EXPRESIONES REGULARES - REGEXP

http://www.w3schools.com/jsref/jsref_obj_regexp.asp

https://www.youtube.com/watch?v=MbUyasEUOJI

Las expresiones regulares son la descripción formal de un ***patrón de búsqueda de cadenas.***

Cuando necesitamos realizar una tarea de búsqueda o reemplazo de cadenas, usando expresiones regulares podemos ahorrar mucho tiempo.

Son muchos los lenguajes de programación que incluyen la posibilidad de utilizar expresiones regulares. JavaScript permite implementar expresiones regulares y así facilitar las comprobaciones de ciertos datos que deben seguir una estructura concreta.

Resultan especialmente útiles a la hora de **validar** algunos tipos de datos en los formularios así como al realizar **búsquedas** a partir de unos criterios especificados.

La sintaxis más usada en javascript para definir un patrón es:

```
var re =/patrón/;
```

Otra forma que nosotros vamos a usar menos es:

```
var re = new RegExp("patrón");
```

Por ejemplo, un patrón que nos sirva para identificar un código postal de Madrid podría ser así:

```
var re=/28[0-9]{3}/;   ( ó var re=new RegExp(28[0-9]{3})   )
```

Además del patrón se puede especificar si la búsqueda ha de ser global (es decir, no pare al encontrar la primera coincidencia) y si diferencia entre mayúsculas y minúsculas o no.

var re =/patrón/g;	global
var re =/patrón/i;	mayúsculas y minúsculas
var re =/patrón/gi;	global y mayúsculas y minúsculas

Un patrón se utiliza fundamentalmente para hacer tres operaciones: validar, buscar, y reemplazar.

Validar con un patrón: Método test()

Sintaxis: regexp.test(*string*)

Recibe una cadena string y devuelve true o false dependiendo de si la cadena se ajusta a la expresión regular regexp

```
var re1 = /recreo/;
var texto1 = 'me gusta el recreo';
var texto2 ='nos vamos en 5 minutos';
alert (re1.test(texto1)); // true
alert (re1.test(texto2)); // false
```

Buscar coincidencias con un patrón: Método match()

Sintaxis: string.match(*regexp*)

Recibe una expresión regular y devuelve las subcadenas que concuerden con esa expresión (en un array). Si no encuentra ninguna devuelve null.

```
texto = 'Como Me Molan Las Patatas';

esta=texto.match(/[lpt]a/g);
alert (esta);                    // la,ta,ta

alert (texto.match(/pan/)
alert(esta);                    // null

esta=texto.match(/[lpt]a/i);
alert (esta);                    // la

esta=texto.match(/[lpt]a/gi);
alert (esta);                    // la,La,Pa,ta,ta
```

Buscar posición con un patron: Método search()

Sintaxis: string.search(*regexp*)

Recibe una expresión regular y devuelve la posición en la cadena de la primera coincidencia, devuelve -1 si no hay ninguna.

```
texto = 'Como Me Molan Las Patatas';
var re=/[MPT]/;
alert (texto.search(re));  // 5
var re=/[MPT]/i;
alert (texto.search(re));  // 2
var re=/[xyz]/i;
alert (texto.search(re));  //  -1
```

Reemplazar usando un patrón: Método replace()

Sintaxis: string.replace(*regexp, valor*)

Recibe una expresión regular y devuelve la cadena reemplazando la primera coincidencia por valor (si no se ha especificado /g para que la búsqueda sea global) o todas las coincidencias las sustituye por valor (si sí se ha especificado)

```
texto = 'Como Me Molan Las Patatas';
```

```
var re=/[MPT]/;
alert (texto.replace(re,'*'));  // 'Como *e Molan Las Patatas'
var re=/[MPT]/gi;
alert (texto.replace(re,'*'));   // 'Co*o *e *olan Las *a*a*as'
```

Construir expresiones regulares

Ejemplos de expresiones regulares para comprobar:

- Si una cadena contiene alguno de los caracteres especificados en un conjunto, por ejemplo para comprobar si una cadena contiene vocales

  var re=/[aeiou]/gi;

  (Obs: no tendrá en cuenta vocales acentuadas)

  var re=/[aeiouáéíóú]/gi;

- Si una cadena contiene alguno de los caracteres NO especificados en un conjunto

  var re=/[^aeiou]/gi;

- Si una cadena contiene alguno de los caracteres especificados en un intervalo de caracteres
  - Si una cadena contiene dígitos

    var re=/[0-9]/g;
  - Si tiene letras

    var re=/[a-z]/gi;
  - Si tiene letras o dígitos

    var re=/[a-z0-9]/gi;
  - Si tiene letras de la a ‘a’ la ‘d’ o dígitos o de la ‘x’ a la ‘z’

    var re=/[a-d0-9x-z]/gi;
- Si una cadena contiene alguno de los caracteres NO especificados en un intervalo de caracteres

  var re=/[^a-z]/gi;
- Si tiene alguno de las subcadenas especificados

  var re=/(cad1|cad2|…|cadn)/

Cuantificadores

Cuantificador	Descripción
n *	Cualquier cadena que contiene cero o más apariciones de n. *Busca el carácter precedente 0 (cero) o más veces.* *Por ejemplo, la expresión /bo*/ encontrará la subcadena 'boooo' en la cadena "A ghost booooed" y el caracter 'b' en la cadena "A bird warbled", pero no encontrará nada en la cadena "A goat grunted".*
n +	Cualquier cadena que contiene al menos un n
n?	Cualquier cadena que contiene cero o una ocurrencias de n
n {X}	Cualquier cadena que contiene una secuencia de *X n* 's
n {X, Y}	Cualquier cadena que contiene una secuencia de X a Y veces de *n* 's
n {X,}	Cualquier cadena que contiene una secuencia de al menos X *n*'s
n $	Cualquier cadena con n al final de la misma
^ n	Cualquier cadena con n al comienzo de la misma
x (? = n)	Cualquier cadena x que venga seguida por una cadena específica n y devuelve la subcadena x /pa(?=ta)/ Vale patata y devuelve la subcadena pa
x(?! n)	Cualquier cadena x que no es seguida por una cadena específica n /pa(?!ta)/ No vale patata

^A	Comience por A	/^A/	Valen Amar, Andar, A, no vale amar
A$	Termine por A	/A$/	Valen A,CANTA, no vale canta
		/^A/i	Valen amar
		/A$/i	Valen canta
*	0 o más veces	/^1*234/	Valen 234, 1234, 11234...
+	1 o más veces	/^1+234/	Valen 1234, 11234...
?	1 o 0 veces	/^1?234/	Valen 234, 1234, No vale 11234...
{n}	n veces	/^1{2}234/	Valen **11**234, no vale **11**1234...
{n,}	Al menos n veces	/^1{2,}234/	Valen **11**234, **111**234...

{m,n} entre m y n veces /^1{2,3}234/ Valen **11**234, **111**234...

/^[0-9]{3}/ Cualquier cadena que comience por tres dígitos, vale 123patata, 1234patata

No valen 12patata

/(abc[01]){2}/ Vale 'abc0abc1', '***abc0abc1abc0***'

No vale 'abc0****abc1'

Metacaracteres

Metacarácter	Descripción
.	Cualquier carácter individual, excepto una línea nueva o final de línea
\w	Cualquier carácter alfanumérico o _. Equivale a [a-zA-Z0-9_]
\W	Cualquier carácter no alfanumérico
\d	Cualquier dígito. Equivale a [0-9]
\D	Cualquier carácter que no sea dígito.
\s	Espacio en blanco
\S	Carácter que no sea blanco
\b	Fin de palabra o retorno de carro
\B	Cualquier carácter que no sea límite de palabra
\n	Salto de linea
\ f	Salto de página
\ r	Retorno de carro
\ t	Tabulación

Ejemplos:

Cadena que empieza y termina por A /^A.*A$/

Cadena que tiene al menos tres A seguidas o no /.*A.*A.*A.*/

Cadena que tiene una A en la segunda posición /^.A.*/

Palabra que tiene una A en la segunda posición /^[a-z]a[a-z]*$/

Ejemplo de validación de un mail con una expresión regular:

```
<html >
<head>
<title></title>
<script type="text/javascript">
function validarMail() {
    var RegEx =/[A-Z0-9._%+-]+@[A-Z0-9-]+.+.[A-Z]{2,4}/igm;
    var dato=document.getElementById('mail').value;
    if (RegEx.test(dato)) {
        alert('Mail correcto');
    } else {
        alert('Mail incorrecto');
        document.form.pass.focus();
    }
}
</script>
</head>
<body>
<form name="form">
<p>Introduce un mail: <input type="text" id="mail">
  <input name="button" type="button" value="Probar" onclick="validarMail(
);">  <br>
  </form>
</body>
</html>
```