobar si dicha cadena respeta el patrón
que especifica la expresión regular (es decir, si la expresión regular valida dicha cadena).<br>
<form name="formulario" action="index.php" method="get"
	onSubmit="return valida(formulario)">
Expresión Regular:<input type="text" name="expresion" size="20"><br>
Cadena a Comprobar: <input type="text" name="cadena" size="20"><br>
<input type="submit" value="Comprobar" name="comprobar">
</form>
</body>
</html>

<?
} //else (Si no recibía los campos del formulario)
?>
```