C.F.G.S.: DESARROLLO DE APLICACIONES WEB
Módulo: DESARROLLO WEB EN ENTORNO CLIENTE

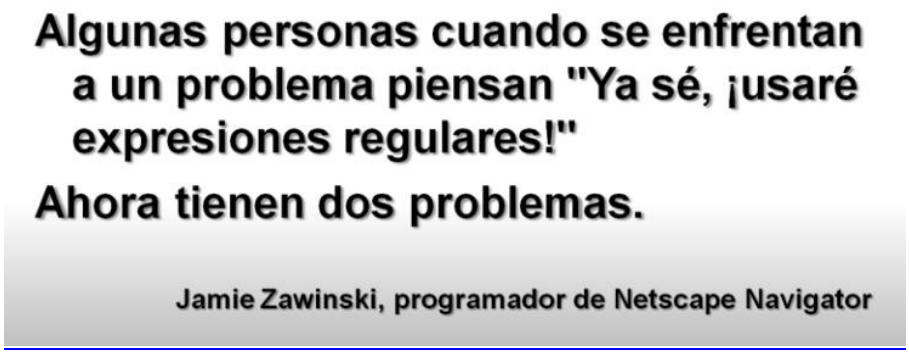
05 EXPRESIONES REGULARES

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

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

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

https://www.youtube.com/watch?v=Xf4wvsnWNK0



¿Qué es una expresión regular?

Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en cadenas.

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

Indican todas las particularidades y reglas que debe cumplir un patrón y generan dinámicamente en memoria el código necesario para buscar y manipular ese patrón descrito.

Cuando necesitamos realizar una tarea de búsqueda o reemplazo de cadenas, si nos ceñimos a los métodos convencionales, debemos proporcionar siempre una cadena exacta que se ha de encontrar. Cuando se requiere mayor flexibilidad podemos hacer uso de expresiones regulares.

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.

```
<html >
<head>
<title></title>
<script type="text/javascript">
function validarPass(campo) {
   var RegEx = /(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$/;
   if (document.form.pass.value.match(RegEx)) {
      alert('Password Correcta');
   } else {
      alert('Password Incorrecta');
      document.form.pass.focus();
   }
}
</script>
</head>
<body>
<form name="form">
<p><input type="text" name="pass" onblur="validarPass();">
 <input name="button" type="button" value="Probar" onclick="validarPass();"> <br>
 Entre 8 y 10 caracteres alfanuméricos, por lo menos un digito y una letra, y no  puede
contener caracteres especiales
</form>
</body>
</html>
```

En JavaScript una expresión regular es un objeto

Objeto RegExp

En JavaScript para definir una expresión regular podemos usar el constructor del objeto RegExp (de Regular Expresion). Será un objeto con sus métodos y propiedades.

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

O utilizar una sintaxis especialmente pensado para ello, que es lo más habitual:

```
let re =/patrón/;
```

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.

```
let re =/patrón/g;        global
let re =/patrón/i;        mayúsculas y minúsculas
let re =/patrón/gi;       global y mayúsculas y minúsculas
let re = new RegExp("patrón",gi);
```

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

```
let re1 = new RegExp ('mola');
let re2 = /patata/;
let texto = 'como me molan las patatas';
alert (re2.test(texto)); // true
alert (re1.test(texto)); // true
```

Métodos sobre string relacionados con expresiones regulares:

Método match()

Sintaxis: string.match(*regexp*)

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

```
alert (texto.match(re2))        // patata
alert (texto.match(/pan/))      // null
```

```
texto = 'Como Me Molan Las Patatas';
let re3=/[lpt]a/g;
esta=texto.match(re3);
alert (esta);                   // la,ta,ta
re3=/[lpt]a/i;
esta=texto.match(re3);
alert (esta);                   // la
re3=/[lpt]a/gi;
esta=texto.match(re3);
alert (esta);                   // la,La,Pa,ta,ta
```

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';
let re1=/[MPT]/;
alert (texto.search(re));  // 5
let re2=/[MPT]/i;
alert (texto.search(re2));  // 2
let re3=/[xyz]/i;
alert (texto.search(re3));  //  -1
```

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';
let re1=/[MPT]/;
alert (texto.replace(re1,'*'));  // 'Como *e Molan Las Patatas'
let re2=/[MPT]/gi;
alert (texto.replace(re2,'*'));   // '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

  let re=/[aeiou]/gi;

  (Obs: no tendrá en cuenta vocales acentuadas)

  let re=/[aeiouáéíóú]/gi;

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

  let 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 *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, no vale paca
x(?! n)	Cualquier cadena x que no es seguida por una cadena específica n /pa(?!ta)/ No vale patata, si vale paca

^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 línea
\ f	Salto de página
\ r	Retorno de carro
\ t	Tabulación

Cuidado!!

Escapando

Si necesitas usar literalmente cualquiera de los caracteres especiales (en realidad buscando un `"*"`, por ejemplo), lo debes escapar colocando una barra invertida delante de él. Por ejemplo, para buscar `"a"` seguido de `"*"` seguido de `"b"`, usarías `/a\*b/` — la barra invertida "escapa" de `"*"`, volviéndola literal en lugar de especial.

De manera similar, si estás escribiendo un literal de expresión regular y necesitas buscar una barra inclinada ("/"), la debes escapar (de lo contrario, esta termina el patrón). Por ejemplo, para buscar la cadena "/ejemplo/" seguida de uno o más caracteres alfabéticos, usarías `/\/ejemplo\/[a-z]+/i`: las barras invertidas antes de cada barra, las hace literales.

Para hacer coincidir una barra invertida literal, debes escapar de la barra invertida. Por ejemplo, para encontrar la cadena "C:\" donde "C" puede ser cualquier letra, usarías `/[A-Z]:\\/` — la primera barra invertida escapa a la que sigue, por lo que la expresión busca una sola barra invertida literal.

Si usas el constructor `RegExp` con un literal de cadena, recuerda que la barra invertida es un escape en los literales de cadena, por lo que para usarlo en la expresión regular, debes escapar en el nivel del literal de cadena. `/a\*b/` y `new RegExp("a\\*b")` crean la misma expresión, que busca "a" seguida de un "*" literal seguido de "b".

Ejemplos:

Cadena que empieza y termina por A	/^A.*A$/
Cadena que tiene al menos tres A seguidas o no	/.*A.*A.*A.*/
Cadena que tiene una A en la segunda posición	/^.A.*/
Palabra que tiene una A en la segunda posición	/^[a-z]a[a-z]*$/

TODITO TODO en:

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

Para probar expresiones regulares:

https://extendsclass.com/regex-tester.html