

Nivelación, JavaScript y configuración del entorno

Nivelación HTML, CSS y JS

Presentación:

React JS es una biblioteca para desarrollo web por lo cual debemos aprender conocimientos mínimos sobre los lenguajes que el navegador web interpreta.

HTML, es un lenguaje de etiquetas el cual dará la estructura para nuestras páginas webs.

CSS, veremos la sintaxis básica de CSS con el cual podremos dar estilos (diseño, colores, márgenes) a nuestras webs desarrolladas con HTML

JS, es el lenguaje de programación web por excelencia. React se desarrolló en base a este lenguaje.

Objetivos:

Que los participantes:

- Aprendan a maquetar en HTML y la sintaxis básica de Javascript
- Conozcan cómo dar diseño a sus páginas utilizando CSS
- Comprendan qué es Javascript, el DOM y la importancia de la estructura de una página web

Bloques temáticos:

- ¿Qué es HTML?
- Un nuevo estándar para una nueva Web, HTML 5
- Cuáles son las novedades de HTML 5
- Un poco mas de HTML
- CSS (Hojas de estilo en cascada)

- Clases CSS
- Javascript
- Javascript – DOM
- Javascript – Formularios
- Javascript – Eventos
- Javascript – Seleccionar elemento e imprimir en html
- Ejemplos en Javascript

¿Qué es HTML?



HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del World Wide Web Consortium (W3C) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

Un nuevo estándar para una nueva Web, HTML 5

HTML5 no es simplemente una nueva versión del lenguaje de marcación HTML, sino una agrupación de diversas especificaciones concernientes al desarrollo web. Es decir, HTML 5 no se limita sólo a crear nuevas etiquetas, atributos y eliminar aquellas marcas que están en desuso o se utilizan inadecuadamente, sino que va mucho más allá.

Así pues, HTML 5 es una nueva versión de diversas especificaciones, entre las que se encuentran:

- HTML 4
- XHTML 1



- CSS Nivel 2
- DOM Nivel 2 (DOM = Document Object Model)

A la par, HTML5 pretende proporcionar una plataforma con la que desarrollar aplicaciones web más parecidas a las aplicaciones de escritorio, donde su ejecución dentro de un navegador no implique falta de recursos o facilidades para resolver las necesidades reales de los desarrolladores.

Cuáles son las novedades de HTML 5

HTML 5 incluye novedades significativas en diversos ámbitos. Este nuevo estándar supone mejoras en áreas que hasta ahora quedaban fuera del lenguaje y para las que se necesitaba utilizar otras tecnologías.

- **Estructura del cuerpo:** La mayoría de las webs tienen un formato común, formado por elementos como cabecera, pie, navegadores, etc. HTML 5 permite agrupar todas estas partes de una web en nuevas etiquetas que representarán cada una de las partes típicas de una página.
- **Etiquetas para contenido específico:** Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.
- **Canvas:** es un nuevo componente que permitirá dibujar, por medio de las funciones de un API, en la página todo tipo de formas, que podrán estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece Flash, pero dentro de la especificación del HTML y sin la necesidad de tener instalado ningún plugin.
- **Bases de datos locales:** el navegador permitirá el uso de una base de datos local, con la que se podrá trabajar en una página web por medio del cliente y a través de un API. Es algo así como las Cookies, pero pensadas para almacenar grandes cantidades de información, lo que permitirá la creación de aplicaciones web que funcionen sin necesidad de estar conectados a Internet.
- **Web Workers:** son procesos que requieren bastante tiempo de procesamiento por parte del navegador, pero que se podrán realizar en un segundo plano, para que el usuario no tenga que esperar que se terminen para empezar a usar la página.
- **Aplicaciones web Offline:** Existirá otro API para el trabajo con aplicaciones web, que se podrán desarrollar de modo que funcionen también en local y sin estar conectados a Internet.

- **Geolocalización:** Las páginas web se podrán localizar geográficamente por medio de un API que permita la Geolocalización.

Un poco mas de HTML

Ahora que ya entendemos que es HTML y que es HTML 5, comencemos a ver un poco más de código, que es lo que todos queremos hacer.

Este es un ejemplo de código en HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
</head>
<body>
  <h1>This is a Heading</h1>
  <p>This is a paragraph.</p>
</body>
</html>
```

Si te preguntas ¿De qué me estás hablando? Te respondo, de un lenguaje diferente que tal vez todavía no conoces. Entonces aprendamos a hablar los dos en este lenguaje.

DOCTYPE

```
<!DOCTYPE html>
```

DOCTYPE no es una etiqueta, esta es una instrucción para indicar al navegador que versión de HTML vamos a utilizar. El DOCTYPE mostrado en nuestro ejemplo es del estándar HTML 5.

Por ejemplo este es el DOCTYPE del estándar HTML 4:

```
<!DOCTYPE HTML PUBLIC »"-//W3C//DTD HTML
4.01//ES"»"http://www.w3.org/TR/html4/strict.dtd">
```

Entonces, todo código html debe comenzar con el DOCTYPE

HTML

Representa la raíz de un documento HTML o XHTML. Todos los demás elementos deben ser descendientes de este elemento.

En él podemos incluir el atributo lang, por ejemplo: `<html lang="es">`.

Este atributo sirve para que el navegador identifique el lenguaje en el que está desarrollado el sitio web.

HEAD

Representa una colección de metadatos acerca del documento, incluyendo enlaces a, o definiciones de, scripts y hojas de estilo.

Metadatos del documento

- `<title>` Define el título del documento, el cual se muestra en la barra de título del navegador o en las pestañas de página. Solamente puede contener texto y cualquier otra etiqueta contenida no será interpretada.
- `<base>` Define la URL base para las URLs relativas en la página.
- `<link>` Usada para enlazar JavaScript y CSS externos con el documento HTML actual.

```
<link rel="stylesheet" type="text/css" href="theme.css">
```

- `<meta>` Define los metadatos que no pueden ser definidos usando otro elemento HTML. Por ejemplo `<meta charset="UTF-8">`

Con esta etiqueta meta definimos la codificación que tendrá nuestro archivo, los mismos pueden ser: UTF-8 o ANSI

ANSI es el formato estándar de codificación de archivos utilizado en el Bloc de notas. Se utiliza más comúnmente en archivos que utilizan caracteres del idioma inglés, ya que requieren menos espacio y menos tiempo para procesar.

UTF-8 es capaz de procesar los idiomas que utilizan más caracteres a un ritmo más

rápido y más eficiente. Esto es útil de usar para las lenguas asiáticas y de Oriente Medio, ya que requieren más caracteres y no son capaces de ser procesadas en una tasa eficiente de formato ANSI.

- **`<style>`** Etiqueta de estilo usada para escribir CSS en línea.

```
<style>
  .clase{
    background-color: rgb(255, 255, 255);
  }
</style>
```

Aquí estamos definiendo estilos para unas clases CSS, dándole un color de fondo al elemento.

Scripting

- **`<script>`** Define ya sea un script interno o un enlace hacia un script externo. El lenguaje de programación es JavaScript
- **`<noscript>`** Define un contenido alternativo a mostrar cuando el navegador no soporta scripting. Este elemento es muy importante para la construcción de sitios accesibles, ya que por ejemplo los lectores de pantallas utilizados por los no videntes no reproducen el contenido javascript.

Secciones

- **<body>** Representa el contenido principal de un documento HTML.
Solo hay un elemento **<body>** en un documento.
- **<section>** **HTML5** Define una sección en un documento.
- **<nav>** **HTML5** Define una sección que solamente contiene enlaces de navegación.
Comúnmente utilizado para los menús en los sitios.
- **<h1>, <h2>, <h3>, <h4>, <h5>, <h6>** Los elementos de cabecera implementan seis niveles de cabeceras de documentos; **<h1>** es la de mayor y **<h6>** es la de menor importancia. Un elemento de cabecera describe brevemente el tema de la sección que introduce.

Utilizado para marcar las diferentes secciones de acuerdo a su importancia, que tiene una página web. Por ejemplo en un diario digital el título de la noticia será **<h1>** su volanta será **<h2>** y así hasta llegar al **<h6>** (No es obligatorio utilizar los 6 encabezados)

- **<header>** **HTML 5** Define la cabecera de una página o sección. Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos.
- **<footer>** **HTML 5** Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación.

Formularios

- **<form>** Representa un formulario, consistiendo de controles que puede ser enviado a un servidor para procesamiento.
- **<title>** Representa el título de un control de formulario.
- **<label>** Representa un campo de datos escrito que permite al usuario o usuaria editar los datos.
- **<button>** Representa un botón.

- `<select>` Representa un control que permite la selección entre un conjunto de opciones.
- `<option>` Representa una opción en un elemento `<select>`, o una sugerencia de un elemento `<datalist>`.
- `<textarea>` Representa un control de edición de texto multilínea.

Un pequeño formulario

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Contacto | UTN Desarrollo para móviles</title>
  <meta charset="UTF-8">
</head>
<body>
  <style>
    h1 {color: rgb(200, 200, 200); }
  </style>
  <h1>Contacto</h1>
  <h2>Completa tus datos de contacto</h2>
  <form>
    <div class>
      <label>Nombre</label>
      <input name="nombre" id="nombre" />
    </div>
    <div>
      <label>Apellido</label>
      <input name="apellido" id="apellido" />
    </div>
    <div>
      <label>Email</label>
      <input name="email" id="email" />
    </div>
    <div>
      <label>Curso</label>
      <select>
```

```

    <option>Curso para desarrollo de aplicaciones móviles</option>
    <option>Desarrollo Web</option>
    <option>Project Management</option>
  </select>
</div>
</form>
<h2>Nuestros teléfonos de contacto</h2>
</body>
</html>

```

Y ahora... el resultado

Contacto

Completa tus datos de contacto

Nombre

Apellido

Email

Curso

Nuestros teléfonos de contacto

CSS (Hojas de estilo en cascada)

Hoja de estilo en cascada o CSS (siglas en inglés de cascading style sheets) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML2 (y por extensión en XHTML).

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

La información de estilo puede ser definida en un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales con el elemento `<style>` o en cada etiqueta particular mediante el atributo `<style>`.



Incluir una hoja de estilo en el HTML

Como se mencionó anteriormente para incluir una hoja de estilo en nuestro documento HTML debemos utilizar el elemento `<link>`.

```
<link ref="stylesheet" type="text/css" href="theme.css">
```

Dentro del archivo CSS externo incluiremos las sentencias correspondientes, como lo hacemos con un archivo css en línea. Por ejemplo:

```
.img-responsive {  
  max-width:100%;  
  height:auto;  
}  
.table-responsive {  
  overflow-x: auto;  
}  
a:hover, a:active {  
  color: #6E9A06;  
}  
table.lamp, table.w3-table-all {  
  margin: 16px 0;
```

```
}  
/*OPPSETT AV TOP, TOPNAV, SIDEMENU, MAIN, RIGHT OG FOOTER:*/  
.top {  
  position: relative;  
  background-color: #ffffff;  
  height: 68px;  
  padding-top: 20px;  
  line-height: 50px;  
  overflow: hidden;  
  z-index: 1;  
}
```

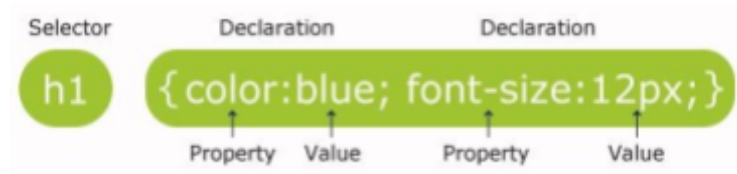
Clases CSS

Podemos hacer referencia a un elemento html a través de una clase o de su id en CSS. Para hacer referencia al objeto a través de una clase lo hacemos a través de un . **(Punto)** en cambio por el id lo hacemos a través de # **(numeral)**.

```
.ejemplo_clase{  
  color: rgb(255, 255, 255);  
}  
#ejemplo_id{  
  color: rgb(255, 255, 255);  
}
```

También podemos hacer referencia a través del elemento directamente, por ejemplo aplicar un estilo a todos los inputs del documento.

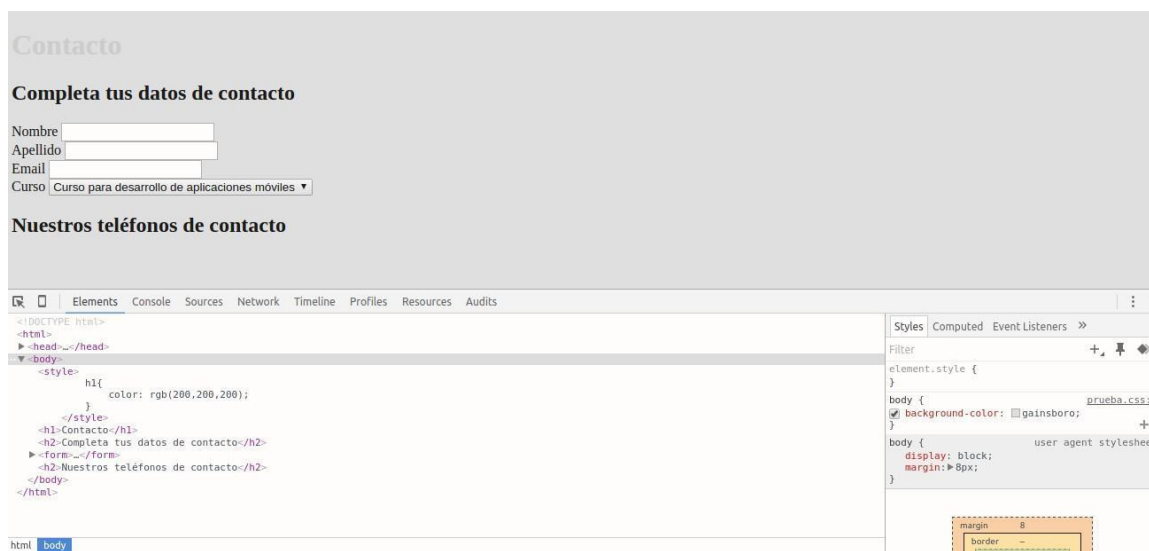
```
input {  
  color: rgb (255, 255, 255)  
}
```



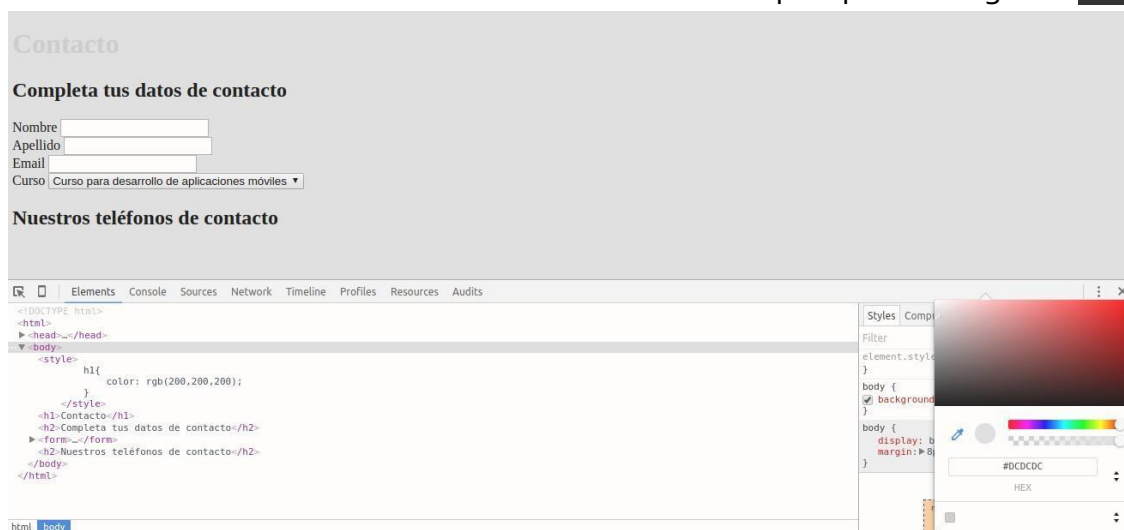
Si quieres ver las propiedades que brinda css puedes ingresar a <http://www.w3schools.com/css>

Utiliza el inspector de elementos

Para facilitar tu tarea a la hora de maquetar es importante que aprendas a utilizar el inspector de elementos. Para ello debes apretar F12 o hacer clic derecho sobre el navegador y luego inspeccionar elemento. Por ejemplo con el inspector de elemento podrás modificar el **background-color** de un elemento y ver cómo queda sin necesidad de recargar la página



Debes hacer click sobre la columna derecha en el color que aparece luego del **background-color**.



Para ver cómo utilizar el inspector de elementos en firefox ingresa aquí:
https://developer.mozilla.org/es/docs/Tools/Page_Inspector

Sigamos con nuestro formulario de ejemplo

Este es nuestro formulario de ejemplo



A screenshot of a web form titled 'Contacto' in a light gray font. Below the title is a section header 'Completa tus datos de contacto'. The form contains four input fields: 'Nombre', 'Apellido', 'Email', and 'Curso'. The 'Curso' field is a dropdown menu with the text 'Curso para desarrollo de aplicaciones móviles' and a downward arrow. Below the form fields is another section header 'Nuestros teléfonos de contacto'.

Muy bien ahora vamos a aplicarle un poco de estilos CSS. Y así quedó...



A screenshot of the same web form as before, but with CSS styling applied. The title 'Contacto' is now in a bold red font. The section header 'Completa tus datos de contacto' is in a bold black font. The input fields for 'Nombre', 'Apellido', and 'Email' are white with gray borders. The 'Curso' dropdown menu is also white with a gray border. The section header 'Nuestros teléfonos de contacto' is in a bold black font. The background of the form is a solid light gray.

JavaScript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario

Tipos de ejecución

Ejecución directa: Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta `<SCRIPT>`, cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra.

Respuesta a un evento: Los eventos son acciones que realiza el usuario. Los programas como Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta. Por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Versiones

Año	Nombre	Descripción
1997	ECMAScript 1	Primer edición
1998	ECMAScript 2	Editorial con algunos cambios
1999	ECMAScript 3	Añadió try/catch y expresiones regulares
	ECMAScript 4	Nunca se puso en producción
2009	ECMAScript 5	Añadió soporte a JSON
2011	ECMAScript 5.1	Cambios editoriales
2015	ECMAScript 6	Añadió clases y módulos
2016	ECMAScript 7	Añadió operador exponencial

Incluir el archivo javascript en nuestro HTML

```
<html>
  <head>
    <title>Clase 1 react</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>

  </body>
  <script src="script.js"></script>
</html>
```

Esto debe ser incluido en el `head` del html o bien al finalizar el `body`, de esta última forma nos aseguramos que todos los elementos estén cargados al ejecutar el script.

También podemos introducir nuestro código javascript dentro de la etiqueta

`<script></script>`. Por ejemplo:

```
<script>
  function mi_funcion() {
    document.getElementById("mi_funcion").innerHTML = "Ejemplo Mi funcion";
  }
</script>
```

Modo estricto

El modo estricto de ECMAScript 5 es una forma de elegir una variante *restringida* de *JavaScript*, así implícitamente se deja de lado el modo poco riguroso.

El modo estricto no es sólo un subconjunto: *intencionalmente* tiene diferencia semántica del código normal.

Los navegadores que no admiten el modo estricto ejecutarán el código con un comportamiento diferente a los que sí lo soportan, por lo tanto no confíes en el modo estricto sin antes hacer pruebas de sus características más relevantes.

Los modos estricto y no estricto pueden coexistir, por lo tanto el código se puede transformar a modo estricto incrementalmente.

El modo estricto tiene varios cambios en la semántica normal de JavaScript:

1. Elimina algunos errores silenciosos de JavaScript cambiándolos para que lancen errores.
2. Corrige errores que hacen difícil para los motores de JavaScript realizar optimizaciones: a veces, el código en modo estricto puede correr más rápido que un código idéntico pero no estricto.
3. Prohíbe cierta sintaxis que probablemente sea definida en futuras versiones de ECMAScript.

Cambiar algunos errores sintácticos tolerados sin su uso

1. Hace imposible crear variables globales por accidente.

En JavaScript no estricto, si se escribe mal una variable en una asignación, se creará una nueva propiedad en el objeto global y el código continuará "trabajando" como si nada (aunque es posible que el código así escrito falle en el futuro, en concreto, en JavaScript moderno).

En modo estricto, cualquier asignación que produzca variables globales por accidente lanzará un error:

```
'use strict';
mistypeVariable = 17;
// Asumiendo que exista una variable global llamada mistypedVariable
// esta línea lanza un ReferenceError debido a
// una errata en el nombre de la variable
```

2. Lanza una excepción en asignaciones que de otro modo fallarían silenciosamente. Por ejemplo, `NaN` es una variable global que no puede ser asignada.

En un código normal, asignar a **NaN** no tiene efecto; el programador no recibe ningún mensaje de error. Lanza una excepción al intentar eliminar propiedades no eliminables (mientras que en código normal el intento no tendría ningún efecto):

```
'use strict';  
delete Object.prototype; // lanza un TypeError
```

3. Todas las propiedades nombradas en un objeto son únicas. En código normal se pueden duplicar nombres, siendo el último el que determina el valor de la propiedad.

```
'use strict';  
var o = { p: 1, p: 2 }; // !!! error de sintaxis
```

4. Requiere que los nombres de los parámetros de una función sean únicos. En código normal, el último argumento repetido oculta argumentos anteriores con el mismo nombre.

```
function sum(a, a, c) { // !!! error de sintaxis  
'use strict';  
return a + a + c; // incorrecto si este código se ejecutó
```

5. Prohíbe establecer propiedades en valores primitivos.

```
(function () {  
  'use strict';  
  false.true = ''; // TypeError  
  (14).sailing = 'home'; // TypeError  
  'with'.you = 'far away'; // TypeError  
})();
```

Comentarios

Los comentarios puede realizarse de una línea completa `(//)` o bien indicando la apertura y cierre del mismo con `/* ... */`

```
<script>
//Este es un comentario de una línea
/*Este comentario se puede extender
por varias líneas.
Las que quieras*/
</script>
```

También podemos desde nuestro HTML incluir un archivo con extensión .js de la siguiente manera:

```
<script src="/carrito_compra/assets/bootstrap/js/bootstrap.min.js"></script>
```

En este caso el archivo js no tendrá embebidas las etiquetas `<script>` sino que contendrá directamente el código programado bajo la sintaxis de javascript.

Propuesta:

- Crea un archivo .html y embebe código Javascript dentro. Ejecute dicho archivo en el navegador (haciendo doble clic en el)
- Quita el código embebido del html, crea un archivo js con el mismo código js e incluirlo en el html.
- Abrió el navegador, apreté la tecla f12, hace clic en la pestaña consola y ejecuta el mismo código js. ¿Para qué crees que es útil esta herramienta? El código js que puedes utilizar es el siguiente:

```
alert("Hola Mundo");
```

Separación de instrucciones

Javascript tiene dos maneras de separar instrucciones:

- La primera es a través del carácter punto y coma (;)

```
llamadoFuncion();
```

- La segunda es a través de un salto de línea.

```
llamadoFuncion()
```

Variables

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_).

Los nombres tienen que comenzar por un carácter alfabético o el subrayado. **No podemos utilizar caracteres raros como el signo +, un espacio.**

Las variables no pueden utilizar nombres reservados, por ejemplo if, for, while, etc

Declaración de Variables

Declarar variables consiste en definir y de paso informar al sistema de qué vas a utilizar una variable.

Javascript cuenta con la palabra "**var**" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla. Ejemplo:

```
var operando1;  
var operando2;  
var operando1 = 23;  
var operando2 = 33;  
var operando1, operando2;
```

Ejemplo:

```
<script language="javascript">  
  //creo una variable  
  var x;  
  //x actualmente no tiene ningún valor  
  X = 25;  
  //ahora x tiene el valor numérico 25  
  //creó una segunda variable  
  var y = 230;  
  //he creado una variable y asignado un valor en un solo paso!!  
  //las variables guardan datos con los que puedo realizar operaciones  
  var suma;  
  suma = x + y;  
  //he creado la variable suma y he asignado la suma de x e y  
  alert(suma);  
  //he mostrado el valor de la variable suma  
</script>
```

Nota: Si nosotros no declaramos el **modo estricto**, podremos utilizar variables sin necesidad de declararlas con la palabra reservada **var**.

Para declarar el **strict mode** debemos anteponer al código “**strict mode**” por ejemplo:

```
v = 15
function f1() {
  "use strict";
  var v = "Hi! I'm a strict mode script!";
}
```

En este caso la primera asignación se hace sin modo estricto por lo cual como vemos no tiene la palabra var adelante. En cambio dentro de la función f1 se hace en modo estricto por lo cual debe tener la palabra var delante.

Ámbito de las Variables

Se le llama ámbito de las variables al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado.

- **Variables globales:** son las que están declaradas en el ámbito más amplio posible.

```
<script>
var variableGlobal
</script>
```

- **Variables locales:** sólo podremos acceder a ellas dentro del lugar donde se ha declarado

```
<script>
function miFuncion () {
  var variableLocal
}</script>
```

En este caso la variable “variableGlobal” se podrá acceder desde cualquier script en ejecución o a través de scripts embebidos.

En el caso de “variableLocal” sólo se accederá desde la función miFuncion.

NO se recomienda el uso de variables locales.

Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque Javascript sólo tiene un operador para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

Para concatenar 2 cadenas de texto debemos usar el operador “+”

```
<script>  
cadenaConcatenada = cadena1 + cadena2  
</script>
```

El operador + también es utilizado para la suma. En caso de querer sumar dos valores asegurarse que ambas variables tienen un tipo de dato **Number** para que el motor infiera que se quiere realizar una suma y no una concatenación. Para esto se puede utilizar la función **parseInt** o **parseFloat**, por ejemplo:

```
var resultado = parseInt (operador1) + parseInt (operador2)
```

De esta manera nos aseguramos que las variables **operador1** y **operador2** se tomen como **number**, se sumen y no se concatenen.

Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts.

```
<script>
prueba && prueba2 //Operador and
prueba || prueba2 //Operador or
</script>
```

Operador and: Retorna true si todos los operandos son true

Operador or: Retorna true si al menos un operador retorna true

Typeof

Para comprobar el tipo de un dato se puede utilizar otro operador que está disponible a partir de Javascript 1.1, el **operador `typeof`**, que devuelve una cadena de texto que describe el tipo del operador que estamos comprobando.

```
<script>
document.write("<br>El tipo de booleano es: " + typeof booleano)
</script>
El tipo de booleano es: boolean
```


Operadores de asignación

Un operador de asignación asigna un valor al operando de la izquierda en función del valor del operando de la derecha.

El operador básico de asignación es el de igual (`=`), que asigna el valor del operando de la derecha al operando de la izquierda.

Por ejemplo, `x = y`, está asignando el valor de `y` a `x`.

Nombre	Operador abreviado	Significado
Operadores de asignación	<code>x = y</code>	<code>x = y</code>
Asignación de adición	<code>x += y</code>	<code>x = x + y</code>
Asignación de sustracción	<code>x -= y</code>	<code>x = x - y</code>
Asignación de multiplicación	<code>x *= y</code>	<code>x = x * y</code>
Asignación de división	<code>x /= y</code>	<code>x = x / y</code>
Asignación de resto	<code>x %= y</code>	<code>x = x % y</code>
Asignación de exponenciación	<code>x **= y</code>	<code>x = x ** y</code>
Asignación de desplazamiento a la izquierda	<code>x <<= y</code>	<code>x = x << y</code>
Asignación de desplazamiento a la derecha	<code>x >>= y</code>	<code>x = x >> y</code>
Asignación de desplazamiento a la derecha sin signo	<code>x >>>= y</code>	<code>x = x >>> y</code>
Asignación AND binaria	<code>x &= y</code>	<code>x = x & y</code>
Asignación XOR binaria	<code>x ^= y</code>	<code>x = x ^ y</code>
Asignación OR binaria	<code>x = y</code>	<code>x = x y</code>

Operadores de comparación

Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (true) o falsa (false).

Los operadores pueden ser numéricos, de cadena de caracteres (Strings), lógicos o de objetos.


Operador	Descripción	Ejemplos devolviendo true
Igualdad (==)	Devuelve true si ambos operandos son iguales.	<pre>3 == var1 "3" == var1 3 == "3"</pre>
Desigualdad (!=)	Devuelve true si ambos operandos no son iguales.	<pre>var1 != 4 var2 != "3"</pre>
Estrictamente iguales (===)	Devuelve true si los operandos son igual y tienen el mismo tipo. Mira también Object.is y sameness in JS .	<pre>3 === var1</pre>
Estrictamente desiguales (!==)	Devuelve true si los operandos no son iguales y/o no son del mismo tipo.	<pre>var1 !== "3" 3 !== "3"</pre>
Mayor que (>)	Devuelve true si el operando de la izquierda es mayor que el operando de la derecha.	<pre>var2 > var1 "12" > 2</pre>
Mayor o igual que (>=)	Devuelve true si el operando de la izquierda es mayor o igual que el operando de la derecha.	<pre>var2 >= var1 var1 >= 3</pre>
Menor que (<)	Devuelve true si el operando de la izquierda es menor que el operando de la derecha.	<pre>var1 < var2 "2" < 12</pre>
Menor o igual que (<=)	Devuelve true si el operando de la izquierda es menor o igual que el operando de la derecha.	<pre>var1 <= var2 var2 <= 5</pre>

Operadores aritméticos

Los operadores aritméticos toman los valores numéricos (tanto literales como variables) de sus operandos y devuelven un único resultado numérico.

Los operadores aritméticos estándar son la suma (+), la resta (-), la multiplicación (*) y la división (/).

Tabla 3.3 Operadores aritméticos

Operador	Descripción	Ejemplo
Resto (%)	Operador binario correspondiente al módulo de una operación. Devuelve el resto de la división de dos operandos.	12 % 5 devuelve 2.
Incremento (++)	Operador unario. Incrementa en una unidad al operando. Si es usado antes del operando (++x) devuelve el valor del operando después de añadirle 1 y si se usa después del operando (x++) devuelve el valor de este antes de añadirle 1.	Si x es 3, entonces ++x establece x a 4 y devuelve 4, mientras que x++ devuelve 3 y, solo después de devolver el valor, establece x a 4.
Decremento (--)	Operador unario. Resta una unidad al operando. Dependiendo de la posición con respecto al operando tiene el mismo comportamiento que el operador de incremento.	Si x es 3, entonces --x establece x a 2 y devuelve 2, mientras que x-- devuelve 3 y, solo después de devolver el valor, establece x a 2.
Negación Unaria (-)	Operación unaria. Intenta convertir a número al operando y devuelve su forma negativa.	- "3" devuelve -3. -true devuelve -1.
Unario positivo (+)	Operación unaria. Intenta convertir a número al operando.	+ "3" devuelve 3. +true devuelve 1.
Exponenciación (**) 	Calcula la potencia de la base al valor del exponente. Es equivalente a $base^{exponente}$	2 ** 3 devuelve 8. 10 ** -1 devuelve 0.1.

Funciones

Primero se escribe la palabra `function`, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guión bajo.

```
<script>
  function nombrefuncion() {
    instrucciones de la función
    ...
  }
</script>
```

Entre paréntesis, luego del nombre de la función, se definirán los parámetros que recibirá la misma. En el caso de js no hace falta el tipo de dato en cada parámetro recibido.

Arrays

Los arrays son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un array son variables.

Dicho en otras palabras, imaginemos que en una variable podemos guardar un solo elemento (nuestro teléfono). Bien un array seria como un organizador con varios bloques en el cual podemos almacenar un elemento por bloque (ejemplo teléfono en un bloque, auriculares en otro, la funda en el tercero, etc)

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador.

```
<script>
  //Array vacío
  var miArray = new Array()
  //Creación de array con número de compartimentos
  var miArray = new Array(10)
```

```
//Setear datos
miArray[0] = 290
miArray[1] = 97
miArray[2] = 127
//Declaracion e inicializacion var
arrayRapido = [12, 45, "array inicializado en su declaración"]
</script>
```

A diferencia de PHP no podemos tener vectores asociativos, es decir arrays cuya clave sea un string.

Longitud de un array

Para obtener la longitud de un array utilizaremos el método `length`

```
<script>
  document.write("Longitud del array: " + miArray.length)
</script>
```

For

La estructura `for` nos permitirá recorrer un array en Javascript.

Cada elemento recorrido se reconoce como una iteración, se “cortara” el for cuando la condición sea falsa.

```
<script>
  for (inicialización; condición; actualización) {
    //sentencias a ejecutar en cada iteración
  }
</script>
```

- **Inicialización:** Aquí inicializamos las variables. Por ejemplo: `i=0`

- **Condición:** Mientras la condición sea verdadera se seguirán produciendo iteraciones. Ejemplo `i<5`
- **Actualización:** se actualiza la variable de la condición. Ejemplo: `i++`

While

La estructura `while` es similar a la `for`, nos permite realizar iteraciones sobre un array

```
<script>
while (condición) {
  //sentencias a ejecutar
}
</script>
```

Como vemos no cuenta con el elemento de inicialización ni el de actualización.

Condicional IF

La estructura `if` nos permitirá mediante el uso de operadores de comparación tomar una decisión a partir de si dicha comparación o valor de una expresión es verdadera o falsa (en este caso entra por el `else`)

```
<script>
if (expresión) {
  //acciones a realizar en caso positivo
  //...
} else {
  //acciones a realizar en caso negativo
  //...
}
</script>
```

Javascript – DOM

Definición

DOM (Document Object Model o modelo de objetos del navegador) que nos sirve para acceder a cualquiera de los componentes que hay dentro de una página. Por medio del DOM podremos controlar al navegador en general y a los distintos elementos que se encuentran en la página. **Jerarquía del DOM:**



Window

Todos los objetos comienzan en un objeto que se llama **window**. Este objeto ofrece una serie de métodos y propiedades para controlar la ventana del navegador. Con ellos podemos controlar el aspecto de la ventana, la barra de estado, abrir ventanas secundarias y otras cosas que veremos más adelante cuando expliquemos con detalle el objeto.

Además de ofrecer control, el objeto **window** da acceso a otros objetos como el documento (La página web que se está visualizando), el historial de páginas visitadas o los distintos frames de la ventana. De modo que para acceder a cualquier otro objeto de la jerarquía deberíamos empezar por el objeto window. Tanto es así que Javascript entiende perfectamente que la jerarquía empieza en window aunque no lo señalemos.

```
<script>
  window.document.bgColor = "red"
  window.document.write("El texto a escribir")
</script>
```

Propiedades:

- **document**: Objeto que contiene el la página web que se está mostrando.
- **frame**: Un objeto frame de una página web. Se accede por su nombre.
- **history**: Objeto historial de páginas visitadas.
- **location**: La URL del documento que se está visualizando. Podemos cambiar el valor de esta propiedad para movernos a otra página.
- **name**: Nombre de la ventana. Lo asignamos cuando abrimos una nueva ventana.

Métodos:

- **alert(texto)**: Presenta una ventana de alerta donde se puede leer el texto que recibe por parámetro
- **blur()**: Quitar el foco de la ventana actual
- **close()**: Cierra la ventana.
- **forward()**: Ir una página adelante en el historial de páginas visitadas.
- **open()**: Abre una ventana secundaria del navegador.

Document

Se trata de las propiedades que son arrays, por ejemplo la propiedad `images` es un array con todas las imágenes de la página web. También encontramos arrays para guardar los enlaces de la página, los applets, los formularios y las anclas.

Cuando una página se carga, el navegador construye en memoria la jerarquía de objetos. De manera adicional, construye también estos arrays de objetos.

```
<script>
  for (i = 0; i < document.images.length; i++) {
    document.write(document.images[i].src)
    document.write("<br>")
  }
</script>
```

Propiedades:

- `bgColor`: El color de fondo del documento.
- `cookie`: Accede a una cookie del navegador
- `domain`: Nombre del dominio del servidor de la página.
- `fgColor`: El color del texto. Para ver los cambios hay que recargar la página.
- `ids`: Para acceder a estilos CSS.
- `title`: El título de la página. Métodos:
- `close()`: Cierra el flujo del documento.
- `open()`: Abre el flujo del documento.
- `write()`: Escribe dentro de la página web. Podemos escribir etiquetas HTML y texto normal.
- `writeln()`: Escribe igual que el método `write()`, aunque coloca un salto de línea al final.

Formularios

Acceder a formularios con document

Acceder al formulario a través del objeto document y el nombre del formulario.

```
> document.form_registrar  
<- <form method="POST" name="form_registrar" class="regform">...</form>
```

Acceder al formulario a través del objeto document y el índice del formulario.

```
> document.forms[0]  
<- <form method="POST" name="form_registrar" class="regform">...</form>
```

Acceder a campos dentro del formulario

```
document.form_registrar.nombre  
<input name= "nombre" type="text" class="form-control" aria-required="true">
```

Acceder al valor del campo

```
document.form_registrar.nombre.value  
" "
```

Evento onclick

Con este evento definimos una acción (función) a ejecutar cuando se hace click en un elemento (un botón)

```
<input type="Button" name="enviar" value=" enviar " onclick="enviar()">
```

JavaScript – Propiedades

- **Action**: Es la acción que queremos realizar cuando se submite un formulario.
- **Encoding**: El tipo de codificación del formulario
- **Length**: El número de campos del formulario.
- **Method**: El método por el que mandamos la información. Corresponde con el atributo METHOD del formulario.
- **Name**: El nombre del formulario, que corresponde con el atributo NAME del formulario.
- **Target**: La ventana o frame en la que está dirigido el formulario. Cuando se submite se actualizará la ventana o frame indicado. Corresponde con el atributo target del formulario.

Ejemplo de propiedad method:

```
document.form_registrar.method  
"post"
```

Métodos

- **submit()**: Para hacer que el formulario se submite, aunque no se haya pulsado el botón de submit.
- **reset()**: Para reiniciar todos los campos del formulario, como si se hubiese pulsado el botón de reset.

Campos de texto

Propiedades

- **defaultValue**: Es el valor por defecto que tiene un campo. Lo que contiene el atributo VALUE de la etiqueta <INPUT>.
- **Form**: Hace referencia al formulario.
- **Name**: Contiene el nombre de este campo de formulario
- **Type**: Contiene el tipo de campo de formulario que es.
- **Value**: El texto que hay escrito en el campo.

Métodos

- **blur()**: Retira el foco de la aplicación del campo de texto.
- **focus()**: Pone el foco de la aplicación en el campo de texto.
- **select()**: Selecciona el texto del campo.

Javascript – Checkbox

Propiedades

- **checked**: Informa sobre el estado del checkbox. Puede ser true o false.
- **defaultChecked**: Si está chequeada por defecto o no.
- **Value**: El valor actual del checkbox.

Métodos

- **click()**: Es como si hiciésemos un click sobre el checkbox, es decir, cambia el estado del checkbox.
- **blur()**: Retira el foco de la aplicación del checkbox.
- **focus()**: Coloca el foco de la aplicación en el checkbox.

Javascript – Select

Propiedades

- **length**: Guarda la cantidad de opciones del campo select. Cantidad de etiquetas `<OPTION>`
- **Option**: Hace referencia a cada una de sus opciones. Son por sí mismas objetos.
- **Options**: Un array con cada una de las opciones del select.
- **selectedIndex**: Es el índice de la opción que se encuentra seleccionada.

Métodos

- **blur()**: Para retirar el foco de la aplicación de ese elemento de formulario.
- **focus()**: Para poner el foco de la aplicación.
- Objeto **option**:
 - **defaultSelected** Indica con un true o un false si esa opción es la opción por defecto.
 - **index** El índice de esa opción dentro del select.
 - **selected** Indica si esa opción se encuentra seleccionada o no.
 - **text** Es el texto de la opción. Lo que puede ver el usuario en el select, que se escribe después de la etiqueta `<OPTION>`.
 - **value** Indica el valor de la opción, que se introduce con el atributo `VALUE` de la etiqueta `<OPTION>`.

Eventos

Manejadores

- **onblur** Se desata un evento `onblur` cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento.

- **onchange** Se desata este evento cuando cambia el estado de un elemento de formulario, en ocasiones no se produce hasta que el usuario retira el foco de la aplicación del elemento.
- **onclick** Se produce cuando se da una pulsación o clic al botón del ratón sobre un elemento de la página, generalmente un botón o un enlace.
- **onload** Este evento se desata cuando la página, o en Javascript 1.1 las imágenes, ha terminado de cargarse.
- **onsubmit** Ocurre cuando el visitante apreta sobre el botón de enviar el formulario. Se ejecuta antes del envío propiamente dicho.

Para ver más manejadores: <http://www.desarrolloweb.com/articulos/1236.php>

Ejemplo de on blur:

```
</head>

<body>
  <form name=f1>
    Escriba un número entero: <input type=text name=numero size=3 value=""
onblur="compruebaValidoEntero()">
  </form>
</body>

</html>
```

Javascript – Seleccionar elemento e imprimir en html

getElementById

Permite, como su nombre indica, seleccionar un elemento del documento por medio del valor del atributo id que se le haya asignado

```
<script>
document.getElementById('id_del_elemento');
</script>
```

InnerHTML

Retorna o setea el contenido HTML de un elemento

- **Set** (Modifica el HTML para mostrar el valor en pantalla)

```
document.getElementById("alertas").innerHTML += "El campo nombre es  
obligatorio<br/>";
```

- **Get** (Obtiene el valor que tiene el elemento en pantalla)

```
document.getElementById("alertas").innerHTML;
```

Cuando el elemento es de formulario (input, select, textarea, etc) obtenemos o modificamos el valor del mismo con la propiedad value.

Cuando el elemento es HTML (div, span, label, etc) obtenemos o modificamos el valor del mismo con la propiedad innerHTML

Ejemplos en Javascript

Validar un entero en Javascript

Creamos un archivo llamado "script.js" el cual contiene la siguiente función:

```
function validarEnteroEnCampo(identificadorDelCampo) {  
  var field = document.getElementById(identificadorDelCampo);  
  var valueInt = parseInt(field.value);  
  if (!Number.isInteger(valueInt)) {  
    alert("No es un entero")  
  } else {  
    field.value = valueInt;  
  }  
}
```

parseInt: Parsea el valor que pasamos como parámetro a un entero. Es decir, js por ejemplo tomará los valores que introducimos en un input como string, aplicando parseInt transforma los mismos a un tipo de dato Number.

Number.isInteger: Retorna true si el valor que pasamos como parámetro es un entero y false si no lo es.

```
<html>  
<head>  
  <title>Validar entero</title>  
  <script src="script.js"></script>  
</head>  
<body>  
  <form name=formul>  
    <input type=text name=texto id="texto">  
    <input type=button value=validar onclick="validarEnteroEnCampo('texto')">  
  </form>  
</body>  
</html>
```


Comprobar si dos claves son iguales

Creamos un archivo llamado “script.js” el cual contiene la siguiente función:

```
function comprobarclave() {  
  clave1 = document.f1.clave1.value  
  clave2 = document.f1.clave2.value  
  if (clave1 == clave2)  
    alert("Las dos claves son iguales... \n Realizaríamos las acciones del caso positivo")  
  else  
    alert("Las dos claves son distintas... \n Realizaríamos las acciones del caso negativo")  
}
```

Como vemos con la utilización del DOM podemos acceder al elemento y su propiedad value con lo cual obtenemos lo que el usuario escribió en ambos campos.

```
<html>  
<head>  
  <title>Comprobar Claves</title>  
  <script src="script.js"></script>  
</head>  
<body>  
  <form action=" " name="f1">  
    Contraseña: <input type="password" name="clave1" size="20">  
    <br>  
    Repite contraseña: <input type="password" name="clave2" size="20">  
    <br>  
    <input type="button" value="Comprobar si son iguales" onClick="comprobarClave()">  
  </form>  
</body>  
</html>
```

Del lado del HTML se resume a implementar un formulario y la captura del evento onclick

Inhibir un campo de texto

El concepto focus, está relacionado con ganar foco de la aplicación. El método `focus()`, que tienen los campos de texto y otros elementos de formulario, sirve para otorgar el foco de la aplicación a ese elemento. El manejador de evento onfocus salta cuando un elemento gana el foco de la aplicación.

El concepto blur, está asociado a perder el foco de la aplicación. El método `blur()` sirve para que los elementos de formulario pierdan el foco y el manejador de eventos onblur se activa cuando el elemento al que lo apliquemos pierda el foco de la aplicación.

Nosotros utilizaremos el evento onfocus para detectar el instante en el que el elemento gana el foco y en ese momento haremos uso del método `blur()` para retirar el foco.

```
<html>

<head>
  <title>Inhibir campo de texto</title>
  <script src="script.js"></script>
</head>

<body>
  <form>

    <input type="text" value="122" onfocus="this.blur()">
  </form>
</body>
</html>
```

El único detalle que merece la pena señalar es el uso de la palabra `this`, que hace referencia al elemento donde se está utilizando, en ese caso el campo de texto. `this.blur()` sería una simple llamada al método `blur()` en el elemento de formulario donde está colocada.

Contar caracteres en un textarea

Creamos un archivo llamado “script.js” el cual contiene la siguiente función:

```
function cuenta() {  
  document.forms[0].caracteres.value = document.forms[0] texto.value.length  
}
```

En este caso accedemos al formulario de índice 0 en la colección de formularios de document (en este caso solo tenemos un formulario, pero si tuviésemos más en el HTML accederíamos al primero). Dentro del formulario actualizamos el value del elemento “caracteres” (input) con la cantidad de caracteres que presente el elemento “texto” (textarea).

Length: Retorna la cantidad de elementos de un array o la cantidad de caracteres de un campo de texto HTML:

```
<html>  
<head>  
  <title>Contar Caracteres</title>  
  <script src="script.js"></script>  
</head>  
<body>  
  <form action="#" method="post">  
    <table>  
      <tr>  
        <td>Texto:</td>  
        <td>  
          <textarea cols="40" rows="5" name="texto" onkeydown="cuenta()" onkeyup="cuenta()"></textarea>  
        </td>  
      </tr>  
      <tr>  
        <td>Caracteres:</td>  
        <td><input type="text" name="caracteres" size=4></td>  
      </tr>  
    </table>  
  </form>  
</body>  
</html>
```

onKeyDown: Captura la escritura de cualquier carácter mediante el teclado al momento de ser presionada la tecla

onKeyUp: Captura la escritura de cualquier carácter mediante el teclado al momento de ser soltada la tecla.

Reloj

Desarrollemos un pequeño “reloj” que muestre la hora actual por pantalla utilizando javascript

La hora es

11 : 45 : 45

Creamos un archivo llamado “script.js” el cual contiene la siguiente función:

```
function mueveReloj() {  
  var momentoActual = new Date()  
  var hora = momentoActual.getHours()  
  var minuto = momentoActual.getMinutes()  
  var segundo = momentoActual.getSeconds()  
  var str_segundo = new String(segundo)  
  if (str_segundo.length == 1)  
    segundo = "0" + segundo  
  var str_minuto = new String(minuto)  
  if (str_minuto.length == 1)  
    minuto = "0" + minuto  
  str_hora = new String(hora)  
  if (str_hora.length == 1)  
    hora = "0" + hora  
  var horaImprimible = hora + ":" + minuto + ":" + segundo  
  document.form_reloj.reloj.value = horaImprimible  
  setTimeout("muevaReloj()",1000)  
}
```

Asignamos `var momentoActual = new Date()`

`new Date()` al no recibir parámetro en su constructor nos retorna la fecha actual

```
var hora = momentoActual.getHours()
var minuto = momentoActual.getMinutes()
var segundo = momentoActual.getSeconds()
```

Con el objeto resultante de `Date()` obtenemos la hora, minutos y segundos

```
var str_segundo = new String(segundo)
if (str_segundo.length == 1)
    segundo = "0" + segundo
var str_minuto = new String(minuto)
if (str_minuto.length == 1)
    minuto = "0" + minuto
str_hora = new String(hora)
if (str_hora.length == 1)
    hora = "0" + hora
```

Verificamos el formato y agregamos un 0 a la izquierda en caso de que la hora, minutos o segundos tenga una longitud de 1

```
var horalmpresionable = hora + ":" + minuto + ":" + segundo
document.form_reloj.reloj.value = horalmpresionable    setTimeout("mueveReloj()",1000)
```

Armamos el string con el formato de la hora, incluyendo los dos puntos. Seteamos el value del input correspondiente y actualizamos cada 1 segundo utilizando la función `setTimeout`

En el HTML tenemos lo siguiente:

```
<html>
<head>
  <title>Reloj con Javascript</title>
  <script src="script.js"></script>
</head>
<body onLoad="mueveReloj()">
  La hora es
  <form name="form_reloj">
    <input type="text" name="reloj" size="10">
  </form>
</body>
</html>
```

onLoad: Esto quiere decir que cuando se termine de cargar la página se llame a la función **mueveReloj()**, que se encargará de mover el reloj y llamarse a sí misma para hacer el proceso de manera continuada.

Bibliografía y Webgrafía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.

<https://es.wikipedia.org/wiki/HTML>

<http://www.w3schools.com/html/>

http://www.w3schools.com/html/html5_intro.asp

<http://www.w3schools.com/css>

https://developer.mozilla.org/es/docs/Tools/Page_Inspector

<http://www.w3schools.com/js/>