

JavaScript. Nivel II

julio, 2019



Objetivos del nivel

- Manipular el objeto Window del DOM.
- Aprender a manipular el contenido de una página.
- Aprender a validar formularios.

Prerrequisitos del nivel

- JavaScript Nivel I
- Programación Orientada a Objetos Nivel II
- Lógica de Programación Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2019. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Funciones. Parte 3

- 1.1.- Funciones Anónimas.
- 1.2.- Addeventlistener.
- 1.3.- El This.

Capítulo 2. El Dom

- 2.1.- ¿Qué es la jerarquía de objetos?.
- 2.2.- El Objeto Window.
- 2.3.- Atributos de Window.

Capítulo 3. Window Open

- 3.1.- El Método Open.
- 3.2.- Manipular la Ventana.
- 3.3.- El Opener.

Capítulo 4. Dom - Document. Parte 1

- 4.1.- El Objeto Document.
- 4.2.- Modificar el Documento.
- 4.3.- Agregar Contenido al Documento.

Capítulo 5. Dom - Document. Parte 2

- 5.1.- Buscar Elementos.
- 5.2.- Manipular Elementos.

Capítulo 6. Arreglos en el Dom

- 6.1.- Elements.
- 6.2.- Forms.
- 6.3.- Childs.

Capítulo 7. Dom - Form. Parte 1

- 7.1.- El Objeto Form.
- 7.2.- El Objeto Input.
- 7.3.- Manipular Valores.

Capítulo 8. Dom - Form. Parte 2

- 8.1.- Input Type Submit.
- 8.2.- Onsubmit.
- 8.3.- Método Submit.

Capítulo 9. Validación de Formularios. Parte 1

- 9.1.- Validación de Formularios.
- 9.2.- Validar Cuadros de Texto.

Capítulo 10. Validación de Formularios. Parte 2

- 10.1.- Validar Select.
- 10.2.- Validar Checkbox.

Capítulo 11. Expresiones Regulares

- 11.1.- Concepto.
- 11.2.- Patrones.
- 11.3.- Validar Correo.

Capítulo 12. Eventos. Parte 2

- 12.1.- Eventos de Teclado.
- 12.2.- Validar Teclas.

Capítulo 13. Ejecución Automática

- 13.1.- Settimeout.
- 13.2.- Setinterval.

Capítulo 1. FUNCIONES. PARTE 3

1.1.- Funciones Anónimas

Existe otra forma de declarar las funciones, que se denomina funciones anónimas. Básicamente es lo mismo, solo que la declaración varía ligeramente. El siguiente es un ejemplo de cómo se declara una función anónima:

```
<script language="javascript">
    var calcular_bono= function (nro, monto)
    {
        if (nro>0)
            return monto*0.5;
        else
            return monto*0.1;
    }
    var bono= calcular_bono(5,1500);
    alert("El bono es de "+bono+" bs");
</script>
```

1.2.- Addeventlistener

La forma vista hasta ahora para asignar manejadores de eventos a un elemento es directamente en la declaración de la etiqueta HTML, sin embargo, existe otra forma de agregar un manejador de evento en el código JavaScript. La forma es la siguiente:

```
objeto.addEventListener("evento" , funcion );
ó
objeto.evento = function;
```

Donde "objeto" es el objeto al cual se le desea agregar el manejador de evento, "evento" es el evento del cual se desea manejar y "funcion" el código que se ejecutará al dispararse el evento. Por ejemplo:

```
boton.addEventListener("click", function (){ alert("hola mundo") });
```

Otra forma de hacerlo es la siguiente:

```
boton.onclick= function(){ alert("Hola mundo") };
```

1.3.- El This

El elemento "this" puede usarse en JavaScript en distintos contextos. Por ahora, se usará en el contexto de la ejecución de un evento. This hace referencia al objeto que está disparando un evento en un momento dado. A través de este elemento, se puede acceder a todos los atributos o métodos del objeto en cuestión, si es un input, si es un form o si es cualquier otro. Por ejemplo:

```
<input type="button" value="Guardar" onclick="calcular(this)" />
```

En este ejemplo, se pasa por parámetro "this" que hace referencia al botón sobre el cual se está haciendo click. En el interior de la función "calcular" se podrá hacer referencia a los atributos y métodos del botón.

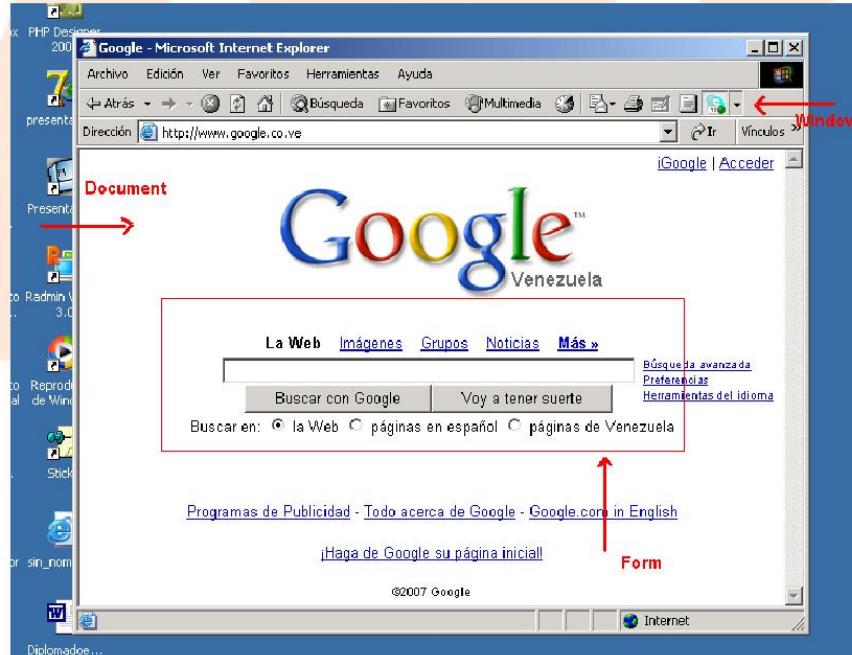
Capítulo 2. EL DOM

2.1.- ¿Qué es la jerarquía de objetos?

Cuando se carga una página, el navegador crea una jerarquía de objetos en memoria llamado DOM (Document Object Model), que sirven para controlar los distintos elementos de dicha página. Con JavaScript se puede manipular esa jerarquía de objetos, acceder a sus propiedades e invocar sus métodos.

Cualquier elemento de la página se puede controlar de una manera u otra accediendo a esa jerarquía. Es crucial conocerla bien para poder controlar perfectamente las páginas web con JavaScript o cualquier otro lenguaje de programación del lado del cliente. Los tres objetos más importantes de la jerarquía son:

- Window: la ventana del navegador.
- Document: la página que se está visualizando.
- Form: los formularios que están en la página (si posee)



2.2.- El Objeto Window

Es el objeto principal en la jerarquía. Contiene las propiedades y métodos para controlar la ventana del navegador. De él dependen todos los demás objetos de la jerarquía. A continuación, podemos ver las propiedades del objeto window:

- document: es la página que está cargada en el momento.
- history: el historial de páginas visitadas en esa ventana.
- location: la dirección de la página que está cargada en el momento.
- menubar: la barra de menú de navegador.
- locationbar: la barra de direcciones del navegador.
- statusbar: la barra de estado del navegador.
- frames: arreglo de marcos de la página (si tiene).
- screen: permite conocer el estado de la pantalla.
- navigator: permite conocer datos del navegador.

También tiene unos métodos (o procedimientos) que permiten ejecutar acciones sobre la ventana del navegador. Los más importantes son:

- alert: muestra un mensaje o dialogo estilo messagebox.
- confirm: muestra un dialogo con los botones Aceptar y Cancelar.
- home: carga la URL definida como HOME en el navegador.
- print: Imprime el contenido de la ventana.
- open: abre una nueva ventana del navegador.
- close: cierra la ventana del navegador (sólo funciona en ventanas abiertas con open).
- moveTo: mueve la ventana a las coordenadas especificadas (sólo funciona con ventanas abiertas con open).
- resizeTo: redimensiona la ventana a los nuevos valores de ancho y alto (sólo funciona con ventanas abiertas con open).

Ejemplos de como usar estos métodos y propiedades del objeto Window son:

```
<script language="javascript">

window.resizeTo (400, 400); // cambia la ventana del navegador a 400x400
window.print(); // hace que aparezca el dialogo para imprimir
window.location = "http://www.google.com"; // hace que se cargue google
window.open("http://www.cantv.net","","","");
window.close(); // cierra la ventana del navegador

</script>
```

2.3.- Atributos de Window

Entre los atributos más usados del objeto Window están: screen (para conocer información de la pantalla) y navigator (para conocer información del navegador). Los atributos de screen son:

- screen.width: ancho de la pantalla.
- screen.height: alto de la pantalla.
- screen.availWidth: ancho disponible de la pantalla.
- screen.availHeight: alto disponible de la pantalla.
- screen.colorDepth: profundidad del color.
- screen.pixelDepth

Los atributos de navigator son:

- navigator.appName: nombre del navegador.
- navigator.appVersion: versión del navegador.
- navigator.product: motor del navegador.
- navigator.userAgent: indica el user-agent enviado al servidor.
- navigator.platform: sistema operativo.
- navigator.language: idioma del navegador.
- navigator.onLine: indica si está en línea la computadora.
- navigator.cookieEnabled: indica si están habilitadas las cookies en el navegador.

La información que retorna el objeto "navigator" en ocasiones puede ser imprecisa y no debe ser usada para detectar del navegador.

Capítulo 3. WINDOW OPEN

3.1.- El Método Open

Uno de los métodos más usados es el "open", que permite abrir una ventana del navegador. La forma general de llamar al método open es:

```
window.open(url, name, specs, replace)
```

El método open tiene 3 parámetros:

- la url del documento que se necesita cargar. Puede ser un archivo del sitio o un archivo de otro sitio.
- name: es un parámetro opcional que indica el nombre de la ventana o la forma en que se va a abrir la ventana, si es una ventana nueva se usa "_blank", si es la misma ventana se utiliza "".
- specs: es un parámetro opcional, que establece parámetros del aspecto de la ventana que se abrirá.
- replace: es un parámetro opcional.

Un ejemplo de uso de open es:

```
window.open("http://www.google.com", "", "width=200,height=100");
```

3.2.- Manipular la Ventana

Una página abierta por medio de "open" puede manipular el tamaño de la ventana y la posición de la misma. Al abrir una ventana con "open" se establece el ancho y el alto de la misma, pero se puede cambiar el tamaño de ésta utilizando el método "resizeTo", que recibe por parámetro el ancho y el alto al cual se desea redimensionar la ventana. También se puede cambiar la posición de la ventana utilizando el método "moveTo" que recibe por parámetro la posición (x,y) a la cual se desea mover la ventana.

Capítulo 4. DOM - DOCUMENT. PARTE 1

4.1.- El Objeto Document

Con el objeto document se controla la página web y todos los elementos que contiene. El objeto document es la página actual que se está visualizando en un momento dado. Depende del objeto window, pero también puede depender del objeto frames en caso de que la página se esté mostrando en un conjunto de marcos (aunque los marcos son poco usados).

El objeto "document" al igual que el "window" tiene propiedades y métodos para manipularlo. Las propiedades más usadas son:

- forms: un arreglo con los formularios que tiene el documento.
- parent: el objeto que contiene al documento. El window o un frame.
- bgColor: el color de fondo del documento.
- title: el título de la página.
- lastModified: fecha de la última modificación del documento.
- fgColor: Color por defecto del texto.
- linkColor: Color de los enlaces.
- domain: Nombre del dominio del servidor de la página.

Aunque poco común, con los métodos del objeto document se puede escribir información dinámicamente en la página. Para tal fin se usan los siguientes métodos:

- open: Abre una corriente de datos para escribir en el documento (con los métodos write o writeln) .
- write: Escribe texto a la corriente de datos del documento.
- writeln: Igual que write pero finaliza el texto con un retorno de carro.
- close: Cierra una corriente de datos abierta por el método open y hace que se muestren todos los elementos.

4.2.- Modificar el Documento

Se pueden modificar algunos aspectos del documento, tales como el título o el color de fondo de la página.

4.3.- Agregar Contenido al Documento

Crear o modificar el contenido de la página en tiempo de ejecución es una práctica poco habitual. Se puede crear el contenido completamente o agregarle contenido. Para modificar el contenido de algún elemento específico se utilizarán otros métodos que se estudiaran en el siguiente capítulo. Para agregar contenido se utilizan los métodos write y writeln. Por ejemplo:



Capítulo 5. DOM - DOCUMENT. PARTE 2

5.1.- Buscar Elementos

Todos los elementos (etiquetas) que están en el body de un documento html son parte del "document". Es una tarea muy común manipular alguno de estos elementos en tiempo de ejecución. Para manipular un elemento del documento primero se debe tener la referencia a éste. Se puede buscar un elemento por su "id", por su "name", por su "classname" o por su "tagname". Para cada uno de los casos, el objeto "document" posee un método para buscar. La forma más común de manipular un elemento particular es buscándolo por su "id" (asumiendo que el "id" es único). El nombre de la función es:

- `getElementById`: busca un elemento por su "id".

Los otros métodos de búsqueda son

- `getElementsByName`.
- `getElementsByTagName`.
- `getElementsByClassName`

Pero estos retornan un arreglo de elementos. Estos métodos serán estudiados en el siguiente capítulo.

Al buscar un elemento por su id, el método retornará una referencia al elemento buscado, en caso de no encontrar el elemento, retornará el valor "undefined". El siguiente es un ejemplo para buscar un elemento por su "id":

```
if (document.getElementById("idElemento") == undefined)
    alert("El elemento no existe");
else
{
    // se hace algo
}
```

El elemento debe estar haber sido leído por el navegador antes de intentar hacer referencia a este.

5.2.- Manipular Elementos

Luego de encontrado el elemento, conociendo el tipo de elemento, se puede manipular, ya sean sus propiedades particulares o los estilos CSS asociados a este elemento. Por ejemplo, si el elemento buscado es un "<div>", un "<p>" o un "<h>", se puede modificar el contenido del mismo con la propiedad "innerHTML". Por ejemplo:

```
var elemento = document.getElementById("idElemento");
if (elemento == undefined) alert("El elemento no existe");
else
{
    elemento.innerHTML = "Nuevo contenido"
}
```

Asimismo, se puede modificar las propiedades del CSS de cualquier elemento, o de varios elementos (en caso de buscar por class). Para lograrlo, se utiliza el atributo "style" y luego la propiedad específica del estilo CSS que se necesita manipular. La forma general es la siguiente:

```
document.getElementById("idElemento").style.propiedad = "valor";
```

Por ejemplo, si se necesita establecer a rojo el color de fondo de un elemento "<p>" con el id "resultado", se debe utilizar la siguiente instrucción:

```
document.getElementById("resultado").style.backgroundColor = "red"
```

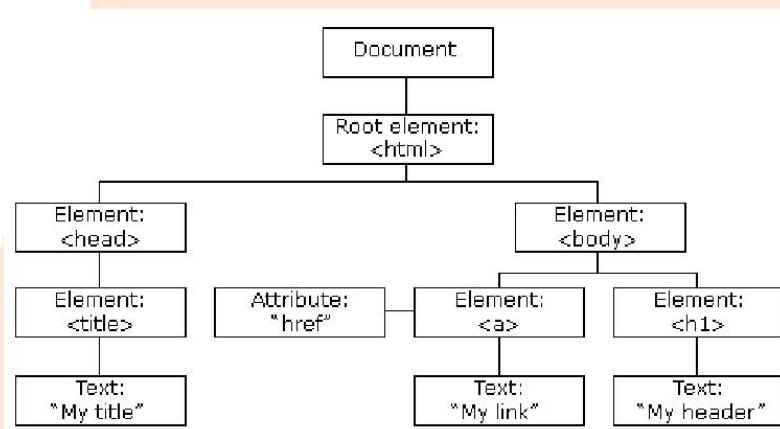
Otra forma de hacerlo:

```
var elemento = document.getElementById("resultado");
elemento.style.backgroundColor = "red"
```

Capítulo 6. ARREGLOS EN EL DOM

6.1.- Elements

Element es la clase base más general de la que heredan todos los objetos de un Documento. Sólo tiene métodos y propiedades comunes a todo tipo de elementos. En el DOM, un Element representa una etiqueta HTML. Adicionalmente, los elementos se relacionan entre estos según como estén contenidos unos dentro de otros. La siguiente imagen muestra la relación que existe entre los elementos de un documento HTML:



En el capítulo anterior se explicó cómo buscar un elemento individual, aunque se pueden buscar varios elementos. Los siguientes métodos para buscar elementos en el DOM retornan un arreglo con los elementos encontrados:

- `getElementsByName`: devuelve los elementos especificados en `TagName`.
- `getElementsByName`: devuelve los elementos especificados en `Name`.
- `getElementsByClassName`: devuelve los elementos de la clase especificada.
- `document.forms`: devuelve un arreglo de elementos de tipo `<form>`.
- `document.images`: devuelve un arreglo de elementos de tipo ``.

El siguiente ejemplo muestra como buscar todos los elementos HTML de tipo `<p>` y cambiar el color del fondo a través de la propiedad `style` del elemento:

```
var listaNodo = document.getElementsByTagName("p");
for (var i = 0; i < listaNodo.length; i++) {
    listaNodo[i].style.backgroundColor = "red";
}
```

6.2.- Forms

Uno de los elementos más importantes del DOM es el elemento HTML <form>. El objeto `document.forms` es un arreglo que contiene todo los formularios del documento y cada form a su vez contiene un arreglo de elementos (`form.elements`) que contiene todos los input, select, textarea, checkbox, etc, que estan adentro del formulario. Un ejemplo de cómo buscar todos los forms en el documento HTML es el siguiente:

```
for (var i = 0; i < document.forms.length; i++) {
    console.log(document.forms[i].name); //muestra el nombre del formulario identificado en el name
    console.log(document.forms[i].method); // muestra el metodo de envio : get/post
    console.log(document.forms[i].action); // muestra el action ej: procesar.php
}
```

El arreglo `forms` puede ser accedido por su índice numérico o su nombre, es decir, `form[0]` devuelve el primer formulario, y `form['form_login']` accede a un formulario con el `name='form_login'`. Un ejemplo de cómo recorrer todos los elementos de cada form es:

```
var form=document.forms['f1'];
for (var i = 0; i < form.elements.length; i++) {
    console.log(form.elements[i].type); //muestra el tipo de input : text, select, submit, checkbox, etc.
    console.log(form.elements[i].value); //muestra el valor del input
}
```

6.3.- Childs

Un objeto Element puede tener nodos hijos (child) que pueden ser de tipo element, texto o comentario. Los elementos contenidos en un objeto element pueden ser accedidos como un arreglo de objetos (nodos child) llamados `NodeList`.

Un objeto NodeList representa una lista de nodos, que es una colección de nodos hijos (child) contenidos en un elemento HTML. Además, un nodo puede tener atributos que se denotan justo después del nombre de la etiqueta por ejemplo el elemento posee un atributo "src" que indica la ruta de la imagen.

Cada elemento tiene las siguientes propiedades:

- element.id : Establece o devuelve el identificador del elemento.
- element.style: Establece o devuelve el atributo de estilo del elemento.
- element.className: Establece o devuelve el atributo de clase del elemento.
- element.tagName: Devuelve el nombre de la etiqueta del elemento.
- element.innerHTML: Establece o devuelve el contenido del elemento.
- element.textContent: Establece o devuelve el texto del nodo.

También tiene métodos más usados que permiten manipular los hijos del elemento, ya sea para crear nuevos elementos o eliminarlos. Los métodos más usados son:

- createElement: creará un elemento HTML que se insertará posteriormente en un documento HTML.
- createTextNode: crea un nodo de texto con el texto especificado.
- appendChild: Añade un nuevo nodo al elemento como el último nodo hijo.
- insertBefore : inserta un elemento antes del nodo child especificado.
- childNodes: Devuelve el elemento NodeList de los nodos secundarios.
- getAttribute: Devuelve el valor de la propiedad especificada de un elemento.
- removeAttribute: Elimina el atributo especificado del elemento.

El siguiente ejemplo muestra cómo crear un elemento child e insertarlo al final del HTML:

```
// Buscar elemento padre
var elemento_paadre = document.body;
// Crear elemento
var titulo = document.createElement('h1');
// Crear el nodo de texto
var texto = document.createTextNode("mi título");
// Adjuntar el nodo de texto al elemento h1
titulo.appendChild(texto);
// Ahora sí, insertar (adjuntar) el elemento hijo (título) al elemento padre (body)
elemento_paadre.appendChild(titulo);
```

Capítulo 7. DOM - FORM. PARTE 1

7.1.- El Objeto Form

El objeto form depende en la jerarquía de objetos del objeto document. Además, de un formulario dependen todos los elementos que hay dentro de éste, como pueden ser campos de texto, cajas de selección, áreas de texto, botones de radio, etc. Para acceder a un formulario desde el objeto document se puede hacer de dos formas:

1. A partir de su nombre, asignado con el atributo NAME de HTML.
2. Mediante el arreglo de formularios del objeto document, con el índice del formulario al que se desea acceder. Por ejemplo: document.forms[0].

Por ejemplo, si en el body hay un formulario como el siguiente:

```
<FORM name="f1">
    <INPUT type=text name=campo1>
    <INPUT type=text name=campo2>
</FORM>
```

Se puede acceder a las propiedades y elementos del formulario de las siguientes maneras: document.f1 o con su índice document.forms[0] (suponiendo que el formulario al cual se desea acceder es el primero de la página).

Algunas de las propiedades de los formularios son:

- action: dirección URL a la que se enviarán los datos del formulario.
- elements: La matriz contiene cada uno de los campos del formulario.
- method: El método de envío del formulario.
- target: La ventana o frame en la que está dirigido el formulario. Cuando se envía el formulario, el resultado se muestra en la ventana o frame indicado.

Estos son los métodos que se pueden usar de un formulario:

- submit: para hacer que el formulario se envíe, aunque no se haya pulsado un input type submit.

- reset: Para reinicializar todos los campos del formulario, como si se hubiese pulsado un input type reset.

7.2.- El Objeto Input

Un elemento input contiene propiedades independientemente del "type" y otros que sí son dependientes del "type". Los atributos comunes son los siguientes:

- defaultValue: Es el valor por defecto que tiene un campo.
- value: El texto que hay escrito en el campo.
- disabled: si es true indica que el cuadro de texto estará inactivo.
- form : Hace referencia al formulario al que pertenece el elemento.

Para acceder a los elementos que están adentro de un formulario se utiliza el name del mismo. Por ejemplo:

```
<FORM name="f1">
    <INPUT type="text" name="campo1">
    <INPUT type="text" name="campo2">
</FORM>
```

Se podría acceder al "campo1" del formulario de dos maneras: por el nombre (document.f1.campo1 o document.forms[0].campo1) o a partir del array de elementos "elements" (document.f1.elements[0] o document.forms[0].elements[0]). También puede buscarse un elemento por su "id" si lo tiene.

Capítulo 8. DOM - FORM. PARTE 2

8.1.- Input Type Submit

Para enviar los datos contenidos en los elementos de captura que están adentro de un formulario (inputs, selects y textarea) a la URL especificada en la propiedad "action" utilizando el método de envío especificado en la propiedad "method", existen 2 alternativas:

- colocar un input type "submit" (botón) que realice automáticamente el envío.
- ejecutar el método "submit" del formulario.

La más simple es la primera. Si no se coloca ningún valor en la propiedad "action", los datos se envian a la misma página. Si la propiedad "method" es omitida, se asume por defecto que los datos se enviaran usando el método HTTP "GET".

8.2.- Onsubmit

Cuando se intenta enviar los datos de un formulario al hacer click en un input type "submit", se dispara el evento "onsubmit" del formulario. En la ejecución de este evento se puede evitar que el formulario se envíe. Por ejemplo:

```
<form onsubmit=".... ">  
  
    <input type="submit" value="Enviar" />  
</form>
```

8.3.- Método Submit

Si por alguna razón no hay un input type "submit", para enviar los datos del form se debe ejecutar el método "submit" del form que contiene los datos que se desean enviar. Por ejemplo:

```
form1.submit();
```

Si no se ha especificado la propiedad "action", el efecto conseguido es que la página se recargará y se borrarán los datos de los inputs. Los datos deberían enviarse a una URL que contenga código del lado del servidor o BackEnd para ser procesados, por ejemplo, una autenticación, una búsqueda, etc. El evento "onsubmit" no se dispara con el llamado del método submit del formulario.



Capítulo 9. VALIDACIÓN DE FORMULARIOS. PARTE 1

9.1.- Validación de Formularios

Uno de los mayores usos que se le da a JavaScript es el de realizar validaciones del lado del cliente para así dar al usuario una respuesta más rápida y descongestionar al servidor al evitar que trate con formularios que probablemente tengan errores que pudieron validarse antes de enviarse.

Las validaciones con JavaScript no deberían eliminar a las validaciones del lado del servidor (hechas con cualquier lenguaje del lado del servidor), debido a que las validaciones del lado del cliente pueden ser alteradas u omitidas por usuarios expertos. La idea básicamente es crear funciones JavaScript en nuestra página que validen los datos introducidos por el usuario. Si los datos son válidos, se envían a la página establecida en el action, si no, se muestra un mensaje de error, con alert o en el documento y no se envían los datos.

Las validaciones de formularios se pueden hacer de 2 formas. La primera es colocando un botón tipo "button" en el formulario y programar el evento "onclick" del botón, en el cual se llama la función de validación. Esta función tendrá las validaciones (condicionales) apropiadas. Si se cumplen las condiciones, se llama el método "submit()" del formulario, sino, se muestran errores y no se envía el formulario. A la función se le envía por parámetro el formulario al que pertenece el botón que la llama (this.form), que contiene los elementos a ser verificados. La siguiente imagen muestra un ejemplo:

```
<script language="javascript">
    function validar(form)
    {
        if ( no_se_cumplen_validaciones )
            alert("Error ....");
        else
            form.submit();
    }
</script>
<body >
    Ejemplo de validaciones
    <form name="f1" action="">
        Cedula <input type="text" name="cedula" />
        Nombre <input type="text" name="nombre" />
        <input type="button" name="btnenviar" value="Enviar"
              onclick="validar(this.form)"/>
    </form>
</body>
```

La otra forma es programando el evento "onsubmit" del formulario que se desea validar. En este caso, el botón debe ser tipo "submit" para que dispare el evento del formulario. En el evento, se hace el llamado a la función de validación, la cual retornará "true" si las validaciones fueron correctas o "false" si no se cumplen las condiciones. El llamado es ligeramente diferente, porque debe colocarse la palabra "return" en el llamado de la función en el evento "onsubmit", de lo contrario se enviará el formulario independientemente de lo que retorne la función:

```
<script language="javascript">
    function validar(form)
    {
        if ( condiciones_de_validacion )
        {
            alert("Error ....");
            return false;
        }else
            return true;
    }
</script>
<body >
    Ejemplo de validaciones
    <form name="f1" action="" onsubmit="return validar(this)">
        Cedula <input type="text" name="cedula" />
        Nombre <input type="text" name="nombre" />
        <input type="submit" name="btnenviar" value="Enviar"/>
    </form>
</body>
```

9.2.- Validar Cuadros de Texto

Una de las validaciones más comunes es evitar que un cuadro de texto esté vacío al momento de enviar el formulario (campos obligatorios). Para ello se debe evaluar el contenido del input, determinando si es igual a la cadena vacía o si la longitud de la cadena es 0 (la propiedad length de los string). Conociendo el DOM se pueden realizar algunas instrucciones extras, como usar "setfocus" para ubicar el cursor en el cuadro de texto que no cumple con la validación. En la siguiente imagen se muestra un ejemplo:

```
<script language="javascript">
    function validar(form)
    {
        if ( form.cedula.value == "" )
        {
            alert("La cedula es obligatorio");
            form.cedula.focus();
            return false;
        }else
            return true;
    }
</script>
<body >
    Ejemplo de validaciones
    <form name="f1" action="" onsubmit="return validar(this)">
        Cedula <input type="text" name="cedula" />
        Nombre <input type="text" name="nombre" />
        <input type="submit" name="btnenviar" value="Enviar"/>
    </form>
</body>
```

Si hace falta validar varios cuadros de texto obligatorios, se puede hacer una condición anidada:

```
<script language="javascript">
    function validar(form)
    {
        if ( form.cedula.value == "" )
        {
            alert("La cedula es obligatorio");
            form.cedula.focus();
            return false;
        }else
            if ( form.nombre.value == "" )
            {
                alert("El nombre es obligatorio");
                form.nombre.focus();
                return false;
            }else
                return true;
    }
</script>
<body >
    Ejemplo de validaciones
    <form name="f1" action="" onsubmit="return validar(this)">
        Cedula <input type="text" name="cedula" />
        Nombre <input type="text" name="nombre" />
        <input type="submit" name="btnenviar" value="Enviar"/>
    </form>
</body>
```

También se puede hacer una función reutilizable en caso de que se tengan muchos campos obligatorios que validar, de forma tal que se tenga que repetir menos código:

```
<script language="javascript">
    function esta_vacio(campo,mensaje)
    {
        if ( campo.value == "" )
        {
            alert(mensaje);
            campo.focus();
            return true;
        }else
            return false;
    }
    function validar(form)
    {
        if ( esta_vacio(form.cedula,"La cedula es obligatorio"))
            return false;
        else
            if (esta_vacio(form.nombre,"El nombre es obligatorio"))
                return false
            else
                return true;
    }
</script>
```

Capítulo 10. VALIDACIÓN DE FORMULARIOS. PARTE 2

10.1.- Validar Select

Para validar los select se puede utilizar la propiedad "value" que retorna el valor de la propiedad "value" del elemento seleccionado, o la propiedad "selectedIndex", que indica la posición del elemento seleccionado. Si un select tiene la propiedad "multiple" (que indica que se pueden seleccionar varios elementos o ninguno), y el usuario no ha seleccionado algún elemento, la propiedad "selectedIndex" tomará el valor -1. La siguiente función muestra cómo evaluar un select, si hay algún elemento seleccionado, retorna true sino retorna false:

```
<script language="javascript">
    function validar_select(select)
    {
        if (select.selectedIndex ==-1)
        {
            alert("Debe seleccionar un elemento de la lista");
            select.focus();
            return false;
        }
        else
            return true;
    }
</script>
```

La función que valida todo el conjunto debe llamar a la función que valida el select:

```
<script language="javascript">
    function validar(form)
    {
        if (esta_vacio(form.cedula,"La cedula es obligatorio"))
            return false;
        else
            if (validar_select(form.ciudad)==false)
                return false;
            else
                return true;
    }
</script>
<body >
    Ejemplo de validaciones
    <form name="f1" action="" onsubmit="return validar(this)">
        Cedula <input type="text" name="cedula" onkeypress="return validar_numero(event)"/>
        Nombre <input type="text" name="nombre" onkeypress="return validarletra(event)"/>
        Correo <input type="text" name="correo" />
        Ciudad <select name="ciudad" multiple>
            <option value="0">Seleccione</option>
            <option value="1">Barquisimeto</option>
            <option value="2">Caracas</option>
        </select>
        <input type="submit" name="btnenviar" value="Enviar"/>
    </form>
</body>
```

10.2.- Validar Checkbox

Para validar un checkbox solo es necesario conocer la propiedad "checked", que tendrá el valor "true" si el checkbox está marcado y el valor "false" si no está marcado. El siguiente ejemplo muestra cómo hacer referencia a esa propiedad:

```
<script language="javascript">
    function validar_checkbox(check)
    {
        if (check.checked == false)
        {
            alert("Debe seleccionar la opcion");
            return false;
        }else
            return true;
    }
</script>
```

La función de validación completa queda de la siguiente forma:

```
<script language="javascript">
    function validar(form)
    {
        if (esta_vacio(form.cedula,"La cedula es obligatorio"))
            return false;
        else
            if (validarcorreo(form.correo)==false)
                return false;
            else
                if (validar_select(form.ciudad)==false)
                    return false;
                else
                    if (validar_checkbox(form.condiciones))
                        return true;
                    else
                        return false;
    }
</script>
```

Capítulo 11. EXPRESIONES REGULARES

11.1.- Concepto

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. En JavaScript, las expresiones regulares también son objetos. Una expresión regular puede crearse de cualquiera de las dos siguientes maneras:

- Utilizando una representación literal de la expresión regular, consistente en un patrón encerrado entre diagonales, como a continuación:

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

- Llamando a la función constructora del objeto RegExp, como a continuación:

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

Algunos usos comunes de las expresiones regulares son:

- verificar si un texto tiene solo números.
- verificar si un texto tiene solo letras.
- verificar si un texto tiene una combinación de caracteres particular, como una placa, por ejemplo: AE477EA.
- verificar si un texto tiene la estructura correcta de un correo electrónico, por ejemplo: admin@cadif1.com
- verificar si un texto tiene la estructura correcta de una URL para acceder a un sitio web, por ejemplo: cadif1.com.

11.2.- Patrones

Un patrón de expresión regular se compone de caracteres simples, como /abc/, o una combinación de caracteres simples y especiales, como /ab*c/ o /Chapter (d+).d*/. Los patrones simples se construyen con caracteres para los que se desea una coincidencia exacta. Por ejemplo, el patrón /abc/ coincidirá sólo con esta exacta secuencia y orden de caracteres "abc". Tal expresión tendría resultados en las cadenas "Hola, ¿conoces tu

abc?". Luego de que el patrón es construido, se utiliza la expresión regular para verificar si una cadena cumple o no con la expresión. La forma de verificar es usando el método "test" de la clase RegExp, de la siguiente forma:

```
var re = new RegExp("ab+c");
if (re.test("xyz"))
    console.log("La cadena no coincide con la expresion regular");
else
    console.log("La cadena si cumple con la expresion regular");
```

Para conocer las reglas para hacer patrones, consultar la URL: developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

11.3.- Validar Correo

Para validar el correo electrónico se utiliza una expresión regular. Se construye una expresión que verifica si cumple con las condiciones apropiadas: tener un @, tener al menos 1 letra antes y después del @, tener un punto (.) para el dominio después del @. Se muestra el ejemplo una función que retorne true si la cadena que está escrita en el cuadro texto tiene la estructura válida de una dirección de correo:

```
function validarcorreo(correo)
{
    regex = /^[0-9a-zA-Z]+[0-9a-zA-Z]*@[0-9a-zA-Z]+[0-9a-zA-Z]+\.[a-zA-Z]{2,9}$/;
    if (regex.test(correo.value))
        return true;
    else
    {
        alert("Formato de correo incorrecto");
        correo.focus();
        return false;
    }
}
```

Esta función se puede colocar en una librería para ser reutilizada en todas las páginas donde hayan formularios que soliciten correo electrónico. En la misma función de validación donde se verifican todos los campos, se puede llamar esta función. En la siguiente imagen se muestra un ejemplo:

```
<script language="javascript">
    function validar(form)
    {
        if (esta_vacio(form.cedula,"La cedula es obligatorio"))
            return false;
        else
            if (esta_vacio(form.nombre,"El nombre es obligatorio"))
                return false
            else
                if (esta_vacio(form.correo,"El correo es obligatorio"))
                    return false;
                else
                    if (validarcorreo(form.correo)==false)
                        return false;
                    else
                        return true;
    }
</script>
```



Capítulo 12. EVENTOS. PARTE 2

12.1.- Eventos de Teclado

Cuando se presiona una tecla del teclado se disparan 3 eventos:

- keydown: ocurre cuando el usuario está presionando una tecla.
- keypress: ocurre cuando el usuario presiona una tecla.
- keyup: ocurre cuando el usuario suelta la tecla presionada.

Cuando un usuario presiona una tecla, los 3 eventos se disparan en ese orden. El evento onkeypress no se dispara con todas las teclas (por ejemplo: ALT, CTRL, SHIFT, ESC) en todos los navegadores. Para detectar si el usuario ha presionado una tecla se debe usar el evento keydown porque funciona en todos los navegadores.

Para esto, debe usarse el evento "keypress" de los input. El evento "keypress" genera un dato llamado "event", que permite, entre otras cosas, conocer la tecla presionada por el usuario. En realidad, es un objeto que posee las siguientes propiedades:

- charCode: el código ASCII de la tecla presionada. No funciona en algunos navegadores. Se utiliza para caracteres imprimibles.
- keyCode: el código ASCII de la tecla presionada. Se utiliza para caracteres no imprimibles, tales como: tabulador, backspace, enter, entre otros.
- shiftKey: indica si la tecla esta shift presionada.
- ctrlKey: indica si la tecla esta ctrl presionada.
- altKey: indica si la tecla esta alt presionada.

Si se desea conocer la tecla equivalente al código ASCII de la tecla presionada, se debe utilizar un método de la clase String que hace la conversión. Este método se llama se "fromCharCode". La siguiente función es un ejemplo de cómo hacer la conversión:

```
<script language="javascript">
    function getChar(event)
    {
        if (event.keyCode != 0)
            return String.fromCharCode(event.keyCode) ;
        else
            if (event.charCode!=0)
                return String.fromCharCode(event.charCode) ;
            else
                return null;
    }
</script>
```

Otro aspecto que se debe entender es que para rechazar una tecla que el usuario ha presionado, se debe colocar "return false" en el evento onkeypress del input. El siguiente ejemplo rechazará todas las teclas que el usuario presione:

```
<body >
    Ejemplo de validaciones
    <form name="f1" action="" onsubmit="return validar(this)">
        Cedula <input type="text" name="cedula" onkeypress="return false"/>
        Nombre <input type="text" name="nombre" />
        Correo <input type="text" name="correo" />
        <input type="submit" name="btnenviar" value="Enviar"/>
    </form>
</body>
```

12.2.- Validar Teclas

Entendiendo todos estos conceptos sobre los eventos del teclado, se puede realizar una función que valide las teclas que son permitidas para ciertos inputs, dependiendo de la naturaleza del dato. El siguiente ejemplo muestra una función que valida que el usuario sólo escriba letras en un cuadro texto:

```
function validarletra(e)
{
    // para que funcione en cualquier navegador
    tecla = (e.keyCode!=0) ? e.keyCode : e.charCode; ;
    // para permitir backspace
    if (tecla==8) return true;
    // de define el conjunto de caracteres validas
    patron =/[A-Za-z\s]/;
    // se convierte a caracter
    te = String.fromCharCode(tecla);
    // se evalua si la tecla presionada este en el conjunto
    return patron.test(te);
}
```

El siguiente ejemplo es de una función que verifica que la tecla presionada sean números:

```
function validar_numero(e)
{
    // para que funcione en cualquier navegador
    tecla = (e.keyCode!=0) ? e.keyCode : e.charCode;
    // para permitir backspace
    if (tecla==8) return true;
    // de define el conjunto de caracteres validas
    patron = /[0-9-]/;
    // se convierte a caracter
    te = String.fromCharCode(tecla);
    // se evalua si la tecla presionada este en el conjunto
    return patron.test(te);
}
```

En los input se debe llamar en el keypress:

```
<body>
  Ejemplo de validaciones
  <form name="f1" action="" onsubmit="return validar(this)">
    Cedula <input type="text" name="cedula" onkeypress="return validar numero(event)"/>
    Nombre <input type="text" name="nombre" onkeypress="return validarletra(event)"/>
    Correo <input type="text" name="correo" />
    <input type="submit" name="btnenviar" value="Enviar"/>
  </form>
</body>
```



Capítulo 13. EJECUCIÓN AUTOMÁTICA

13.1.- Settimeout

Es muy común necesitar ejecutar código automáticamente al cabo de cierto tiempo, por ejemplo: cerrar una página al pasar cierto tiempo de inactividad (como hacen las páginas de los bancos), mostrar la hora actualizada en algún lugar de la página, entre otras cosas. Para realizar esta tarea se pueden utilizar dos métodos: `setTimeout` y `setInterval`. El método `setTimeout` ejecuta una sola vez una función al transcurrir el tiempo indicado en milisegundos. La forma general del `setTimeout` es la siguiente:

```
window.setTimeout( función, milisegundos);
```

Donde "función" es el nombre de una función que debe estar definida y "milisegundos" es la cantidad de milisegundos que van a transcurrir antes de ejecutarse el código de la función (considere que 1 segundo es igual a 1000 milisegundos). Se puede omitir "window" y se puede ejecutar simplemente el nombre de la función "setTimeout".

En el siguiente ejemplo se muestra el uso del método `setTimeout`, en el cual luego de transcurrir 5 segundos de cargarse la página se mostrará un mensaje:

```
<script language="javascript">

    function saludar()
    {
        alert("Transcurrieron 5 segundos");
    }
</script>
<body onload="window.setTimeout('saludar()',5000);">

</body>
```

Es posible detener la ejecución del `setTimeout` almacenando en una variable el identificador asociado al timer que se programa y luego ejecutando el método "`clearTimeout`" para cancelar o limpiar la ejecución del código. Por ejemplo:

```
<script language="javascript">
    var timer;
    function activar()
    {
        timer=window.setTimeout('saludar()',5000);
        alert("Se activo el reloj");
    }
    function desactivar()
    {
        clearTimeout(timer);
    }
    function saludar()
    {
        alert("Transcurrieron 5 segundos");
    }
</script>
<body>
    <input type="button" value="Activar" onclick="activar()" />
    <input type="button" value="Desactivar" onclick="desactivar()" />
</body>
```

En el ejemplo anterior se realizan las siguientes actividades:

- se declara una variable global con el nombre "timer" para ser usada en varias funciones.
- se programa que al hacer click en un hipervínculo con el texto "Activar" se ejecute la función "activar", que programa con "setInterval" la ejecución de la función "saludar" al transcurrir 5 segundos de que se haga click en el hipervínculo.
- al ejecutar "setInterval" se almacena el resultado en la variable global "timer".
- se programa que al hacer click en un hipervínculo con el texto "Desactivar" se ejecute la función "desactivar" que ejecuta el método "clearTimeout", que usando la variable "timer" elimina la programación hecha con el setInterval (esto tendría sentido sólo si el usuario hace click en el hipervínculo antes de que transcurran los 5 segundos).

13.2.- Setinterval

El funcionamiento del "setInterval" es muy parecido al "setTimeout", con la particularidad de que "setInterval" se ejecuta cada vez que transcurre el tiempo indicado, a diferencia de "setTimeout" que se ejecuta una sola vez al transcurrir el tiempo indicado. El siguiente ejemplo muestra el uso del setInterval:

```
<script language="javascript">

    function saludar()
    {
        alert("Transcurrieron 5 segundos");
    }
</script>
<body onload="window.setInterval('saludar()',5000)">

</body>
```

En la siguiente imagen se muestra un ejemplo de como usar el setInterval, en el que se mostrará un mensaje cada 5 segundos (5 mil milisegundos) con el valor de una variable que se va incrementando cada vez que se ejecuta la función:

```
<script language="javascript">
    var timer;
    var contador=0;
    function activar()
    {
        timer=window.setInterval('saludar()',5000);
        alert("Se activo el reloj");
    }
    function desactivar()
    {
        clearInterval(timer);
    }
    function saludar()
    {
        contador++;
        alert("Se ha ejecutado "+contador+" veces");
    }
</script>
<body >
    <input type="button" value="Activar" onclick="activar()" />
    <input type="button" value="Desactivar" onclick="desactivar()" />
</body>
```