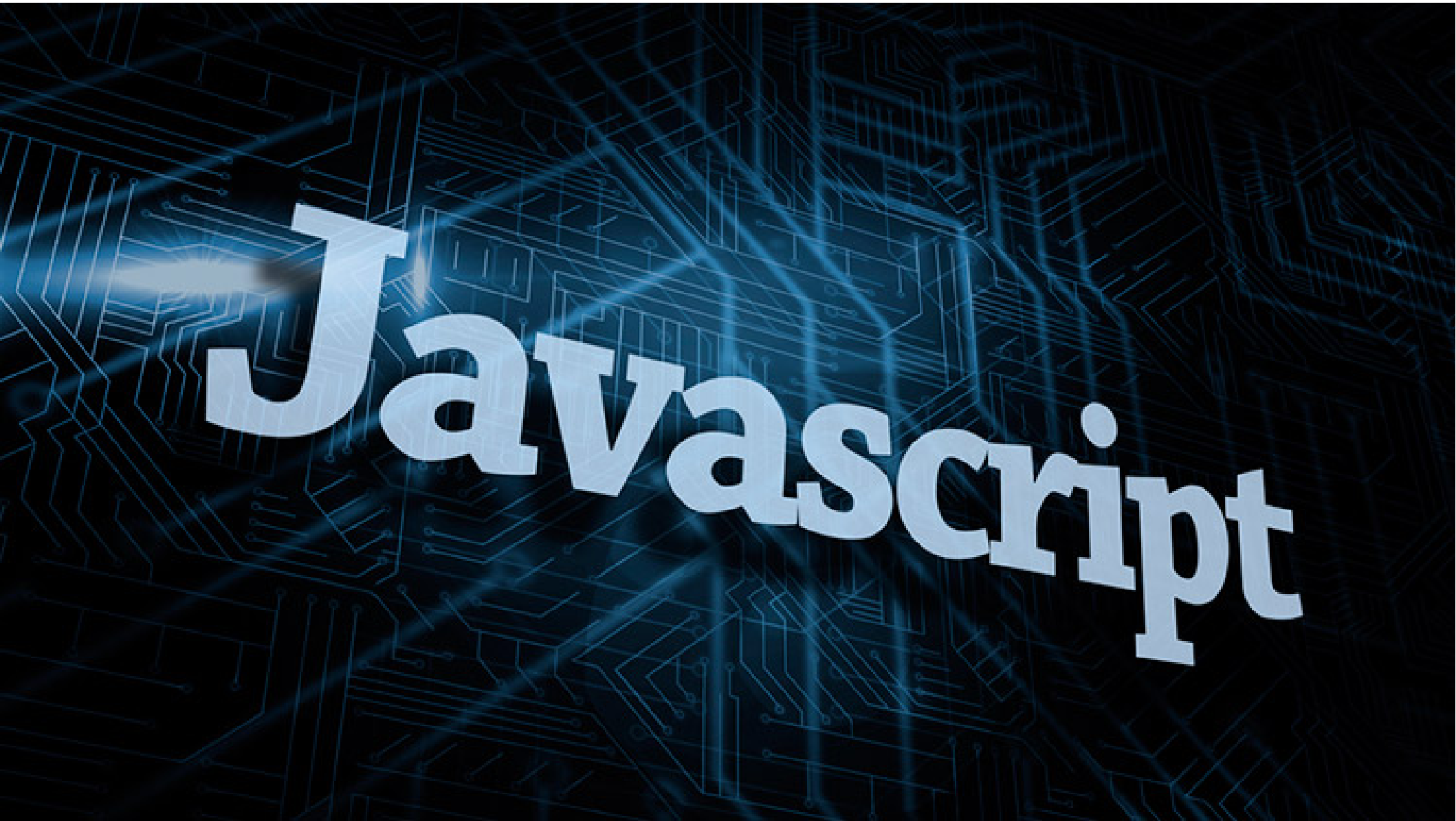
CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com

Validación de formularios con JavaScript



INDICE

Atributos de formularios

- **Type**: tipo de capo, cada campo lleva asociada una validación de tipo.
- **Required**: Si el campo es obligatorio de rellenar.
- **Pattern**: Permite definir una expresión regular para que personalizar la validación de tipo.
- **Placeholder**: Rellena el campo con un valor ilustrativo.
- **Title**: Permite añadir información extra al mensaje de error de validación.
- **Oninvalid**: Permite definir el código javascript a ejecutar cuando no se cumpla la validación.
- **Novalidate**: Aplicado sobre el input submit o sobre el formulario, no valida el formulario al hacer submit.

Expresiones regulares

- Las expresiones regulares son 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 son una forma sencilla para
  - Manipular datos.
  - Realizar búsquedas.
  - Reemplazar strings.

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]).

Expresiones regulares

Carácter	Texto buscado
[^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.

Expresiones regulares

Carácter	Texto buscado
\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_].

Ejemplos de expresiones regulares

Cualquier letra en minuscula	[a-z]
Entero	^(?:\+|-)?\d+$
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\-\.\?\,\'\/\\\+&%\$#_]*)?$
Contraseña segura	(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$ (Entre 8 y 10 caracteres, por lo menos un digito y un alfanumérico, y no puede contener caracteres espaciales)
Fecha	^\d{1,2}\/\d{1,2}\/\d{2,4}$ (Por ejemplo 01/01/2007)
Hora	^(0[1-9]|1\d|2[0-3]):([0-5]\d):([0-5]\d)$ (Por ejemplo 10:45:23)
Número 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}$
Certificado Identificación Fiscal	^(X(-|\.)?0?\d{7}(-|\.)?[A-Z]|[A-Z](-|\.)?\d{7}(-|\.)? [0-9A-Z]|\d{8}(-|\.)?[A-Z])$

Expresiones regulares

- En javascript para manejar expresiones regulares se tiene la tipología **RegExp**.

```
var reg = new RegExp("patron","flags");
```

- Aunque se pueden definir la expresiones de forma directa sin necesidad de hacer referencia al tipo **RegExp**, no es recomendable, ya que dificulta la comprensión.

```
var reg = /patron/flags
```

Expresiones regulares

- Para definir la expresión regular, se definen
  - patrón: La expresión
  - flag: Combinación de los siguientes valores
    - g: Indica que se realice una búsqueda global.
    - i: Indica que se ignoren mayúsculas o minúsculas.
    - m: Tratar caracteres de inicio y fin (^ y $) como inicio y fin de línea y no de texto.

Expresiones regulares

- Un ejemplo definido de las dos formas posibles

```
var re = new RegExp("\\w+");

var re = /\w+/;
```

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

Expresiones regulares

- Utilizando una representación literal de la expresión:

```
var re = /ab+c/;
```

  - La representación literal compila la expresión regular una vez que el script ha terminado de cargar.
  - Es recomendable utilizar esta representación cuando la expresión regular vaya a permanecer sin cambios durante la ejecución del script, puesto que ofrece un mejor desempeño.

Expresiones regulares

- Constructor del objeto RegExp:
  - El uso del constructor ofrece una compilación de la expresión regular en tiempo de ejecución.
  - Su uso es recomendable en aquellos escenarios en que el patrón de la expresión regular pueda cambiar durante la ejecución del script, o bien, se desconoce el patrón, dado que se obtiene desde otra fuente (cuando es suministrado por el usuario, por ejemplo).

```
var re = new RegExp("ab+c");
```

Expresiones regulares

- Los métodos que proporciona este tipo son
  - **patrón.exec(cadena)**: devuelve un array donde el elemento 0, tiene la primera correspondencia hallada en la cadena
  - **patrón.test(cadena)**: devuelve el booleano que indica si la cadena cumple o no con la expresión.

Expresiones regulares

- Se aplican expresiones en métodos de la clase String.
  - cadena.match(patron): devuelve el array con las coincidencias definidas por patrón, encontradas en cadena o null.
  - cadena.replace(patron,cadena2): devuelve un string, donde partiendo de cadena, se han sustituido las coincidencias con el patrón, por cadena2.
  - cadena.split(patron): devuelve un array, cuyos elementos son trozos de cadena, donde se han tomando como puntos de corte, la expresión indicada por el patrón.
  - cadena.search(patron): devuelve la posición de la primera coincidencia

Expresiones regulares

- **Ejercicio:** Haz un formulario que valide 2 campos:
  - Número de teléfono: 9 cifras numéricas
  - DNI: 8 cifras numéricas + carácter alfabético
  - Que la la evaluación la haga mediante JS, con una función que se llamada al enviar el formulario mostrando 3 alerts en función de dicha validación:
    - Campo TELEFONO no válido.
    - Campo DNI no válido.
    - Gracias por rellenarlo correctamente.
  - Pistas:

```
<form name="formulario" onSubmit="return ValidaCampos(this)">

function ValidaCampos(formulario) {
    return false; // sale de la función y NO envía el formulario
    // return true; // sale de la función y SÍ envía el formulario
}
```

Validación con JS

- Existe algo llamado "**Constraint API**" que son una serie de métodos y propiedades DOM de las cuales podemos hacer uso para un mayor control de nuestros formularios.
- Se pueden definir validaciones personalizadas, para ello se hace uso de javascript, y del evento **invalid** que lanzan los input, y que se asocia con un listener en los input, mediante la propiedad **oninvalid**.
    - **<input oninvalid**="metodoValidacionJS(this)" >
- Se ha incluido un propiedad en el objeto javascript que representa el input, llamada **input.validity**, que permite conocer que validación ha fallado.
- Además se puede personalizar el mensaje de error a mostrar con el método **input.setCustomValidity("");**

Validación con JS

- Propiedad **input.validity**: Devuelve un objeto con propiedades determinar si un input es valido

Validación con JS

- Propiedad **input.validity.patternMismatch:**
  - Devuelve true si el valor del input no concuerda con la expresion regular en el attributo pattern.

```
<input id="bar" pattern="[0-9]{4}" value="ABCD" />
<script>
    document.getElementById('bar').validity.patternMismatch; //true
</script>
```

Validación con JS

- Propiedad **input.validity.rangeOverflow:**
  - Devuelve true si el valor del input es max alto que su atributo max.

```
<input id="bar" type="number" max="2" value="3" />
<script>
    document.getElementById('bar').validity.rangeOverflow; //true
</script>
```

Validación con JS

- Propiedad **input.validity.rangeUnderflow:**
  - Devuelve true si el valor del input esta por debajo de su atributo min.

```
<input id="bar" type="number" min="2" value="1" />
<script>
    document.getElementById('bar').validity.rangeUnderflow; //true
</script>
```

Validación con JS

- Propiedad **input.validity.typeMismatch :**
  - Devuelve true si el valor del input no es del mismo tipo del especificado en el atributo type.

```
<input id="bar" type="url" value="foo" />
<input id="bar2" type="email" value="bar" />
<script>
    document.getElementById('bar').validity.typeMismatch; //true
    document.getElementById('bar2').validity.typeMismatch; //true
</script>
```

Validación con JS

- Propiedad **input.validity.valueMissing:**
  - Devuelve true si el input es tiene el atributo required y no tiene ningún valor

```
<input id="foo" type="text" required value="foo" />
<input id="bar" type="text" required value="" />
<script>
    document.getElementById('foo').validity.valueMissing; //false
    document.getElementById('bar').validity.valueMissing; //true
</script>
```

Validación con JS

- Propiedad **input.validity.stepMismatch:**
  - Devuelve true si el valor del input no es acorde con el especificado en el atributo step.
- Propiedad **input.validity.tooLong:**
  - Devuelve true si el valor del input es mayor que el especificado en el atributo maxlength, una cosa interesante aqui es que los navegadores no permiten que esto pase por efecto.

Validación con JS

- Propiedad **input.validity.valid:**
  - Devuelve true si todas las condiciones anteriores son false.

```
<input id="valid-1" type="text" required value="foo" />
<input id="valid-2" type="text" required value="" />
<script>
    document.getElementById('valid-1').validity.valid; //true
    document.getElementById('valid-2').validity.valid; //false
</script>
```

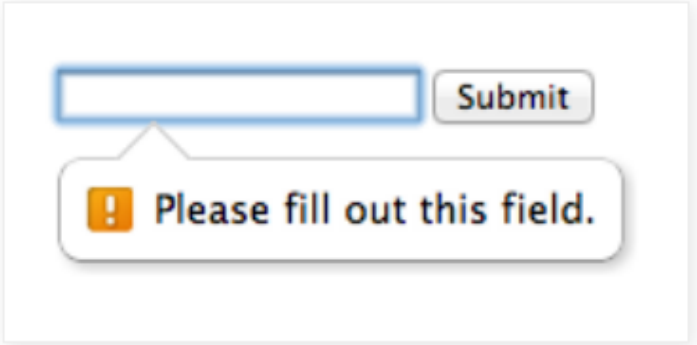
Validación con JS

- willValidate : Este método nos indica si un input sera validado o no cuando el formulario sea enviado.

```
<input type="text" id="name" />
<script>
    document.getElementById('name').willValidate; //true
</script>
```

Validación con JS

- validationMessage : Esta propiedad contiene el mensaje que se muestra cuando un input no cumple con las validaciones del formulario.




```
<input type="text" id="foo" required />
<script>
    document.getElementById('foo').validationMessage;
    //Chrome  --> 'Please fill out this field.'
    //Firefox --> 'Please fill out this field.'
    //Safari  --> 'value missing'
    //IE10    --> 'This is a required field.'
    //Opera   --> ''
</script>
```

Validación con JS

- CheckValidity:
  - Este es un método que devuelve true si un elemento de formulario contiene datos validos. (input, textarea, ect)

```
<form id="form-1">
    <input id="input-1" type="text" required />
</form>
<form id="form-2">
    <input id="input-2" type="text" />
</form>
<script>
    document.getElementById('form-1').checkValidity();  //false
    document.getElementById('input-1').checkValidity(); //false
    document.getElementById('form-2').checkValidity();  //true
    document.getElementById('input-2').checkValidity(); //true
</script>
```

Validación con JS

- checkValidity:
  - Cada vez que un elemento de formulario es verificado con "checkValidity" y falla, un evento "invalid" es disparado en ese elemento.
    - Podemos usar este evento para hacer cualquier cosa que queramos.
    - También como no existe el evento "valid" en su lugar se usa "change".

Validación con JS

- checkValidity:
  - Cada vez que un elemento de formulario es verificado con "checkValidity" y falla, un evento "invalid" es disparado en ese elemento.
    - Podemos usar este evento para hacer cualquier cosa que queramos.
    - También como no existe el evento "valid" en su lugar se usa "change".

Validación con JS

- checkValidity:

```
<form id="form-1">
    <input id="input-1" type="text" required />
</form>
<script>
document.getElementById('input-1').addEventListener('invalid', function()
{//you code here, whatever you wnat here
}, false);

document.getElementById('input-1').addEventListener('change', function(event)
{
    if (event.target.validity.valid) {
        //Field contains valid data.
    } else {
        //Field contains invalid data.
    }
}, false);

</script>
```

Validación con JS

- Ejercicio:
  - Añade dos campos al formulario anterior,
    - El primero de tipo text para recibir un mensaje (debe tener al menos 1 carácter),
    - Y otro de tipo email, que llame a una función JS llamada **validacionPersonalizadaEmail(input) {}** que cambie el mensaje de validación por el contenido del primer campo.