Programador JavaScript Avanzado

Unidad 2: Eventos y Formularios

Contactos: tecnologia@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 2: Eventos y Formularios

Objetivos

Que el alumno logre:

- Conocer e implementar Eventos y Callbacks

Expresiones regulares

Las expresiones regulares son patrones que proporcionan una forma de buscar y reemplazar texto.

En JavaScript, están disponibles a través del objeto RegExp, además de integrarse en métodos de cadenas.

Una expresión regular (también “regexp”, o simplemente “reg”) consiste en un patrón.

Hay dos sintaxis que se pueden usar para crear un objeto de expresión regular.

La sintaxis “larga”:

```
regexp = new RegExp("patrón", "banderas");
```

Y el “corto”, usando barras "/":

```
regexp = /pattern/; // sin banderas
regexp = /pattern/gmi; // con banderas g,m e i (para ser cubierto
pronto)
```

Las barras /.../ le dicen a JavaScript que estamos creando una expresión regular. Juegan el mismo papel que las comillas para las cadenas.

En ambos casos, **regexp** se convierte en una instancia de la clase incorporada **RegExp**.

La principal diferencia entre estas dos sintaxis es que el patrón que utiliza barras /.../ no permite que se inserten expresiones (como los literales de plantilla de cadena con ${...}). Son completamente estáticos.

Las barras se utilizan cuando conocemos la expresión regular en el momento de escribir el código, y esa es la situación más común. Mientras que new **RegExp**, se usa con mayor frecuencia cuando necesitamos crear una expresión regular “sobre la marcha” a partir de una cadena generada dinámicamente.

Flags

Las expresiones regulares pueden usar **flags** (banderas) que afectan la búsqueda.

i

Usando **i** la búsqueda no distingue entre mayúsculas y minúsculas: no hay diferencia entre A y a.

g

La búsqueda encuentra todas las coincidencias, sin ella, solo se devuelve la primera coincidencia.

m

Modo multilínea. Solo afecta el comportamiento de ^ y $.

En el modo multilínea, coinciden no solo al principio y al final de la cadena, sino también al inicio/final de la línea.

s

Habilita el modo “dotall”, que permite que un punto . coincida con el carácter de línea nueva \n

u

Permite el soporte completo de Unicode. La bandera permite el procesamiento correcto de pares sustitutos.

y

Modo “adhesivo”: búsqueda en la posición exacta del texto

Caracteres especiales o Clases de caracteres

Una clase de caracteres es una notación especial que coincide con cualquier símbolo de un determinado conjunto.

\d (“d” es de dígito")

Un dígito: es un caracter de 0 a 9.

\s (“s” es un espacio)

Un símbolo de espacio: incluye espacios, tabulaciones \t, líneas nuevas \n y algunos otros caracteres raros, como \v, \f y \r.

\w (“w” es carácter de palabra)

Un carácter de palabra es: una letra del alfabeto latino o un dígito o un guión bajo _. Las letras no latinas (como el cirílico o el hindi) no pertenecen al \w.

Por ejemplo, \d\s\w significa un “dígito” seguido de un “carácter de espacio” seguido de un “carácter de palabra”, como **1 a**.

Para cada clase de caracteres existe una "**clase inversa**", denotada con la misma letra, pero en mayúscula.

El "inverso" significa que coincide con todos los demás caracteres, por ejemplo:

\D

Sin dígitos: cualquier carácter excepto \d, por ejemplo, una letra.

\S

Sin espacio: cualquier carácter excepto \s, por ejemplo, una letra.

\W

Sin carácter de palabra: cualquier cosa menos \w, por ejemplo, una letra no latina o un espacio.

Cuantificador {n}

Digamos que tenemos una cadena como +54 (011) 15-5566-7777 y queremos encontrar todos los números en ella. Pero contrastando el ejemplo anterior, no estamos interesados en un solo dígito, sino en números completos: 54, 011, 15, 5566, 7777.

Un número es una secuencia de 1 o más dígitos \d. Para marcar cuántos necesitamos, podemos agregar un cuantificador.

Cantidad {n}

El cuantificador más simple es un número entre llaves: {n}.

Se agrega un cuantificador a un carácter (o a una clase de caracteres, o a un conjunto [...], etc) y especifica cuántos necesitamos.

Tiene algunas formas avanzadas, veamos los ejemplos:

El recuento exacto: {5}

\d{5} Denota exactamente 5 dígitos, igual que \d\d\d\d\d.

El siguiente ejemplo busca un número de 5 dígitos:

```
alert( "Tengo 12345 años de edad".match(/\d{5}/) );  //  "12345"
```

Podemos agregar \b para excluir números largos: \b\d{5}\b.

El rango: {3,5}, coincide 3-5 veces

Para encontrar números de 3 a 5 dígitos, podemos poner los límites en llaves: \d{3,5}

```
alert( "No tengo 12, sino 1234 años de edad".match(/\d{3,5}/) ); //
"1234"
```

Podemos omitir el límite superior

Luego, una regexp \d{3,} busca secuencias de dígitos de longitud 3 o más:

```
alert( "No tengo 12, sino, 345678 años de edad".match(/\d{3,}/) );
// "345678"
```

Anclas: inicio ^ y final $ de cadena

Los patrones caret ^ y pesos $ tienen un significado especial en una expresión regular. Se llaman "anclas".

El patrón caret ^ coincide con el principio del texto y pesos $ con el final.

Por ejemplo, probemos si el texto comienza con Juan:

```
let cadena = "Juan trabaja en la empresa";
alert( /^Juan/.test(cadena) ); // true
```

El patrón **^Juan** significa: "inicio de cadena y luego Juan".

Similar a esto, podemos probar si la cadena termina con nieve usando **empresa$**:

```
let cadena = "Juan trabaja en la empresa";
alert( /Juan$/.test(cadena) ); // true
```

En estos casos particulares, en su lugar podríamos usar métodos de cadena **beginWith/endsWith**. Las expresiones regulares deben usarse para pruebas más complejas.

Ambos anclajes ^...$ se usan juntos a menudo para probar si una cadena coincide completamente con el patrón. Por ejemplo, para verificar si la entrada del usuario está en el formato correcto.

Verifiquemos si una cadena esta o no en formato de hora 12:34. Es decir: dos dígitos, luego dos puntos y luego otros dos dígitos.

En el idioma de las expresiones regulares eso es \d\d:\d\d:

```
let horaCorrecta = "12:34";
let horaIncorrecta = "12:345";

let regexp = /^\d\d:\d\d$/;
```

```
alert( regexp.test(horaCorrecta) ); // true
alert( regexp.test(horaIncorrecta) ); // false
```

La coincidencia para \d\d:\d\d debe comenzar exactamente después del inicio de texto^, y seguido inmediatamente, el final $.

Toda la cadena debe estar exactamente en este formato. Si hay alguna desviación o un carácter adicional, el resultado es falso.

Límite de palabra: \b

Un límite de palabra **\b** es una prueba, al igual que **^** y **$**.

Cuando el motor regex (módulo de programa que implementa la búsqueda de expresiones regulares) se encuentra con \b, comprueba que la posición en la cadena es un límite de palabra.

Hay tres posiciones diferentes que califican como límites de palabras:

- Al comienzo de la cadena, si el primer carácter de cadena es un carácter de palabra \w.
- Entre dos caracteres en la cadena, donde uno es un carácter de palabra \w y el otro no.
- Al final de la cadena, si el último carácter de la cadena es un carácter de palabra \w.

Por ejemplo, la expresión regular **\bJava\b** se encontrará en **Hola, Java!**, donde **Java** es una palabra independiente, pero no en **Hola, JavaScript!**.

```
alert( "Hola Java!".match(/\bJava\b/) ); // Java
alert( "Hola JavaScript!".match(/\bJava\b/) ); // null
```

Expresiones regulares y formularios

Una de las implementaciones de las Expresiones regulares de Javascript es en formularios de HTML.

Veamos el siguiente ejemplo:

```
<form>
    <input type="text">
    <button>Enviar</button>
</form>
```

Y en el script:

```
let formulario = document.querySelector('form')
let campo = document.querySelector('input')
```

```
formulario.addEventListener("submit", function(e){
    e.preventDefault()
    let valor = campo.value
})
```

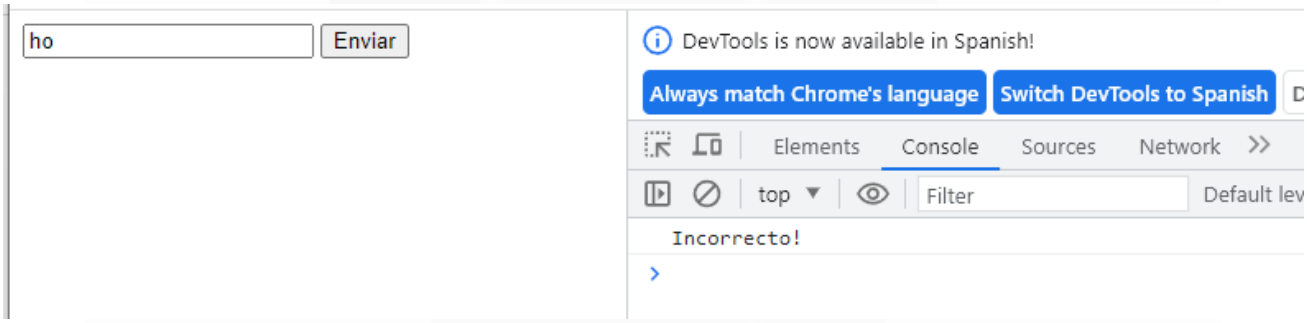
Creamos una validación con una expresión regular sencilla, en donde verifiquemos que el dato sea alfanumérico entre 5 y 10 caracteres:

```
let regexp = /^\w{5,10}$/
```

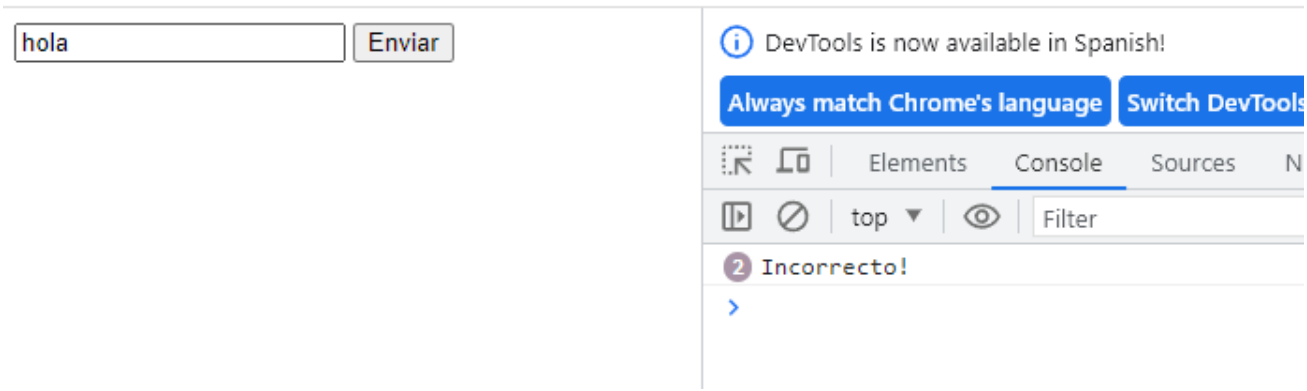
Para asociar esto a nuestro formulario, vamos a usar el método test()

```
if(regexp.test(valor)) {
        console.log('Correcto!')
    } else {
        console.log('Incorrecto!')
    }
```

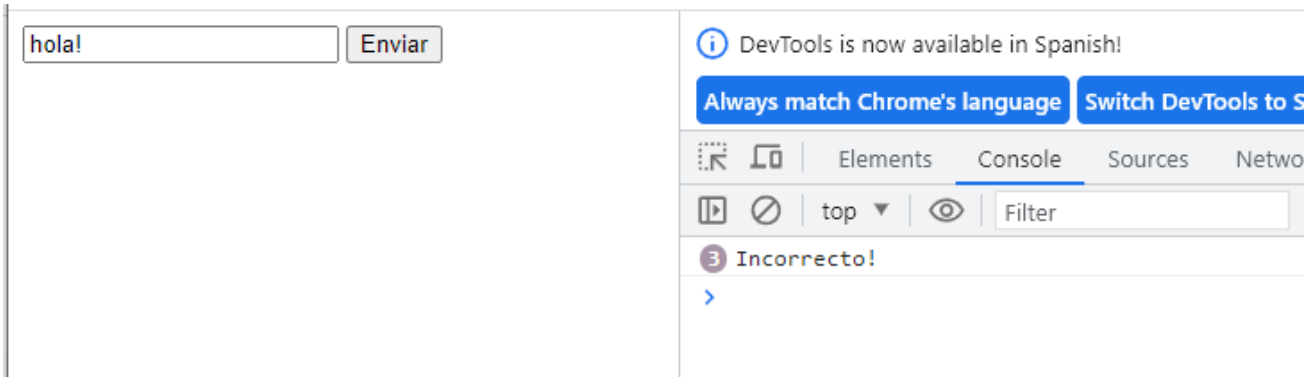
Si colocamos menos de 5 caracteres no va a marcar que es incorrecto:



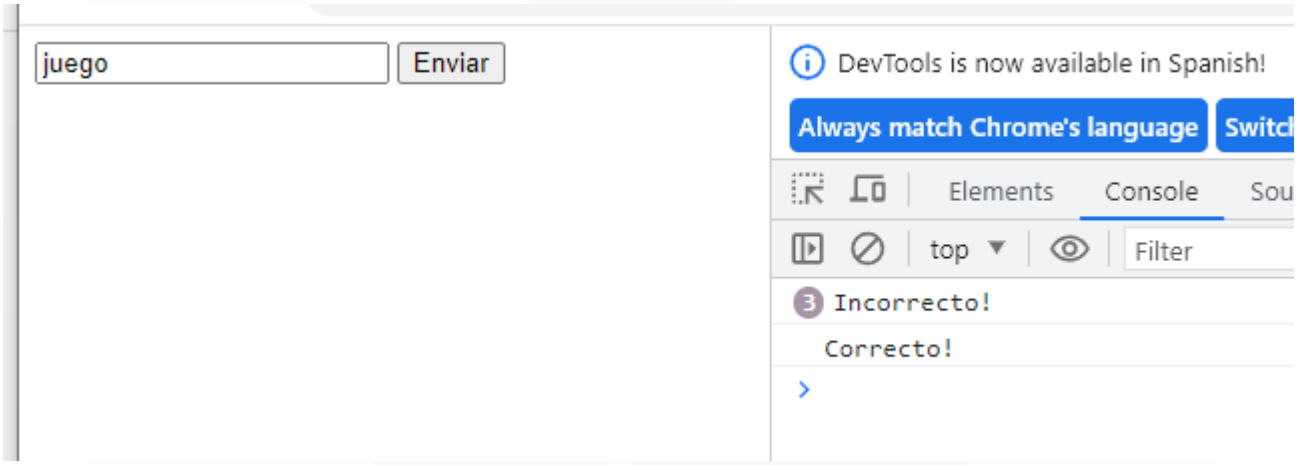
Lo mismo si usamos 4 caracteres:



O si utilizamos algún carácter especial o espacio:



Solo valida con 5 caracteres alfanuméricos:



Resumen

En esta Unidad…

Trabajamos con Formularios y Eventos

En la próxima Unidad…

Trabajaremos con AJAX