

# OBJETO REGEXP

Este objeto nos permite crear patrones de caracteres

## SINTAXIS DEL OBJETO RegExp:

/patrón/modificadores

Ejemplo:

/hola/g

## Constructor:

```
var patt = new RegExp("Hello World", "g");
```


## Los objetos de tipo RegExp tienen los siguientes modificadores:

**i** --> ignora mayúsculas y minúsculas en la búsqueda del patrón

**g** --> Busca todas las apariciones del patrón en lugar de parar en la primera coincidencia

**m** --> Para realizar una búsqueda multilínea

```
var str = "\nIs th\nis it?";  
var patt1 = /^is/m;
```



Is th  
is it?

## Patrones - Encontrar un rango de caracteres

**[abc]** --> Encuentra cualquiera de los caracteres escritos entre []

**[^abc]** --> No encuentra cualquiera de los caracteres escritos entre []

**[0-9]** --> Encuentra cualquiera de los dígitos escritos entre []

**[^0-9]** --> No encuentra cualquiera de los dígitos escritos entre []

**(x|y)** --> Encuentra cualquiera de las alternativas especificadas entre ()

```
var str = "re, green, red, green, gren, gr, blue, yellow";  
var patt1 = /(red|green)/g;
```



re, green, red, green, gren, gr, blue, yellow

## Patrones - Metacaracteres

Metacharacter	Description
<code>.</code>	Find a single character, except newline or line terminator
<code>\w</code>	Find a word character [A-Za-z0-9_]
<code>\W</code>	Find a non-word character [^A-Za-z0-9_]
<code>\d</code>	Find a digit
<code>\D</code>	Find a non-digit character
<code>\s</code>	Find a whitespace character
<code>\S</code>	Find a non-whitespace character
<code>\b</code>	Find a match at the beginning/end of a word
<code>\B</code>	Find a match not at the beginning/end of a word
<code>\0</code>	Find a NUL character
<code>\n</code>	Find a new line character
<code>\f</code>	Find a form feed character
<code>\r</code>	Find a carriage return character
<code>\t</code>	Find a tab character
<code>\v</code>	Find a vertical tab character
<code>\xxx</code>	Find the character specified by an octal number xxx
<code>\xdd</code>	Find the character specified by a hexadecimal number dd
<code>\uxxxx</code>	Find the Unicode character specified by a hexadecimal number xxxx

```
var str = "That's hot!";  
var patt1 = /h.t/g;
```



hat,hot

That's hot!

```
var str = "Hola mundo";  
var patt1 = /\bmu/g;  
var result = str.match(patt1);
```



mu

```
var str = "Hola mundo mundoho ho";  
var patt1 = /\bho/gi;  
var result = str.match(patt1);
```



Ho,ho

Hola mundo mundoho ho

```
var str = "Hola mundos";  
var patt1 = /\Bundo/g;  
var result = str.match(patt1);
```



undo

## Patrones - Cuantificadores

Quantifier	Description
<u><code>n+</code></u>	Matches any string that contains at least one <i>n</i>
<u><code>n*</code></u>	Matches any string that contains zero or more occurrences of <i>n</i>
<u><code>n?</code></u>	Matches any string that contains zero or one occurrences of <i>n</i>
<u><code>n{X}</code></u>	Matches any string that contains a sequence of <i>X</i> <i>n</i> 's
<u><code>n{X,Y}</code></u>	Matches any string that contains a sequence of <i>X</i> to <i>Y</i> <i>n</i> 's
<u><code>n{X,}</code></u>	Matches any string that contains a sequence of at least <i>X</i> <i>n</i> 's
<u><code>n\$</code></u>	Matches any string with <i>n</i> at the end of it
<u><code>^n</code></u>	Matches any string with <i>n</i> at the beginning of it
<u><code>?=n</code></u>	Matches any string that is followed by a specific string <i>n</i>
<u><code>?!n</code></u>	Matches any string that is not followed by a specific string <i>n</i>

### Ejemplos:

```
var str = "Is this all there is";  
var patt1 = /is(=? all)/g;
```



Is **this** all there is

```
var str = "Is this all there is";  
var patt1 = /is(?! all)/gi;
```



**Is** this all there **is**

## Grupos anónimos


Los grupos anónimos se establecen cada vez que se encierra una expresión regular en paréntesis, por lo que la expresión “/([a-zA-Z]\w\*)/” define un grupo anónimo. El motor de búsqueda almacenará una referencia al grupo anónimo que corresponda a la expresión encerrada entre los paréntesis.

```
var re = /(\w+)\s(\w+)/;  
var str = "John Smith";  
var newstr = str.replace(re, "$2, $1");  
console.log(newstr); // "Smith, John"
```

## Los objetos de tipo RegExp tienen la siguientes propiedades:

**global;** Devuelve true o false en función de si se ha utilizado o no el modificador /g en el patrón.

```
var patt1 = /W3S/g;  
var res = patt1.global;
```



```
true
```

**ignoreCase;** Devuelve true o false en función de si se ha utilizado o no el modificador /i en el patrón.

**lastIndex;** Está propiedad funciona si se ha utilizado /g en el patrón. Devuelve la posición siguiente a la última ocurrencia encontrada por los métodos exec() o test(). En caso de no encontrar el patrón devuelve 0.

```
var str = "The rain in Spain stays mainly in the plain";  
var patt1 = /ain/g;  
  
while (patt1.test(str)==true)  
{  
  document.write("'ain' found. Index now at: "+patt1.lastIndex);  
  document.write("<br>");  
}
```



```
'ain' found. Index now at: 8  
'ain' found. Index now at: 17  
'ain' found. Index now at: 28  
'ain' found. Index now at: 43
```

**multiline;** Devuelve true o false en función de si se ha utilizado o no el modificador /m en el patrón.

**source;** Devuelve el texto de un patrón.

```
var patt1 = /W3S/g;  
var res = "The text of the RegExp is: " + patt1.source;
```




```
The text of the RegExp is: W3S
```

**\$1..\$9:** Los valores de las propiedades \$1...\$9 se modifican cada vez que se obtiene una coincidencia correcta entre paréntesis. En un patrón de expresión regular se puede especificar cualquier número de subcadenas entre paréntesis, pero solo se pueden almacenar las nueve más recientes.

## Los objetos de tipo RegExp tienen todos los siguientes métodos:


**exec():** Busca un patrón en una cadena. Devuelve la primera aparición del patrón buscado.

```
var str = "The best things in life are free";  
var patt = new RegExp("e");  
var res = patt.exec(str);
```

 e


**test():** Busca un patrón en una cadena. Devuelve true o false.

```
var str = "The best things in life are free";  
var patt = new RegExp("e");  
var res = patt.test(str);
```

 true

**toString():** Devuelve una cadena de una expresión regular

```
var patt = new RegExp("Hello World", "g");  
var res = patt.toString();
```

 /Hello World/g

## Otros métodos que trabajan con expresiones regulares

String.**replace**(valor\_buscado(string o patrón), nuevo\_valor);

String.**search**( valor\_buscado(string o patrón));

String.**match**(patrón);

String.**split**(separador(cadena o patrón),límite)

**NOTA:** si se quiere utilizar metacaracteres o escapar algún carácter dentro de new RegExp() hay que utilizar **\\**. Ej: \\s en lugar de \s

## Ejemplos

### – 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}\V\d{2}\V\d{4}$/`

Comienza (^) por una cifra numérica (\d) de la que habrá 2 ocurrencias ({2}), después una barra (\V), 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 (\$).

```
<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>
```

```

var newLine = "<br />";

var re = /(\w+)@(\w+)\.(\w+)/g
var src = "Please send mail to george@contoso.com and someone@example.com. Thanks!"

var result;
var s = "";

// Get the first match.
result = re.exec(src);

while (result != null) {
    // Show the entire match.
    s += newLine;

    // Show the match and submatches from the RegExp global object.
    s += "RegExp.lastMatch: " + RegExp.lastMatch + newLine;
    s += "RegExp.$1: " + RegExp.$1 + newLine;
    s += "RegExp.$2: " + RegExp.$2 + newLine;
    s += "RegExp.$3: " + RegExp.$3 + newLine;

    // Show the match and submatches from the array that is returned
    // by the exec method.
    for (var index = 0; index < result.length; index++) {
        s += index + ": ";
        s += result[index];
        s += newLine;
    }

    // Get the next match.
    result = re.exec(src);
}

document.write(s);

// Output:
// RegExp.lastMatch: george@contoso.com
// RegExp.$1: george
// RegExp.$2: contoso
// RegExp.$3: com
// 0: george@contoso.com
// 1: george
// 2: contoso
// 3: com

// RegExp.lastMatch: someone@example.com
// RegExp.$1: someone
// RegExp.$2: example
// RegExp.$3: com
// 0: someone@example.com
// 1: someone
// 2: example
// 3: com

```