



# **Desarrollo de Entornos Web**

---



# Índice

Presentación	5
Red de contenidos	7
<b>Unidad de Aprendizaje 1</b>	
<b>EL MODELO DE OBJETO DE DOCUMENTO (DOM)</b>	<b>9</b>
<b>1.1 Tema 1 : El DOM</b>	<b>11</b>
1.1.1 : Definición	11
1.1.2 : Accediendo a los elementos	17
1.1.3 : Manipulando los estilos de los documentos	22
<b>Unidad de Aprendizaje 2</b>	
<b>PROGRAMACIÓN BÁSICA EN JAVASCRIPT</b>	<b>34</b>
<b>2.1 Tema 2 : Fundamentos de JavaScript</b>	
2.1.1 : Elementos de un programa en javascript	36
2.1.2 : Entradas y salidas: alert, input	36
2.1.3 : Estructuras de selección	39
<b>2.2 Tema 3 : Estructuras de repetición</b>	
2.2.1 : Sintaxis	46
2.2.2 : Ejemplos aplicados al desarrollo web	50
<b>2.3 Tema 4 : Funciones en el lenguaje JavaScript</b>	
2.3.1 : Introducción	55
2.3.2 : Sintaxis general	<b>55</b>
2.3.3 : Funciones y argumentos	<b>56</b>
2.3.4 : Variables y su ámbito	58
2.3.5 : Funciones con return.	60
<b>2.4 Tema 5 : Árreglos</b>	
2.4.1 : Definición de arreglos	<b>67</b>
2.4.2 : Creacion de arreglos	67
2.4.3 : Trabajando con arreglo de objetos	72
<b>2.5 Tema 6 : Expresiones regulares</b>	
2.5.1 : Definición de expresiones regulares	78
2.5.2 : Creación de expresiones	78
2.5.3 : Manejo de Caracteres especiales	81
<b>Unidad de Aprendizaje 3</b>	
<b>PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVASCRIPT</b>	<b>96</b>
<b>3.1 Tema 7 : Introducción</b>	
3.1.1 : Definición de programación orientada a objetos	97
3.1.2 : Definición de clases, Objetos y eventos	97
3.1.3 : Implementando el paradigma de la POO en páginas web	109
<b>3.2 Tema 8 : Clases predefinidas</b>	<b>118</b>
3.2.1 : Array	118

3.2.2 : Date	121
3.2.3 : Math	124
3.2.4 : String	127

#### Unidad de Aprendizaje 4

<b>JQUERY</b>	<b>134</b>
<b>4.1 Tema 9 : Introducción</b>	<b>135</b>
4.1.1 : Definicion	135
4.1.2 : Breve referencia CSS	135
4.1.3 : Software necesario	138
4.1.4 : Selectores	138
<b>4.2 Tema 10 : Características principales</b>	<b>146</b>
4.2.1 : Constructor	146
4.2.2 : Iteracion implicita	147
4.2.3 : Consultas a través del DOM	147
4.2.4 : Encadenamiento (chaining)	150
<b>4.3 Tema 11 : Manejo de eventos</b>	<b>155</b>
4.3.1 : Introduccion a los eventos	155
4.3.2 : Evento click	159
4.3.3 : Comportamiento en cola	160
4.3.4 : Evento hover	161
4.3.5 : Eventos del teclado	162
4.3.6 : Borrar eventos	164
<b>4.4 Tema 12 : Efectos y modificaciones sobre el DOM</b>	<b>169</b>
4.4.1 : Introducción a los efectos	169
4.4.2 : Efectos incorporados	169
4.4.3 : Efectos personalizados	174
4.4.4 : Control de efectos	177

#### Unidad de Aprendizaje 5

<b>JSON</b>	<b>181</b>
<b>5.1 Tema 13 : Introducción</b>	<b>182</b>
5.1.1 Definicion	182
5.1.2 Estructura de un archivo JSON	182
5.1.3 Creación de un archivo JSON	184

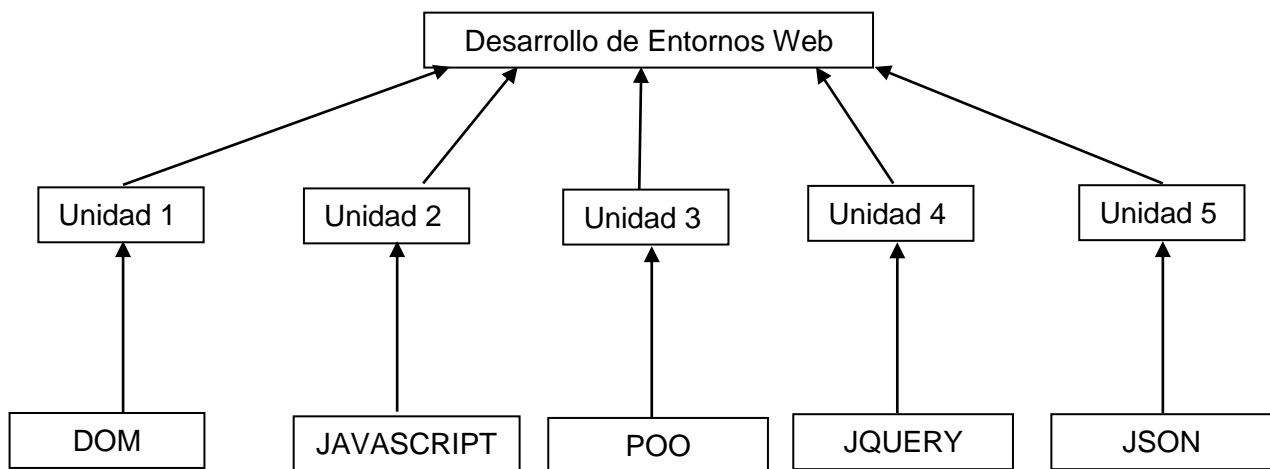
# Presentación

Desarrollo de entornos web es un curso que pertenece a la línea técnica y se dicta en las carreras Computación e Informática, Administración y Sistemas, Redes y Electrónica. Brinda a los alumnos un conjunto de aplicativos, como editores de textos para HTML5, CSS, JavaScript, así como JQuery y JSON, para el diseño y desarrollo de sitios web con aplicaciones multimedia y validación de formularios.

El curso es eminentemente práctico y consiste en el diseño de páginas web y programación avanzada con JavaScript para realizar operaciones, cálculos y validaciones en las mismas. En primer lugar, se inicia con el lenguaje JavaScript básico y avanzado con aplicaciones en el desarrollo de entornos web. Luego, estudiaremos JQuery que es un producto que sirve como base para la programación avanzada de aplicaciones compatibles con cualquier navegador. Finalmente, estudiaremos JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos.



# Red de contenidos









# EL MODELO DE OBJETO DE DOCUMENTO

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno diseña y construye páginas para un sitio web aplicando el modelo de objetos de documento DOM.

## TEMARIO

### 1.1 Tema 1 : EI DOM

- 1.1.1 : Definición
- 1.1.2 : Accediendo a los elementos
- 1.1.3 : Manipulando los estilos de los documentos

## ACTIVIDADES PROPUESTAS

- Mostrar número de enlaces de la pagina
- Mostrar la dirección url a la que enlaza el penúltimo enlace
- Mostrar número de enlaces que enlazan a <http://prueba>
- Mostrar número de enlaces del tercer párrafo
- Mostrar u ocultar texto.
- Agregar elementos a una lista.



## EL DOM

### 1.1.1. Definición

Acorde al W3C el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos validos HTML y bien contruidos XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula.

El Modelo de Objetos del Documento (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos que un programa Javascript puede actuar sobre ellos.

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos,<div>, etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario *"transformar"* la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

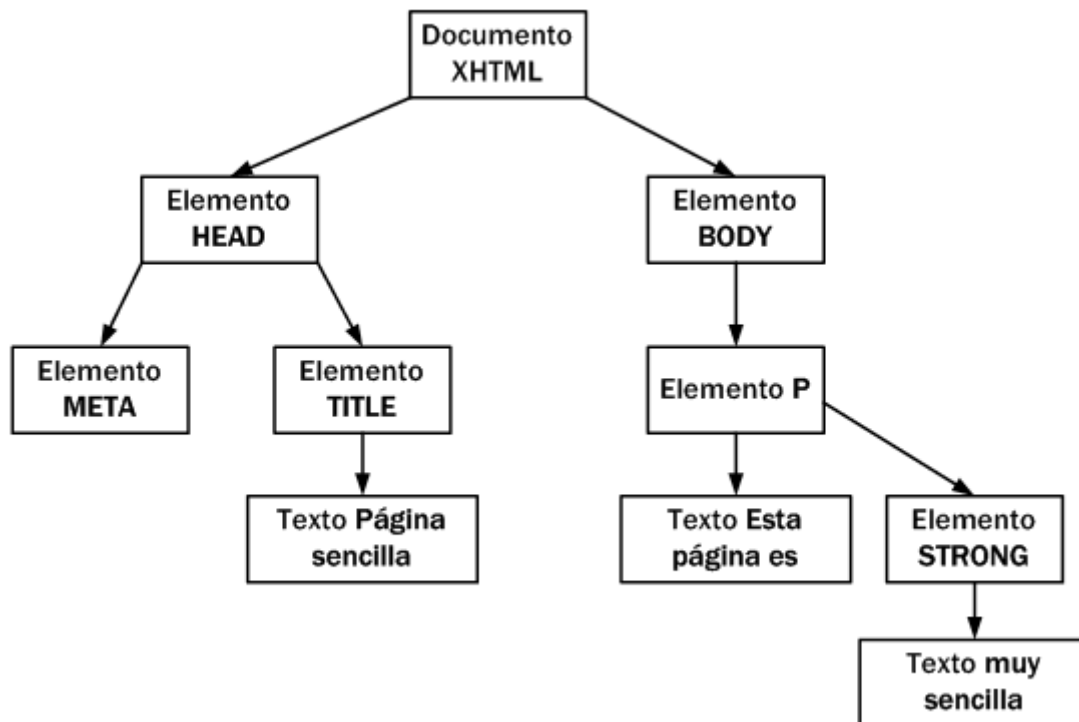
DOM transforma todos los documentos HTML en un conjunto de elementos llamados *nodos*, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama *"árbol de nodos"*.

La siguiente página HTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:



En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página HTML siempre es la misma: un nodo de tipo especial denominado "*Documento*".

A partir de ese nodo raíz, cada etiqueta HTML se transforma en un nodo de tipo "*Elemento*". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta HTML se transforma en un nodo que deriva del nodo correspondiente a su "*etiqueta padre*".

La transformación de las etiquetas HTML habituales genera dos nodos: el primero es el nodo de tipo "*Elemento*" (correspondiente a la propia etiqueta HTML) y el segundo es un nodo de tipo "*Texto*" que contiene el texto encerrado por esa etiqueta HTML.

Así, la siguiente etiqueta HTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:

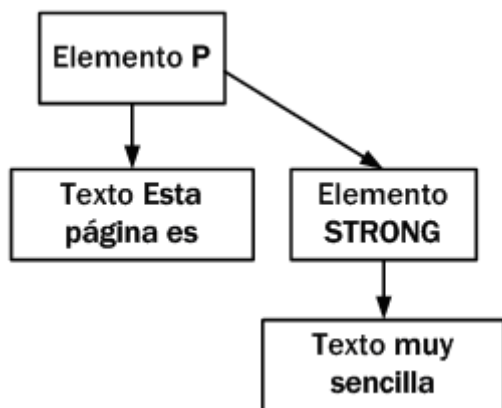


De la misma forma, la siguiente etiqueta HTML:

<p>Esta página es <strong>muy sencilla</strong></p>

Genera los siguientes nodos:

- Nodo de tipo "*Elemento*" correspondiente a la etiqueta <p>.
- Nodo de tipo "*Texto*" con el contenido textual de la etiqueta <p>.
- Como el contenido de <p> incluye en su interior otra etiqueta HTML, la etiqueta interior se transforma en un nodo de tipo "*Elemento*" que representa la etiqueta <strong> y que deriva del nodo anterior.
- El contenido de la etiqueta <strong> genera a su vez otro nodo de tipo "*Texto*" que deriva del nodo generado por <strong>.



La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas HTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta HTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

Como se puede suponer, las páginas HTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

Decimos que una página web es un documento HTML. Este documento puede ser representado de diferentes maneras:

- Representación web:** como una página web en un navegador donde vemos imágenes, texto, colores, etc.
- Representación texto:** como un texto plano (código HTML) que podemos visualizar en cualquier editor de textos como el bloc de notas de Windows ó Notepad++ ó cualquier otro.

- c) **Representación DOM:** como un árbol donde los elementos de la página web están organizados jerárquicamente, con nodos superiores (nodos padre o parent) y nodos que derivan de los nodos padre (nodos hijo o child).

Veamos un ejemplo de representación DOM.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>

<head>
  <title>Ejemplo DOM - aprenderaprogramar.com</title>
  <meta charset="utf-8">
  <style type="text/css">
    body {background-color:yellow; font-family: sans-serif;}
    label {color: maroon; display:block; padding:5px;}
  </style>
</head>

<body>
  <div id="cabecera">
    <h1>Portal web aprenderaprogramar.com</h1>
    <h2>Didáctica y divulgación de la programación</h2>
  </div>
  <!-- Formulario de contacto -->
  <form name ="formularioContacto" class="formularioTipo1" method="get"
  action="accion.html">
    <p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
    <label for="nombre">
      <span>Nombre:</span>
      <input id="nombre" type="text" name="nombre" />
    </label>
    <label for="apellidos">
      <span>Apellidos:</span>
      <input id="apellidos" type="text" name="apellidos" />
    </label>
    <label for="email">
      <span>Correo electrónico:</span>
      <input id="email" type="text" name="email" />
    </label>
    <label>
      <input type="submit" value="Enviar">
      <input type="reset" value="Cancelar">
    </label>
  </form>
</body>

</html>
```

La anterior representación se corresponde con la representación del documento como texto. La imagen que vemos en el navegador se corresponde con la representación del documento como página web en un navegador.

# Portal web aprenderaprogramar.com

## Didáctica y divulgación de la programación

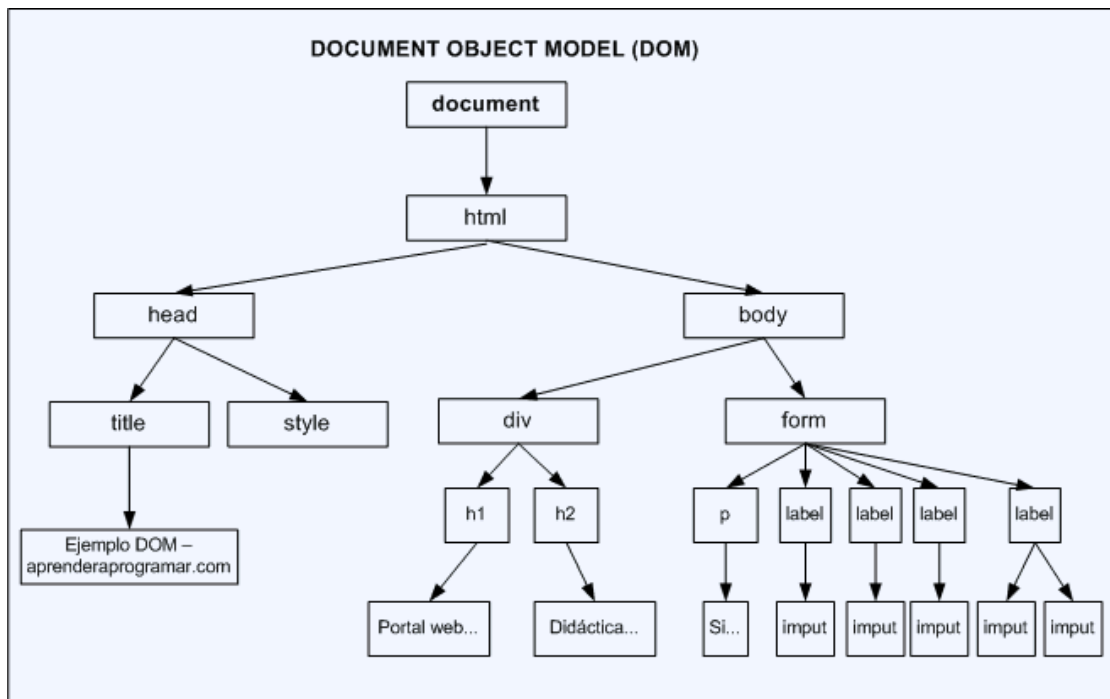
Si quieres contactar con nosotros envíanos este formulario relleno:

Nombre:

Apellidos:

Correo electrónico:

La representación del documento conforme al estándar del DOM sería (de forma aproximada) esta:



La representación anterior es solo aproximada: no nos va a interesar representar una página web conforme al DOM, simplemente queremos conocer cómo se estructura una página web conforme al DOM para saber cómo podemos acceder a sus elementos y manipularlos usando JavaScript (u otro lenguaje).

El DOM no es parte de JavaScript, de hecho puede ser utilizado por otros lenguajes de programación. No obstante, el DOM está íntimamente ligado a JavaScript ya que JavaScript lo utilizará con profusión para acceder y modificar las páginas web dinámicamente.

Decimos que conforme al DOM la página web se representa como un árbol de nodos, interconectados y relacionados de acuerdo con una jerarquía.

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>
<ul>
<li>Primer punto en la lista</li>
<li>Otro punto en la lista</li>
</ul>
</body>
```

Como puede verse un elemento [a] se encuentra localizado dentro de un elemento [p] del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo [p], de manera similar [p] es el nodo padre. Los dos nodos li son hijos del mismo padre, llamándose nodos hermanos o simplemente hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos comúnmente son asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como <p>, <img> y <div> por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

### Siempre use el DOCTYPE correcto

El DOCTYPE (abreviado del inglés “document type declaration”, declaración del tipo de documento) informa cual versión de (X)HTML se usará para validar el documento; existen varios tipos a seleccionar. El DOCTYPE, debe aparecer siempre en la parte superior de cada página HTML y siendo un componente clave de las páginas web “obedientes” a los estándares.

En caso de usarse un DOCTYPE incompleto, no actualizado o simplemente no usarlo llevará al navegador a entrar en modo raro o extraño, donde el navegador asume que se ha programado fuera de los estándares.

Todavía todos los navegadores actuales no son capaces de procesar correctamente todos los tipos de documentos, sin embargo, muchos de ellos funcionan correctamente en los navegadores más utilizados actualmente, tales como:

HTML 4.01 Strict y Transitional, XHTML 1.0 Strict y Transitional los se comportan del modo correcto en Internet Explorer (versión 6, 7 Beta), Mozilla y Opera 7. De ahora en adelante se adoptará para cada ejemplo HTML 4.01 Strict :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Resultando una única línea de código, o dos líneas con un salto de línea después de EN”.

### La importancia de validar el HTML

Si los elementos son anidados de manera inadecuada pueden generarse problemas, véase la siguiente línea:

```
<p>Estos elementos han sido <strong>incorrectamente </p>anidados </strong>
```



El árbol que resulta de esto se encuentra incorrectamente anidado del todo, por tanto generará errores inesperados en los navegadores. Manteniendo su HTML válido se pueden evitar tales problemas.

## Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

1. **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
2. **Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
3. **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
4. **Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.
5. **Comment**, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos existentes que no se van a considerar son `DocumentType`, `CDataSection`, `DocumentFragment`, `Entity`, `EntityReference`, `ProcessingInstruction` y `Notation`.

### 1.1.2. Accediendo a los elementos

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>
<ul>
<li>Primer punto en la lista</li>
<li>Otro punto en la lista</li>
</ul>
</body>
```

Como puede verse un elemento [a] se encuentra localizado dentro de un elemento [p] del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo [p], de manera similar [p] es el nodo padre. Los dos nodos li son hijos del mismo padre, llamándose nodos hermanos o simplemente hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos comúnmente son asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como <p>, <img> y <div> por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

Afortunadamente, Javascript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades.

Si desea encontrar de manera rápida y fácil un elemento se tiene a la mano el método `getElementById`. El mismo permite un acceso inmediato a cualquier elemento tan sólo conociendo el valor de su atributo `id`. Véase el siguiente ejemplo:

```
<p>
<a id="contacto" href="contactos.html">Contáctenos</a>
</p>
```

Puede usarse el atributo `id` del elemento `a` para acceder al mismo:

```
var elementoContacto = document.getElementById("contacto");
```

Ahora el valor de la variable `elementoContacto` está referida al elemento `[a]` y cualquier operación sobre la misma afectará el hipervínculo.

El método `getElementById` es adecuado para operar sobre un elemento en específico, sin embargo, en ocasiones se necesita trabajar sobre un grupo de elementos por lo que en este caso puede utilizarse el método `getElementsByTagName`. Este retorna todos los elementos de un mismo tipo. Asumiendo la siguiente lista desordenada:

```
<ul>
<li><a href="editorial.html">Editorial</a></li>
<li><a href="semblanza.html">Autores</a></li>
<li><a href="noticias.html">Noticias</a></li>
<li><a href="contactos.html">Contáctenos</a></li>
</ul>
```

Puede obtenerse todos los hipervínculos de la siguiente manera:

```
var hipervinculos= document.getElementsByTagName("a");
```

El valor de la variable `hipervinculos` es una colección de elementos `[a]`. Las colecciones son arreglos pudiéndose acceder a cada elemento a través de la ya conocida notación con corchetes.

Los elementos devueltos por `getElementsByTagName` serán ordenado según el orden que aparezcan en el código fuente. Por tanto para el caso anterior quedaría así:

- `hipervinculos[0]` el elemento `[a]` para “Editorial”
- `hipervinculos[1]` el elemento `[a]` para “Autores”
- `hipervinculos[2]` el elemento `[a]` para “Noticias”
- `hipervinculos[3]` el elemento `[a]` para “Contáctenos”

Otras maneras de acceder a un elemento usando su `id` es `document.all["id"]` la cual fue introducida en Internet Explorer 4 y `document.layers["id"]` introducida por Netscape 5 por que el W3C todavía no había estandarizado la manera de acceder a los elementos mediante su `id`. Sin embargo, no se recomienda su uso porque al estar fuera de los estándares actuales hay navegadores que no soportan estos métodos.

Por otro lado existen varios elementos en un documento HTML que pueden ser accedidos de otras maneras. El elemento `body` de un documento puede accederse a través de la forma `document.body`, mientras que el conjunto de todos los formularios en un documento puede encontrarse en `document.forms`, así mismo el conjunto de todas las imágenes sería mediante `document.images`.

Actualmente la mayoría de los navegadores soportan estos métodos aún así es recomendable el uso del método `getElementsByTagName`, véase el siguiente ejemplo para acceder al elemento `body`:

```
var body = document.getElementsByTagName("body")[0];
```

### Creando elementos y textos

La creación de nodos es posible mediante el uso de dos métodos disponibles en el objeto `document`. Dichos métodos son:

- `createElement(Tipo cadena)`: Crea un nuevo elemento del tipo especificado y devuelve un referencia a dicho elemento.
- `createTextNode(Cadena de texto)`: Crea un nuevo nodo de texto con el contenido especificado en la cadena de texto.

El siguiente ejemplo muestra cómo se crea un nuevo elemento de párrafo vacío:

```
var nuevoEnlace = document.createElement("a");
```

La variable `nuevoEnlace` ahora referencia un nuevo enlace listo para ser insertado en el documento. El texto que va dentro del elemento `[a]` es un nodo de texto hijo, por lo que debe ser creado por separado.

```
var nodoTexto = document.createTextNode("Semblanza");
```

Luego si desea modificar el nodo de texto ya existente, puede utilizarse la propiedad `nodeValue`, esta permite coger y poner el nodo de texto:

```
var textoViejo = nodoTexto.nodeValue;
```

```
nodoTexto.nodeValue = "Novedades";
```

El valor de la variable `textoViejo` es ahora "Semblanza" y el nuevo texto "Novedades". Se puede insertar un elemento o texto (nodo) como último hijo de un nodo ya existente usando el método `appendChild`. Este método coloca el nuevo nodo después de todos los hijos del nodo.

```
NuevoEnlace.appendChild(nodoTexto);
```

Ahora todo lo que se necesita es insertar el enlace en el cuerpo del documento. Para hacer esto, se necesita una referencia al elemento `body` del documento, teniendo como guía los estándares siguientes:

```
var cuerpoRef = document.getElementsByTagName("body")[0];  
cuerpoRef.appendChild(nuevoEnlace);
```

Otra manera sería utilizando el método `getElementById`. Para ello se asume que la etiqueta `<body>` tiene asignado un valor para el atributo `id`.

```
<body id="cuerpo">  
var cuerpoRef = document.getElementById("cuerpo");  
cuerpoRef.appendChild(nuevoEnlace);
```

Existen básicamente tres maneras mediante las cuales un nuevo elemento o nodo de texto puede ser insertado en una página Web. Todo ello depende del punto en el cual se desee insertar el nuevo nodo: como último hijo de un elemento, antes de otro nodo o reemplazo para un nodo.

El caso de apertura de un nuevo hijo ya fue visto en el ejemplo anterior, luego para insertar el nodo antes de otro nodo se realiza utilizando el método `insertBefore` de su elemento padre, mientras que el reemplazo de nodo se utiliza el método `replaceChild` de su elemento padre.

Al usar `insertBefore`, se necesita tener referencias al nodo que va ser insertado y donde va a ser insertado, considérese el siguiente código HTML:

```
<p id="mwEnlaces">
<a id="editor" href="editorial.html">Editorial</a>
</p>
```

Luego el nuevo enlace será insertado antes de enlace ya existente llamando el método `insertBefore` desde el nodo padre (párrafo):

```
var anclaTexto = document.createTextNode("Actualidad");
var nuevoAncla = document.createElement("a");
nuevoAncla.appendChild(anclaTexto);
var anclaExistente = document.getElementById("editor");
var padre = anclaExistente.parentNode;
var nuevoHijo = padre.insertBefore(nuevoAncla, anclaExistente);
```

Si se hiciera una traducción del DOM hacia HTML después de esta operación el resultado sería el siguiente:

```
<p id="mwEnlaces">
<a> Actualidad </a><a id="editor" href="editorial.html">Editorial</a>
</p>
```

En el caso de reemplazar el enlace usando `replaceChild`:

```
var nuevoHijo = padre.replaceChild(nuevoAncla, anclaExistente);
```

El DOM lucirá así:

```
<p id="mwEnlaces">
<a> Actualidad </a>
</p>
```

### Usando `innerHTML`

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso recurriéndose a la propiedad `innerHTML`. Dicha propiedad fue introducida por Microsoft permitiendo leer y escribir el contenido HTML de un elemento.

Por ejemplo, puede crearse fácilmente una tabla con múltiples celdas e insertarla luego en la página con `innerHTML`:

```
var tabla = '<table border="0">';
tabla += '<tr><td>Celda 1</td><td>Celda 2</td><td>Celda 3</td></tr>';
```

```
tabla += '</table>';  
document.getElementById("datos").innerHTML = tabla;
```

### Eliminando un elemento o nodo de texto

Se pueden eliminar nodos existentes y nuevos. El método `removeChild` permite eliminar nodos hijos a cualquier nodo con tan sólo pasarle las referencias del nodo hijo [a] eliminar y su correspondiente padre. Para mejor comprensión retómese el ejemplo anterior:

```
<p id="mwEnlaces">  
<a id="editor" href="editorial.html">Editorial</a>  
</p>
```

El método `removeChild` será usado para eliminar el hipervínculo del elemento padre párrafo:

```
var ancla = document.getElementById("editor");  
var padre = ancla.parentNode;  
var hijoRemovido = padre.removeChild(ancla);
```

La variable `hijoRemovido` todavía hace referencia al elemento, de manera que fue removido pero no destruido, no pudiéndose localizar en ninguna parte del DOM. Este se encuentra disponible en memoria como si fuera creado usando el método `createElement`. Esto permite posicionarlo en cualquier otra parte de la página.

### Lectura y escritura de los atributos de un elemento

Las partes más frecuentemente usadas de un elemento HTML son sus atributos, tales como: `id`, `class`, `href`, `title`, estilos CSS, entre muchas otras piezas de información que pueden ser incluidas en una etiqueta HTML.

Los atributos de una etiqueta son traducidos por el navegador en propiedades de un objeto. Dos métodos existen para leer y escribir los atributos de un elemento, `getAttribute` permite leer el valor de un atributo mientras que `setAttribute` permite su escritura.

En ocasiones se hace necesario ver las propiedades y métodos de un determinado elemento, esto puede realizarse mediante la siguiente función utilitaria:

```
function inspector(el) {  
    var str = "";  
    for (var i in el){  
        str+=i + ": " + el.getAttribute(i) + "\n";  
    }  
    alert(str);  
}
```

Para usar la función `inspector()` tan sólo debe pasarle la referencia al elemento, continuando con el ejemplo anterior resulta:

```
var ancla = document.getElementById("editor");  
  
inspector(ancla);
```

Para modificar el atributo title del hipervínculo, elemento referenciado por la variable ancla, se usará el setAttribute, pasándole el nombre del atributo y el valor:

```
var ancla = document.getElementById("editor");
ancla.setAttribute("title", "Artículos de programación");
var nuevoTitulo = ancla.getAttribute("title");
```

El valor de la variable nuevoTitulo es ahora "Artículos de programación".

### 1.1.3. Manipulando los estilos de los documentos

Como se ha visto, los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.

Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS pero sin los guiones y llevando la primera letra de las palabras a mayúsculas. Véase el siguiente ejemplo para mayor entendimiento donde se utiliza un atributo CSS modelo:

algun-atributo-css

Tendrá como equivalente la siguiente propiedad o método en Javascript:

algunAtributoCss

Por tanto, para cambiar el atributo CSS font-family de un elemento, podría realizarse de lo siguiente:

```
ancla.style.fontFamily = 'sans-serif';
```

Los valores CSS en Javascript serán en su mayoría del tipo cadena; por ejemplo: font-size, pues posee dimensiones tales como "px", "%". Sólo los atributos completamente numéricos, tales como z-index serán del tipo entero.

En muchos casos es necesario aparecer y desaparecer un determinado elemento, para ellos se utiliza el atributo CSS display, por ejemplo, para desaparecer:

```
ancla.style.display = 'none';
```

Luego para volverlo a mostrar se le asigna otro valor:

```
ancla.style.display = 'inline';
```

El siguiente código permite aplicar estilo de forma general al documento y el segundo es más específico, al ser dirigido a un bloque o elemento definido por un identificador ID.

```
document.body.style.fontSize="24px"
document.getElementById("id").style.property="value"
```

```
var el = document.getElementById(id);
el.style.color = "red";
```

```
el.style.fontSize = "15px";
el.style.backgroundColor = "#FFFFFF";
```

### Lista de propiedades de Javascript para modificar el estilo de las páginas

Propiedad	Descripción
<b>background</b>	<p>Establece todas las propiedades del background o fondo en una única declaración de forma abreviada, son 5 las propiedades separadas por un espacio:  <i>background-color, background-image, background-repeat, background-attachment, background-position</i>            Se usa de la siguiente forma:  <i>Object.style.background="color image repeat attachment position"</i>            Por ejemplo:  <i>document.body.style.background="#f3f3f3 url('foto.png') no-repeat right top";</i></p>
<b>backgroundAttachment</b>	<p>Especifica cuando la imagen empleada como fondo permanece fija o se desplaza. Las opciones son:  <i>scroll</i> Se desplaza con la página (Predeterminado)  <i>fixed</i> Permanece fija</p>
<b>backgroundColor</b>	Especifica el color del fondo
<b>backgroundImage</b>	<p>Especifica la ubicación de la imagen a emplear como fondo.            Por ejemplo:  <i>document.body.style.backgroundImage="url('foto.png')";</i></p>
<b>backgroundPosition</b>	<p>Posición de la imagen utilizada. Se usan dos valores, si solo se especifica uno la imagen será centrada. Los valores pueden ser:  <i>top left, top center, top right, center left, center center, center right, bottom left, bottom center, bottom right</i>  <i>x% y%</i>  <i>xpos ypos</i></p>
<b>backgroundRepeat</b>	<p>Especifica si la imagen empleada se repite para llenar toda el área de la página.  <i>repeat</i> Es repetida en las dos dimensiones, es el valor predeterminado.  <i>repeat-x</i> Solo se repite en el eje horizontal  <i>repeat-y</i> Solo se repite en el eje vertical  <i>no-repeat</i> No es repetida</p>
<b>border</b>	<p>Establece las propiedades del borde en una sola declaración de la siguiente forma:  <i>Object.style.border="width style color"</i>            Por ejemplo:  <i>document.getElementById("ejemplo").style.border="thick solid green";</i>            Las restantes propiedades que se pueden usar de forma individual para definir el estilo del borde son:  <i>borderBottom</i>  <i>borderBottomColor</i>  <i>borderBottomStyle</i></p>

	<b><i>borderBottomWidth</i></b> <b><i>borderColor</i></b> <b><i>borderLeft</i></b> <b><i>borderLeftColor</i></b> <b><i>borderLeftStyle</i></b> <b><i>borderLeftWidth</i></b> <b><i>borderRight</i></b> <b><i>borderRightColor</i></b> <b><i>borderRightStyle</i></b> <b><i>borderRightWidth</i></b> <b><i>borderStyle</i></b> <b><i>borderTop</i></b> <b><i>borderTopColor</i></b> <b><i>borderTopStyle</i></b> <b><i>borderTopWidth</i></b> <b><i>borderWidth</i></b>
outline	<p>Especifica las propiedades de outline (borde de fuente) en una sola declaración de la siguiente forma:  <b><i>Object.style.outline="width style color"</i></b>          Por ejemplo:  <b><i>document.getElementById("ejemplo").style.outline="thick solid #0000FF";</i></b>          Las restantes propiedades que se pueden usar de forma individual para definir el estilo outline son:  <b><i>outlineColor</i></b>  <b><i>outlineStyle</i></b>  <b><i>outlineWidth</i></b></p>
listStyle	<p>Especifica las siguientes propiedades en una sola declaración:  <b><i>list-style-image</i></b>, <b><i>list-style-position</i></b> y <b><i>list-style-type</i></b>, se usa:  <b><i>Object.style.listStyle="type position image"</i></b>          También se pueden usar de forma individual:  <b><i>listStyleImage</i></b>  <b><i>listStylePosition</i></b>  <b><i>listStyleType</i></b></p>
margin	<p>Establece todas las propiedades de los márgenes en una sola declaración. Esta propiedad puede establecerse indicando desde 1 a 4 valores.</p> <ul style="list-style-type: none"> <li>• Un valor, por ejemplo: <b><i>div {margin: 50px}</i></b> todos los márgenes serán de 50px</li> <li>• Dos valores, por ejemplo: <b><i>div {margin: 50px 10px}</i></b> top (superior) y bottom (inferior) serán de 50px, left (izquierda) y right (derecha) serán de 10px.</li> <li>• Tres valores, por ejemplo: <b><i>div {margin: 50px 10px 20px}</i></b> el valor de top será 50px, left y right será 10px, bottom será 20px.</li> <li>• Cuatro valores, por ejemplo: <b><i>div {margin: 50px 10px 20px 30px}</i></b> el valor de top será 50px, right será de 10px, bottom será de 20px, left será de 30px.</li> </ul> <p>Los valores se pueden definir de tres formas alternativas  <b><i>"% length auto"</i></b>          Por ejemplo:  <b><i>document.getElementById("ejemplo").style.margin="2px 2px 5px 5px";</i></b>          También se pueden usar de forma individual:  <b><i>marginBottom</i></b>  <b><i>marginLeft</i></b></p>



	<i>marginRight</i> <i>marginTop</i>
padding	Especifica de forma conjunta los valores del padding (espaciado) de un elemento, se pueden usar hasta cuatro valores. Para emplear los valores utiliza el mismo método de margin. Por ejemplo: <code>document.getElementById("ejemplo").style.padding="10px 0 5px 0";</code> También se pueden usar de forma individual: <i>paddingBottom</i> <i>paddingLeft</i> <i>paddingRight</i> <i>paddingTop</i>
cssText	Especifica una declaración de estilo como una cadena de texto.
clear	Especifica la posición de un elemento de forma relativa a un objeto que flota.
clip	Especifica que parte de un elemento posicionado es visible.
cssFloat	Establece la alineación horizontal de un elemento. Pueden usarse tres valores de la siguiente forma: <code>Object.style.cssFloat="left right none"</code> Por ejemplo: <code>document.getElementById("ejemplo").style.cssFloat="left";</code>
cursor	Establece el estilo a emplearse en el cursor del ratón de la siguiente forma: <code>Object.style.cursor="valor"</code> Los valores pueden ser: <i>auto(predeterminado)</i> , <i>crosshair</i> , <i>e-resize</i> , <i>help</i> , <i>move</i> , <i>n-resize</i> , <i>ne-resize</i> , <i>nw-resize</i> , <i>pointer</i> , <i>s-resize</i> , <i>se-resize</i> , <i>sw-resize</i> , <i>text</i> , <i>url</i> , <i>w-resize</i> , <i>wait</i> . Por ejemplo: <code>document.getElementById("ejemplo").style.cursor="pointer";</code>
display	Define la forma en que se muestra un elemento HTML, los métodos más empleados son: "inline" o "block", también es muy usado para ocultar elementos de la forma siguiente: <code>document.getElementById("ejemplo").style.display="none";</code>
overflow	Especifica que hacer si el contenido de un elemento rebasa el espacio que proporciona este.
position	Especifica el tipo de posicionamiento usado para un elemento, pueden usarse los siguientes valores ( <i>static</i> , <i>relative</i> , <i>absolute</i> o <i>fixed</i> )
verticalAlign	Especifica la alineación vertical de un elemento
visibility	Especifica la visibilidad de un elemento
zIndex	Especifica el orden del posicionamiento de un elemento
orphans	Especifica el mínimo número de líneas para un elemento que tiene que ser visible en la parte inferior de la página.
widows	Especifica el mínimo número de líneas para un elemento que tiene que ser visible en la parte superior de la página.

<b>borderCollapse</b>	Establece si el borde de una tabla debe colapsar en una simple línea
<b>borderSpacing</b>	Espaciado del borde en una tabla
<b>captionSide</b>	Posición del elemento caption de una tabla
<b>emptyCells</b>	Especifica si se debe mostrar el borde y fondo de una celda vacía
<b>color</b>	Establece el color del texto.
<b>direction</b>	Establece la dirección del texto
<b>font</b>	<p>Establece todas las propiedades del elemento fuente en una sola declaración.  Pueden usarse los valores: <i>font-style</i>, <i>font-variant</i>, <i>font-weight</i>, <i>font-size</i>, <i>line-height</i> y <i>font-family</i>.  Hazlo de la siguiente forma:  <b><i>Object.style.font="style variant weight size/lineHeight family"</i></b>  Por ejemplo:  <b><i>document.getElementById("ejemplo").style.font="italic bold 18px arial,serif";</i></b>  También pueden usarse de forma individual las siguientes propiedades:  <i>fontFamily</i>  <i>fontSize</i>  <i>fontSizeAdjust</i>  <i>fontStyle</i>  <i>fontVariant</i>  <i>fontWeight</i></p>
<b>letterSpacing</b>	Especifica el espacio entre caracteres en el texto.
<b>lineHeight</b>	Especifica el espacio entre líneas en el texto.
<b>textAlign</b>	Especifica la alineación horizontal del texto.
<b>textDecoration</b>	Especifica el estilo de decoración del texto.
<b>textIndent</b>	Establece la indentación utilizada en la primera línea del texto.
<b>textTransform</b>	Establece la propiedad Transform usada en el texto.
<b>wordSpacing</b>	Establece el espacio entre palabras en el texto.
	<p>Para la posición y tamaño de cada elemento se pueden usar las siguientes propiedades:  <i>Bottom</i>, <i>left</i>, <i>right</i>, <i>top</i>  <i>Height</i>, <i>width</i>, <i>maxHeight</i>, <i>maxWidth</i>, <i>minHeight</i>, <i>minWidth</i></p>

## Laboratorio 1.1

En este laboratorio vamos a acceder a los elementos de una página web mediante DOM para contar el número de párrafos, el número de enlaces en un determinado párrafo, el número de enlaces a una url específica y mostrar el texto de un párrafo.

Lorem ipsum dolor sit amet, [consectetuer adipiscing elit](#). turpis. Quisque sapien nunc, posuere vitae, uis, faucibus ut, rhoncus non, mi. [Fusce porta](#). Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin consequat auctor diam. [Ut bibendum blandit est](#). Curabitur vestibulum. quis eros nec lectus tempor lacinia. Aliquam nec lectus nec neque aliquet dictum. Etiam [consequat sem quis massa](#). Donec aliquam euismod diam.

Vestibulum aliquet, nulla sit amet imperdiet suscipit, est, a [aliquam leo odio sed sem](#). Quisque eget eros vehicula diam euismod tristique. Ut dui. Donec in metus dictum interdum. Proin [egestas](#) adipiscing ligula. Duis iaculis laoreet turpis. ipsum odio euismod tortor, a vestibulum nisl mi at odio. [Sed non lectus non est pellentesque](#) auctor.

Reporte:

Numero de enlaces en la pagina = 3

El penultimo enlace apunta a: <http://prueba4>

3 enlaces apuntan a <http://prueba>

Numero de enlaces del tercer párrafo = 3

Cantidad de párrafos: 3

texto del párrafo 1:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. turpis. Quisque sapien nunc, posuere vitae, uis, faucibus ut, rhoncus non, mi. [Fusce porta](#). Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema1 y lo guarda en la carpeta Tema1.
2. Cree una archivo js e inserte el siguiente código:

```
// Numero de enlaces de la pagina
var enlaces = document.getElementsByTagName("a");
```

```
// Direccion del penultimo enlace
var penultimo = enlaces[enlaces.length-2];

// Numero de enlaces que apuntan a http://prueba
var contador = 0;
for(var i=0; i<enlaces.length; i++) {
    // Es necesario comprobar los enlaces http://prueba y
    // http://prueba/ por las diferencias entre navegadores
    if(enlaces[i].getAttribute('href') == "http://prueba" ||
        enlaces[i].getAttribute('href') == "http://prueba/") {
        contador++;
    }
}

// Numero de enlaces del tercer párrafo
var parrafos = document.getElementsByTagName("p");
enlaces = parrafos[2].getElementsByTagName("a");

//mostrar reporte en el documento
document.write("Reporte: <br>Numero de enlaces en la pagina =
"+enlaces.length+
"<br>El penultimo enlace apunta a: "+penultimo.getAttribute('href')+
"<br>" + contador + " enlaces apuntan a http://prueba"+
"<br>Numero de enlaces del tercer parrafo = "+enlaces.length);

var cantidad=document.getElementsByTagName("p");
document.write("<br>Cantidad de parrafos: "+cantidad.length);

document.write("<br>texto del parrafo 1: <br>" +
    document.getElementsByTagName("p")[0].innerText);
```

2. Guardelo en la carpeta js con el nombre códigos.js
3. Cree un archivo html e inserte el siguiente código entre las etiquetas <body> y </body>:

```
<body>
<p>Lorem ipsum dolor sit amet, <a href="http://prueba">consectetur
adipiscing elit</a>.
turpis. Quisque sapien nunc, posuere vitae, uis, faucibus ut, rhoncus
non, mi.
<a href="http://prueba2">Fusce porta</a>. Duis pellentesque, felis eu
adipiscing ullamcorper,
odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum
dolor sit amet,
consectetur adipiscing elit.</p>
<p>primis in faucibus orci luctus et ultrices posuere cubilia Curae;
Proin consequat auctor
diam. <a href="http://prueba">Ut bibendum blandit est</a>. Curabitur
vestibulum. quis eros
nec lectus tempor lacinia. Aliquam nec lectus nec neque aliquet
dictum. Etiam
<a href="http://prueba3">consequat sem quis massa</a>. Donec aliquam
eiusmod diam. </p>
<p>Vestibulum aliquet, nulla sit amet imperdiet suscipit, est, a
<a href="http://prueba">
aliquam leo odio sed sem</a>. Quisque eget eros vehicula diam euismod
tristique. Ut dui.
Donec in metus dictum interdum. Proin
<a href="http://prueba4">egestas</a> adipiscing ligula.
Duis iaculis laoreet turpis. ipsum odio euismod tortor, a vestibulum
```

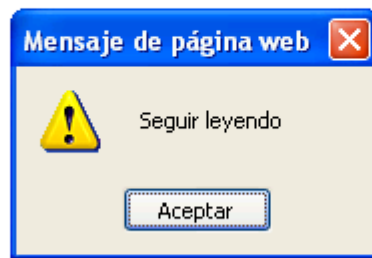
```
nisl mi at odio.  
<a href="http://prueba5">Sed non lectus non est pellentesque</a>  
autor.</p>  
  
<script type="text/javascript" src="js/codigos.js"></script>  
  
</body>
```

4. Guardelo en la carpeta principal Tema1 con el nombre accesoaobjetos.html. luego ejecute su archivo en un navegador.

## Laboratorio 1.2

En este laboratorio vamos a acceder a los elementos de una página web mediante DOM para mostrar un texto oculto. Los párrafos pueden ser texto falso. Se puede crear una variante usando un enlace ver más.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor, pede.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema1b y lo guarda en la carpeta Tema1b.
2. Cree una archivo js e inserte el siguiente código:

```
alert("Seguir leyendo");  
var elemento = document.getElementById("adicional");  
elemento.className = "visible";  
  
var enlace = document.getElementById("enlace");  
enlace.className = "oculto";
```

3. Grabelo en la carpeta js con nombre códigos.js
4. Cree una archivo css e inserte el siguiente código:

```
.oculto{display:none;}  
.visible{display:inline;}
```

5. Grabelo en la carpeta css con nombre estilo.css
6. Cree una archivo html e inserte el siguiente código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1" />
<title>Ejercicio 2 DOM</title>

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

<body>

<p id="texto">Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl
arcu, consequat congue, commodo nec, commodo ultricies, turpis.
Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor,
pede.

<span id="adicional" class="oculto">Nullam sit amet nisl elementum
elit convallis malesuada. Phasellus magna sem, semper quis, faucibus
ut, rhoncus non, mi. Duis pellentesque, felis eu adipiscing
ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis
scelerisque. Donec lacus neque, vehicula in, eleifend vitae,
venenatis.</span></p>

<script src="js/codigos.js" type="text/javascript"></script>

</body>

</html>
```

7. Grabe el archivo como mostrartexto.html y luego ejecute su pagina en un navegador.

# Resumen

1. El Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos válidos HTML y bien contruidos XML que define la estructura lógica de los documentos y el modo en que se accede y manipula.
2. El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto.
3. En HTML todas las etiquetas son elementos, tales como <p>, <img> y <div> por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.
4. Los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.
5. Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS pero sin los guiones y llevando la primera letra de las palabras a mayúsculas.

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [http://librosweb.es/libro/javascript/capitulo\\_5/ejercicios\\_sobre\\_dom.html](http://librosweb.es/libro/javascript/capitulo_5/ejercicios_sobre_dom.html)
- [http://librosweb.es/libro/ajax/capitulo\\_4.html](http://librosweb.es/libro/ajax/capitulo_4.html)
- <http://www.lawebera.es/como-hacer/ejemplos-css/cambiar-estilo-css-web-dinamicamente-i.php>
- <http://lineadecodigo.com/javascript/crear-elementos-html-con-javascript/>
-







# PROGRAMACIÓN BÁSICA EN JAVASCRIPT

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript, diseña programas, incorporados en una página del Sitio Web y para validar formularios

## TEMARIO

### 2.1 Tema 2 : Fundamentos de JavaScript

- 2.1.1 : Elementos de un programa en javascript
- 2.1.2 : Entradas y salidas: alert, input
- 2.1.3 : Estructuras de selección

### 2.2 Tema 3 : Estructuras de repetición

- 2.2.1 : Sintaxis
- 2.2.2 : Ejemplos aplicados al desarrollo web

### 2.3 Tema 4 : Funciones en el lenguaje JavaScript

- 2.3.1 : Introducción
- 2.3.2 : Sintaxis general
- 2.3.3 : Funciones y argumentos
- 2.3.4 : Variables y su ámbito
- 2.3.5 : Funciones con return.

### 2.4 Tema 5 : Árreglos

- 2.4.1 : Definición de arreglos
- 2.4.2 : Creacion de arreglos
- 2.4.3 : Trabajando con arreglo de objetos

### 2.5 Tema 6 : Expresiones regulares

- 2.5.1 : Definición de expresiones regulares
- 2.5.2 : Creación de expresiones
- 2.5.3 : Manejo de Caracteres especiales

**ACTIVIDADES PROPUESTAS**

- Los alumnos insertan un reloj digital en su página web.
- Los alumnos crean una galería de imágenes.
- Los alumnos validan un formulario con expresiones regulares

## 2.1. FUNDAMENTOS DE JAVASCRIPT

### 2.1.1. Elementos de un programa en javascript

JavaScript es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java.

El lenguaje fue inventado por Brendan Eich en la empresa Netscape Communications, que es la que fabricó los primeros navegadores de Internet comerciales.

Apareció por primera vez en el producto de Netscape llamado Netscape Navigator 2.0.

Se utiliza en páginas web HTML, para realizar tareas y operaciones en el marco de la aplicación cliente.

Los autores inicialmente lo llamaron Mocha y más tarde LiveScript pero fue rebautizado como JavaScript en un anuncio conjunto entre Sun Microsystems y Netscape, el 4 de diciembre de 1995.

El lenguaje JavaScript utiliza una sintaxis parecida a la de Java y admite:

- Estructuras de selección y de repetición, como **if...else**, **for** y **do...while**.
- Se utilizan llaves ({} ) para delimitar los bloques de instrucciones.
- El lenguaje admite varios tipos de datos, como String, Number, Boolean, Object y Array.
- Admite las características de fecha mejoradas, las funciones trigonométricas y las expresiones regulares.
- JavaScript usa prototipos en vez de clases. Puede definir un objeto creando una función constructora.
- JavaScript no declara explícitamente los tipos de datos de las variables. En muchos casos, JavaScript realiza las conversiones automáticamente cuando es necesario.

### 2.1.2. Entradas y salidas: alert, input

#### 2.1.2.1 La instrucción alert(cadena)

Se utiliza para mostrar mensajes o datos en cuadros de diálogos predefinidos. El siguiente ejemplo muestra el valor del atributo href de la etiqueta <a> llamada enlace.

Código javascript:

```
var enlace = document.getElementById("enlace");  
alert(enlace.href); // muestra http://www...com
```

Código html:

```
<a id="enlace" href="http://www...com">Enlace</a>
```

El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

Código javascript:

```
var imagen = document.getElementById("imagen");  
alert(imagen.style.margin);
```

Código html:

```

```

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

Código javascript:

```
var parrafo = document.getElementById("parrafo");  
alert(parrafo.style.fontWeight); // muestra "bold"
```

Código html:

```
<p id="parrafo" style="font-weight: bold;">...</p>
```

### 2.1.2.2 La instrucción prompt(cadena)

Se utiliza para pedir al usuario que introduzca un texto en una ventana modal. Veámoslo con un ejemplo:

```
<script>  
    // Pedimos al usuario que introduzca su nombre  
    var nombre = prompt("Introduzca su nombre");  
    // Mostramos texto concatenado con el nombre ingresado  
    alert("Hola " + nombre);  
}  
</script>
```

### 2.1.2.3 La instrucción document.write(cadena)

Se utiliza para escribir un texto en el documento html actual. El primer ejemplo escribe en la página el texto 'Hola Mundo', el segundo el resultado de la variable screen.width que devuelve el ancho de la pantalla en píxeles y el tercero agrupa texto y la variable.

```
<script>  
document.write('Hola Mundo<br>')  
</script>
```

```
<script>  
document.write(+screen.width+'<br>')  
</script>
```

```
<script>  
document.write('Ancho de la pantalla: '+screen.width+' píxeles')  
</script>
```

Se mostrará en la página lo siguiente:

Hola Mundo  
1024  
Ancho de la pantalla: 1024 pixeles

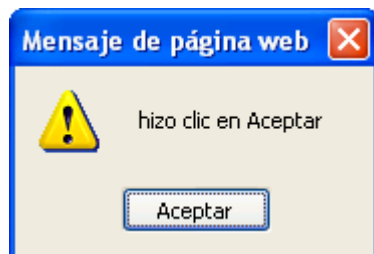
#### 2.1.2.4 La instrucción confirm(cadena)

Es muy útil para confirmar acciones que realicen acciones críticas como eliminación de datos o confirmar resultados. Por ejemplo, el siguiente código muestra un mensaje con dos botones Aceptar y Cancelar:

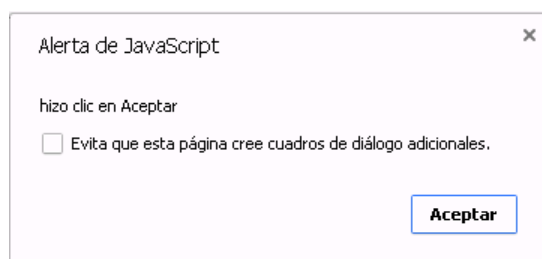
```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
<script>
    var a=1,b=5;
    var opcion=confirm(a+" es mayor que "+b+"?");
    if(opcion)alert("hizo clic en Aceptar");
    else alert("hizo clic en Cancelar");
</script>
</head>
<body>
</body>
</html>
```

Cada navegador muestra el cuadro de alerta de diferentes formas. Por ejemplo, del código anterior:

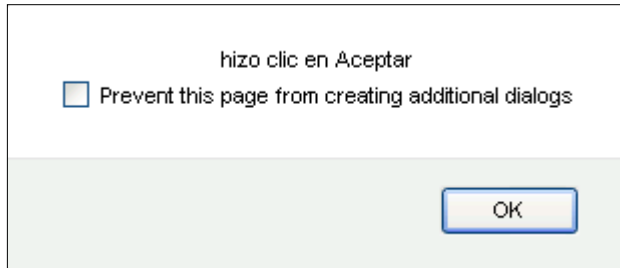
Internet Explorer muestra así el alert



Chrome muestra de la siguiente manera



Firefox muestra así el alert:



### 2.1.3. Estructuras de selección

#### 2.1.3.1. Estructura if

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {  
    ...  
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script. Ejemplo:

```
var mensaje = true;  
if(mensaje) {  
    alert("Hola Mundo");  
}
```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable mensaje tiene un valor de true y por tanto, el programa entra dentro del bloque de instrucciones del if.

El ejemplo se podría reescribir también como:

```
var mensaje = true;  
if(mensaje == true) {  
    alert("Hola Mundo");  
}
```

En este caso, la condición es una comparación entre el valor de la variable mensaje y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

```
var mensaje = true;  
// Se comparan los dos valores
```

```
if(mensaje == false) {  
    ...  
}
```

// Error - Se asigna el valor "false" a la variable

```
if(mensaje = false) {  
    ...  
}
```

La condición que controla el if() puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;
```

```
if(!mostrado) {  
    alert("Es la primera vez que se muestra el mensaje");  
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
```

```
var mensaje = true;
```

```
if(!mostrado && mensaje) {  
    alert("Es la primera vez que se muestra el mensaje");  
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de mostrado es false, el valor !mostrado sería true. Como la variable mensaje vale true, el resultado de !mostrado && mensaje sería igual a true && true, por lo que el resultado final de la condición del if() sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if().

### 2.1.3.2. Estructura if else

En ocasiones, las decisiones que se deben realizar no son del tipo *"si se cumple la condición, hazlo; si no se cumple, no hagas nada"*. Normalmente las condiciones suelen ser del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*.

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {  
    ...  
}  
else {  
    ...  
}
```



Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else {}. Ejemplo:

```
var edad = 18;
```

```
if(edad >= 18) {  
    alert("Eres mayor de edad");  
}  
else {  
    alert("Todavía eres menor de edad");  
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque else {}. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";  
  
if(nombre == "") {  
    alert("No has ingresado tu nombre");  
}  
else {  
    alert("Nombre grabado correctamente");  
}
```

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else {}.

### 2.1.3.3. Estructura if else if

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {  
    alert("Todavía eres muy pequeño");  
}  
else if(edad < 19) {  
    alert("Eres un adolescente");  
}  
else if(edad < 35) {  
    alert("Aun sigues siendo joven");  
}
```

```
else {  
  alert("Piensa en cuidarte un poco más");  
}
```

No es obligatorio que la combinación de estructuras if...else acabe con la instrucción else, ya que puede terminar con una instrucción de tipo else if().

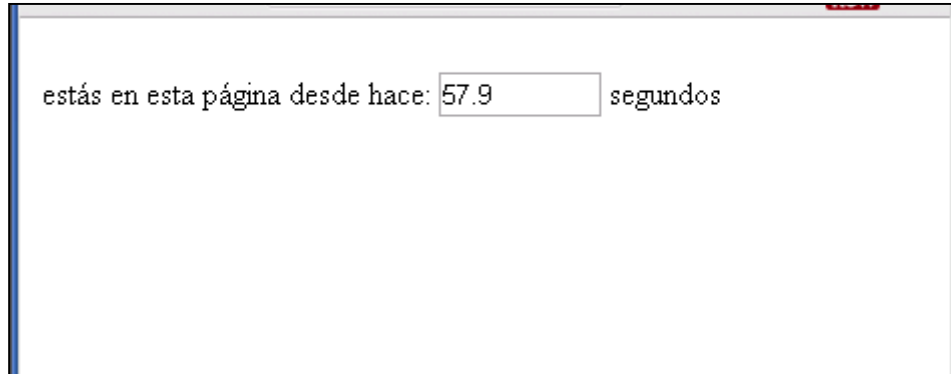
#### 2.1.3.4. Estructura switch

A diferencia del if else, el switch case nos permite tener varias opciones o simplificar un poco el código. En el siguiente ejemplo dependiendo del día de la semana imprime un texto diferente.

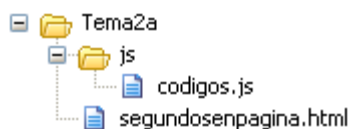
```
<html>  
<body>  
<script type="text/javascript">  
  var a = new Date();  
  dia=a.getDay();  
  switch (dia)  
  {  
    case 5:  
      document.write("<b>Viernes social</b>");  
      break;  
    case 6:  
      document.write("<b>Sábado deportivo</b>");  
      break;  
    case 0:  
      document.write("<b>Domingo familiar</b>");  
      break;  
    default:  
      document.write("<b>añoro que llegue el fin de semana!</b>");  
  }  
</script>  
  
<p>En este script el Domingo=0, Lunes=1, Martes=2, etc.</p>  
  
</body>  
</html>
```

## Laboratorio 2.1

En este laboratorio vamos a usar estructuras de selección para interactuar con el usuario. Se mostrará en la página el tiempo de permanencia del usuario expresado en segundos.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema2a y lo guarda en la carpeta Tema2a.
2. Cree un archivo js e inserte el siguiente código:

```
/*By George Chiang. (JK's ultimate JavaScript tutorial and free JavaScripts site!)  
http://www.geocities.com/SiliconValley/Vista/9892  
Credit MUST stay intact for use*/
```

```
var milisec=0  
var seconds=0  
document.d.d2.value='0'  
function display(){  
  if (milisec>=9){  
    milisec=0  
    seconds+=1  
  }  
  else  
    milisec+=1  
  document.d.d2.value=seconds+"."+milisec  
  setTimeout("display()",100)  
}  
display()
```

3. Grabe el archivo en la carpeta js con nombre códigos.js
4. Cree un archivo html e ingrese el siguiente código

```
<!DOCTYPE html>  
<html lang="en">  
<head>
```

```
<title></title>
</head>
<body>
<tableborder="0">
<tbody>
<tr>
<td>
estas en esta pagina desde hace:
</td>
<td>
<form name="d">
<p><input name="d2" size="8" /></p>
</form>
</td>
<td>segundos</td>
</tr>
</tbody>
</table>
<script src="js/codigos.js"></script>
</body>
</html>
```

5. Grabe el archivo con el nombre segundosenpagina.html y luego lo ejecuta.

# Resumen

1. JavaScript es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java.
2. Las instrucciones javascript para solicitar datos son `prompt` y `confirm`. Las instrucciones para mostrar datos son `alert` y `document.write`.
3. Si el cuerpo del **if** o el cuerpo del **else** incluyen más de una acción, éstas deben ir encerradas entre llaves de bloque **{ }**.
4. Colocar un **;** al final de la condición de un **if** hace que la acción del **if** sea nula
5. Si un **case** no tiene **break**, sucederá que al ejecutar las acciones de dicho **case** se ejecutarán, también, las acciones de los **case** que siguen hasta encontrar un **break** o hasta llegar al final del **switch**
6. Se puede usar la estructura **switch** en una toma de decisiones únicamente si las condiciones consisten en comparaciones de una misma variable con una lista de constantes enteras o de carácter

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://msdn.microsoft.com/es-es/library/6974wx4d\(v=vs.94\).aspx](https://msdn.microsoft.com/es-es/library/6974wx4d(v=vs.94).aspx)
- <http://www.desarrolloweb.com/javascript/>
- <http://www.desarrolloweb.com/articulos/tipos-datos-variables-entrada-salida.html>
- <http://www.desarrolloweb.com/manuales/20/>
- [http://librosweb.es/libro/javascript/capitulo\\_3/estructuras\\_de\\_control\\_de\\_flujo.html](http://librosweb.es/libro/javascript/capitulo_3/estructuras_de_control_de_flujo.html)
-

## 2.2. ESTRUCTURAS DE REPETICIÓN

### 2.2.1. Sintaxis

#### 2.2.1.1. Estructura while

La estructura while permite crear bucles (instrucciones repetitivas) que se ejecutan una o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {  
    ...  
}
```

El funcionamiento del bucle while se resume en: *"mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle"*.

Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta. Si la condición se cumple, se ejecutan las instrucciones una vez y se vuelve a comprobar la condición. Si se sigue cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición no se cumpla.

Evidentemente, las variables que controlan la condición deben modificarse dentro del propio bucle, ya que de otra forma, la condición se cumpliría siempre y el bucle while se repetiría indefinidamente.

El siguiente ejemplo utiliza el bucle while para sumar todos los números menores o iguales que otro número:

```
var resultado = 0;  
var numero = 100;  
var i = 0;  
  
while(i <= numero) {  
    resultado += i;  
    i++;  
}  
  
alert(resultado);
```

El programa debe sumar todos los números menores o igual que otro dado. Por ejemplo si el número es 5, se debe calcular:  $1 + 2 + 3 + 4 + 5 = 15$

Este tipo de condiciones *"suma números mientras sean menores o iguales que otro número dado"* se resuelven muy fácilmente con los bucles tipo while, aunque también se podían resolver con bucles de tipo for.

En el ejemplo anterior, mientras se cumpla la condición, es decir, mientras que la variable i sea menor o igual que la variable numero, se ejecutan las instrucciones del bucle.

Dentro del bucle se suma el valor de la variable i al resultado total (variable resultado) y se actualiza el valor de la variable i, que es la que controla

la condición del bucle. Si no se actualiza el valor de la variable *i*, la ejecución del bucle continua infinitamente o hasta que el navegador permita al usuario detener el script.

### 2.2.1.2. Estructura do while

El bucle de tipo do...while es muy similar al bucle while, salvo que en este caso **siempre** se ejecutan las instrucciones del bucle al menos la primera vez. Su definición formal es:

```
do {  
    ...  
} while(condicion);
```

De esta forma, como la condición se comprueba después de cada repetición, la primera vez siempre se ejecutan las instrucciones del bucle. Es importante no olvidar que después del while() se debe añadir el carácter ; (al contrario de lo que sucede con el bucle while simple).

Utilizando este bucle se puede calcular fácilmente el factorial de un número:

```
var resultado = 1;  
var numero = 5;  
  
do {  
    resultado *= numero; // resultado = resultado * numero  
    numero--;  
} while(numero > 0);  
  
alert(resultado);
```

En el código anterior, el resultado se multiplica en cada repetición por el valor de la variable *numero*. Además, en cada repetición se decrementa el valor de esta variable *numero*. La condición del bucle do...while es que el valor de *numero* sea mayor que 0, ya que el factorial de un número multiplica todos los números menores o iguales que él mismo, pero hasta el número 1 (el factorial de 5 por ejemplo es  $5 \times 4 \times 3 \times 2 \times 1 = 120$ ).

Como en cada repetición se decrementa el valor de la variable *numero* y la condición es que *numero* sea mayor que cero, en la repetición en la que *numero* valga 0, la condición ya no se cumple y el programa se sale del bucle do...while.

### 2.2.1.3. Estructura for

La estructura for permite realizar bucles de manera sencilla. Su sintaxis es la siguiente:

```
for(inicializacion; condicion; actualizacion) {  
  
    ...  
}
```

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Ejemplo:

```
var mensaje = "Hola, estoy dentro de un bucle";
```

```
for(var i = 0; i < 5; i++) {  
    document.write(mensaje);  
}
```

La parte de la inicialización del bucle consiste en: `var i = 0;`

Por tanto, en primer lugar se crea la variable `i` y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es: `i < 5`

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable `i` valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable `i` se ha inicializado a un valor de 0 y la condición para salir del bucle es que `i` sea menor que 5, si no se modifica el valor de `i` de alguna forma, el bucle se repetiría indefinidamente. Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle: `i++`

En este caso, el valor de la variable `i` se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el `for`.

Así, durante la ejecución de la quinta repetición el valor de `i` será 4. Después de la quinta ejecución, se actualiza el valor de `i`, que ahora valdrá 5. Como la condición es que `i` sea menor que 5, la condición ya no se cumple y las instrucciones del `for` no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles `for` se llama `i`, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

Ejemplo:



```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",  
"Domingo"];  
  
for(var i=0; i<7; i++) {  
    document.write(dias[i]);  
}
```

#### 2.2.1.4. Estructura for in

La estructura Javascript **for in** nos permite recorrer una lista de elementos de una forma sencilla. Javascript **for in** es una estructura en bucle que nos permite tratar los elementos indicados en la sentencia.

La estructura Javascript **for in** tiene la siguiente sintaxis:

```
for (variable in objeto) {  
    // Acciones  
}
```

La estructura de control **for in** es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle.

Las sentencias **break** y **continue** permiten manipular el comportamiento normal de los bucles **for in** para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia **break** permite terminar de forma abrupta un bucle y la sentencia **continue** permite saltarse algunas repeticiones del bucle.

El siguiente ejemplo muestra el uso de la sentencia **break**:

```
var cadena = "Instituto Superior Tecnológico Privado Cibertec";  
var letras = cadena.split("");  
var resultado = "";  
  
for(i in letras) {  
    if(letras[i] == 'a') {  
        break;  
    }  
    else {  
        resultado += letras[i];  
    }  
}  
alert(resultado);  
// muestra "InstitutoSuperior Tecnológico Priv"
```

Si el programa llega a una instrucción de tipo **break**, sale inmediatamente del bucle y continúa ejecutando el resto de instrucciones que se encuentran fuera del bucle **for**. En el ejemplo anterior, se recorren todas las letras de una cadena de texto y cuando se encuentra con la primera letra "a", se detiene la ejecución del bucle **for**.

La utilidad de **break** es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

En ocasiones, lo que se desea es saltarse alguna repetición del bucle cuando se dan algunas condiciones. Siguiendo con el ejemplo anterior, ahora se desea que el texto de salida elimine todas las letras "o" de la cadena de texto original:

```
var cadena = "Instituto Superior Tecnológico Privado Cibertec";
var letras = cadena.split("");
var resultado = "";
```

```
for(i in letras) {
    if(letras[i] == 'o') {
        continue;
    }
    else {
        resultado += letras[i];
    }
}
```

```
alert(resultado);
```

```
// muestra "Institut Superir Tecnlgic Privad Cibertec"
```

En este caso, cuando se encuentra una letra "a" no se termina el bucle, sino que no se ejecutan las instrucciones de esa repetición y se pasa directamente a la siguiente repetición del bucle for.

La utilidad de continue es que permite utilizar el bucle for para filtrar los resultados en función de algunas condiciones o cuando el valor de alguna variable coincide con un valor determinado.

## 2.2.2. Ejemplos aplicados al desarrollo web

### Ejemplo1: Adjuntar múltiples ficheros a la vez

Este es el primer ejemplo completo donde se propone utilizar la manipulación del DOM mediante javascript con el objetivo de adicionar tantos elementos input del tipo file como tantos ficheros se deseen subir al servidor.

Se muestra una versión simplificada del problema limitada tan sólo al lado del cliente. Para ello es necesario imaginarse un sistema en línea donde se suben ficheros al servidor, ejemplo de ello podría ser una aplicación de correo electrónico.

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

<head>
<title>Ejemplo para adjuntar múltiples ficheros</title>
<script language="javascript" type="text/javascript">
    function nuevoFichero() {
        var input = document.getElementsByTagName("input")[0];
        var nuevoInput = input.cloneNode(true);
        input.parentNode.appendChild(nuevoInput);
    }
</script>
</head>
```

```
<body>
<form action="upload.php" method="post" enctype="multipart/form-data">
  <fieldset><legend>Adjuntar múltiples ficheros</legend>
  <input name="ficheros[]" type="file" size="60" >
  </fieldset>
  <a href="javascript: nuevoFichero();" >Adjuntar otro fichero</a>
  <input name="Subir" type="submit" value="Adjuntar" >
</form>
</body>

</html>
```

El enlace para adjuntar otro fichero hace una llamada a la función nuevoFichero() realizándose las siguientes tareas en la misma:

5. Se accede al primer elemento input encontrado en el documento:

```
var input = document.getElementsByTagName("input")[0];
```

6. Se crea un nuevo elemento input referenciado por la variable nuevoInput utilizando el método cloneNode resultando un elemento idéntico al primero:

```
var nuevoInput = input.cloneNode(true);
```

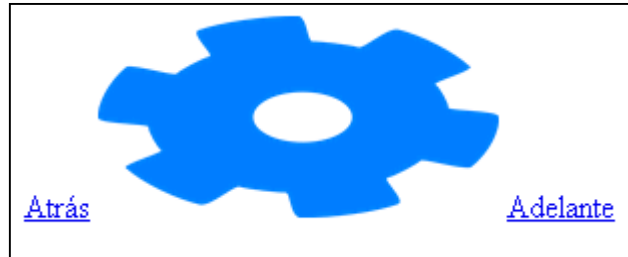
- Aquí se accede al nodo padre del elemento input mediante el método parentNode y la vez se inserta un nuevo elemento hijo copia del primero por medio del método appendChild:

```
input.parentNode.appendChild(nuevoInput);
```

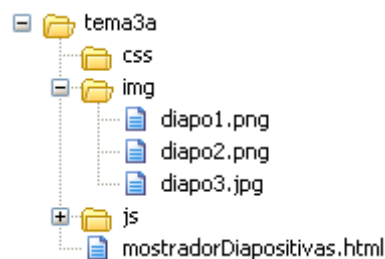
Es de notar que en este ejemplo podría haberse usado el evento clic para la llamada de la función, si embargo se dejó reservado para cuando el tema de la manipulación de eventos del DOM sea abordado.

## Laboratorio 3.1

En este laboratorio vamos a usar estructuras de selección para interactuar con el usuario. Se mostrará en la página el tiempo de permanencia del usuario expresado en segundos.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema3a y lo guarda en la carpeta Tema3a.
2. Cree una archivo js e inserte el siguiente código:

```
var conta = 0;
var imag = [];
function inicio() {
    var diapos = document.getElementById("diapositivas");
    while (diapos.childNodes.length > 0) {
        if (diapos.getElementsByTagName("img")[0]==diapos.firstChild) {
            imag[imag.length] = diapos.removeChild(diapos.firstChild);
        }
        else diapos.removeChild(diapos.firstChild);
    }
}
function adelante() {
    conta++;
    if (conta >= imag.length) conta = 0;
    ponerImagen()
}
function atras() {
    conta--;
    if (conta < 0 ) conta = imag.length - 1;
    ponerImagen();
}
function ponerImagen() {
    var diapos = document.getElementById("diapositivas");
    if (diapos.childNodes.length==0)
        diapos.appendChild(imag[conta]);
    else diapos.replaceChild(imag[conta],diapos.childNodes[0]);
}
```

```
inicio();  
ponerImagen();
```

3. Grabe el archivo en la carpeta js con el nombre códigos.js.
4. Cree un archivo html e inserte el siguiente código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Ejemplo de un mostrador de diapositivas</title>  
</head>  
<body>  
  
<a href="javascript: atras();" >Atr&aacute;s</a>  
<span id="diapositivas">  
  
  
  
</span>  
<a href="javascript: adelante();" >Adelante</a>  
<script src="js/codigos.js" type="text/javascript"></script>  
  
</body>  
</html>
```

5. Guarde el archivo con nombre mostradorDiapositivas.html.

Se tiene dos variables globales, *conta* y *imag*, la primera para indicar la diapositiva que se está mostrando actualmente y la última para almacenar en un arreglo el conjunto de diapositivas a mostrar.

La función *inicio()* remueve todos elementos *img* y los almacena. Es de notar que todo nodo hijo del nodo con *id* *diapositivas* irrelevantes (ejemplo: salto de línea) es eliminado para evitar un mal funcionamiento posterior.

Por otro lado la función *ponerImagen()* se encarga de colocar la imagen apuntada por el contador en el contenedor de diapositivas. Mientras que las funciones *atras()* y *adelante()* se encargan de decrementar e incrementar el contador respectivamente.

# Resumen

1. La estructura **while** es una estructura de propósito general que puede ser usada para resolver cualquier tipo de problema que involucre procesos repetitivos.
2. Si la condición del **while** resulta **falsa** la primera vez que se evalúa su condición de control, el **while** no efectuará ninguna iteración.
3. Si la condición del **while** no se hace **falsa** nunca, se genera un *bucle infinito*.
4. La estructura **for** es ideal para bucles en los que se conoce el número de iteraciones.
5. Si la condición del **for** resulta **falsa** la primera vez que se evalúa su condición de control, el **for** no efectuará ninguna iteración.
6. La estructura **do-while** evalúa su condición de control luego de ejecutar su cuerpo de acciones, a diferencia de la estructura **while** que, primero, prueba su condición de control y, luego, ejecuta su cuerpo de acciones.
7. El cuerpo de acciones de la estructura **do-while** se ejecutará por lo menos una vez, a diferencia de la estructura **while** que podría no ejecutar su cuerpo de acciones.

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.arkaitzgarro.com/javascript/capitulo-5.html>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Bucles\\_e\\_iteraci%C3%B3n](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Bucles_e_iteraci%C3%B3n)
- <http://www.desarrolloweb.com/articulos/567.php>
-

## 2.3. FUNCIONES EN EL LENGUAJE JAVASCRIPT

### 2.3.1. Introducción

Una función no es más que un bloque de enunciados que componen un comportamiento que puede ser invocado las veces que sea necesario.

### 2.3.2. Sintaxis general

Una función de JavaScript presenta este aspecto:

```
function nombre_de_la_función(){
    ...enunciados a ejecutar...
}
```

Para ejecutar la función posteriormente no hay más que invocar su nombre en cualquier momento y desde cualquier parte de un código, con una excepción: la función debe haber sido definida anteriormente. Así, por ejemplo, este código:

```
function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAAAAAAA!");
}

dame_una_a();
```

ejecutaría la alerta, pero éste:

```
dame_una_a();

function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAAAAAAA!");
}
```

Generaría un error, porque en el momento en que se invoca la función ésta aún no ha sido registrada.

Hay que poner especial atención a la hora de crear las funciones, para no repetir los nombres, principalmente porque esto no genera errores en JavaScript, y puede suponer quebraderos de cabeza cuando un script no funciona pero la consola de errores no muestra mensaje alguno.

Si definimos dos funciones con el mismo nombre, como en este ejemplo:

```
function mensaje(){
    alert("hola que tal");
}

function mensaje(){
    alert("hola que tal saludos a todos");
}
```

Sólo funciona la segunda, que ha sido la última definida.

En el siguiente ejemplo se muestra el uso de funciones para cambiar una imagen por otra al pasar el mouse encima.

Cuando ponemos el puntero del mouse en el enlace se cargará la imagen2 y cuando el puntero sale del enlace, regresará la imagen original imagen1.

```
<HTML>
<HEAD>
<TITLE>cambia imagen</TITLE>
<SCRIPT>
function cambiar () {
    document.images["modelo"].src = "imagen2.jpg";
}
function volver () {
    modelo.src = "imagen1.jpg";
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onMouseOver="cambiar();" onMouseOut="volver();" >Pasa el mouse
sobre mi y cambiará la imagen</A><BR><BR><BR><IMG src="imagen1.jpg"
NAME="modelo">
</BODY>
</HTML>
```

### 2.3.3. Funciones y argumentos

Podemos desear que una función ejecute unos enunciados en los que opere con una serie de valores que no hayamos definido dentro de la misma, sino que los reciba de otra función o enunciado. Esos valores son los argumentos o parametros, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas:

```
function nombre_de_la_función(argumento1,argumento1,...){
    ...enunciados a ejecutar...
}
```

Los argumentos se nombran como las variables:

```
function sumar(x,y){
    var total = x + y;
    alert(total);
}
```

Si después se ejecuta unas líneas como las siguientes:

```
sumar(1,2);
sumar(3,5);
sumar(8,13);
```

en la función sumar total adquiere sucesivamente los valores de 3, 8 y 21, que es lo que mostrarían tres alertas.



Aunque en los ejemplos emplee numerales, como argumentos se puede enviar cualquier variable. Sólo hay que recordar que si se trata de una cadena literal, debe ir entrecomillada:

```
la_función('cadena','otra_cadena');
```

Por último, sobre los argumentos hay que recordar las respuestas a tres preguntas:

- ¿Qué ocurre si se envía a una función menos argumentos que los que se han especificado?: Los argumentos que no han recibido un valor adoptan el de undefined.
- ¿Qué ocurre si se envía a una función más argumentos que los que se han especificado?: Los argumentos que sobran son ignorados.
- ¿Cuántos argumentos se pueden especificar como máximo?: 255.

Vamos a crear una función que realice cuatro operaciones aritméticas con los valores que introduzca un usuario. Se podría crear una función para cada operación, pero para simplificar el código es mejor crear una sola que obtenga los valores de los campos de formulario con los valores con los que operar, y que luego realice la operación elegida. ¿Y cómo sabe la función qué operación realizar? Eso es justo lo que pasamos como argumento.

La función sería ésta:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>funciones con argumentos</title>

<script>
function operar(x){
var valor_01 = eval(document.getElementById('campo_01').value);
var valor_02 = eval(document.getElementById('campo_02').value);
switch(x){
case('sumar'):
var resultado = valor_01 + valor_02;
break;
case('restar'):
var resultado = valor_01 - valor_02;
break;
case('multiplicar'):
var resultado = valor_01 * valor_02;
break;
case('dividir'):
var resultado = valor_01 / valor_02;
break;
}
document.getElementById('total').value = 'El resultado de '+x+' '+valor_01+' y '+valor_02+' es '+resultado+'.';
}
</script>
</head>
<body>
```

```
<textarea id="total"></textarea>
Ingrese un numero <input id="campo_01" />
Ingrese otro numero <input id="campo_02" />

<button onclick="operar('sumar');">+</button>
<button onclick="operar('restar');">-</button>
<button onclick="operar('multiplicar');">x</button>
<button onclick="operar('dividir');">/</button>

</body>
</html>
```

Hacer una redirección con JavaScript es muy sencillo. Podemos hacer que la redirección actúe justo al cargar la página o que actúe tras un cierto tiempo.

Redirección al cargar la página:

```
<body onLoad="document.location.href='http://www.cibertec.edu.pe'>
```

Si ponemos esto en nuestra etiqueta "body", cuando la página cargue, redireccionará a la página web de Cibertec.

Si nos interesa más una redirección que sea efectiva transcurridos unos segundos, podemos hacerlo así:

Entre <head> y </head> pondremos:

```
<script type="text/javascript">
    var pagina = 'http://www.cibertec.edu.pe';
    var segundos = 5;
    function redireccion() {
        document.location.href=pagina;
    }
    setTimeout("redireccion()",segundos);
</script>
```

Tan solo tenemos que cambiar las variables pagina y segundos, marcadas en negrita, para que redirija a la página que queramos y en el tiempo que queramos.

Lo que hacemos es crear un "timeout" que llame a la función "redireccion()" transcurridos tantos segundos como marque la variable "segundos". La función redirección lo único que hace es redirigir a la página que se indica en la variable "pagina".

#### 2.3.4. Variables y su ámbito

El ámbito sería algo así como el espacio en el que las variables existen y al que pueden acceder enunciados u otras funciones. Si se define una variable dentro de una función, esa variable sólo existe para esa función, y otras funciones no pueden acceder a su valor a menos que lo reciban como un argumento:

```
function concatenar_cadenas(){  
  
    var a = "Hola";  
    var b = "que tal";  
    var c = a + b;  
  
}  
  
function mostrar_resultado(){  
  
    alert(c);  
  
}  
  
concatenar_cadenas();  
mostrar_resultado();
```

En este caso no obtenemos una alerta con el texto “Hola que tal”, sino un error, puesto que en la función `mostrar_resultado()` pedimos que se muestre el valor de una variable que no existe en su ámbito. Dicho de otra manera, las variables existen sólo para `concatenar_cadenas()`, y aunque adquieren los valores definidos en cuanto ejecutamos la función, ésta es una “barrera” que impide que se pueda acceder a aquellos desde fuera.

Para que el código funcione, deberíamos definir las variables fuera de ambas funciones, para que su ámbito sea global:

```
var a = "";  
var b = "";  
var c = "";  
  
function concatenar_cadenas(){  
  
    a = "Hola";  
    b = "que tal";  
    c = a + b;  
  
}  
  
function mostrar_resultado(){  
  
    alert(c);  
  
}  
  
concatenar_cadenas();  
mostrar_resultado();
```

De esta forma, toda función puede acceder a las variables. `concatenar_cadenas()` modifica los valores iniciales, pero ahora estos se almacenan de manera global fuera de ella; así están disponibles para `mostrar_resultado()`.

### 2.3.5. Funciones con return

Una función con return es un módulo de programa que puede recibir datos de entrada a través de variables locales denominadas argumentos y que retorna un resultado al punto donde es invocado. Este tipo de función se utiliza para efectuar cualquier tipo de proceso que produzca un resultado.

La función anterior llamada concatenar\_cadenas, se puede modificar para que retorne un valor, de la siguiente manera:

```
function concatenar_cadenas(){
    var a = "orda";
    var b = "lía";
    var c = a + b;
    return c;
}

function mostrar_resultado(){
    alert(concatenar_cadenas());
}

mostrar_resultado();
```

#### Ejemplo 1: Deshabilita la selección de textos con el mouse

En el siguiente ejemplo, se muestra el uso de las funciones con return para deshabilitar la selección de textos con el mouse:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
</head>
<body>
Hola que tal, intenta seleccionar este texto
<script language="Javascript">
<!-- Begin
function disableselect(e){
return false
}
function reEnable(){
return true
}
document.onselectstart=new Function ("return false" )
if (window.sidebar){
document.onmousedown=disableselect
document.onclick=reEnable
}
// End -->
</script>
</body>
</html>
```

## Ejemplo 2: Mostrar fecha

Confeccionar una función que reciba una fecha con el formato de día, mes y año y retorne un string con un formato similar a: "Hoy es 10 de junio de 2013".

```
<html>
<head>
</head>
<body>
<script type="text/javascript">

function formatearFecha(dia,mes,año)
{
    var s='Hoy es '+dia+' de ';
    switch (mes) {
        case 1:s=s+'enero ';
            break;
        case 2:s=s+'febrero ';
            break;
        case 3:s=s+'marzo ';
            break;
        case 4:s=s+'abril ';
            break;
        case 5:s=s+'mayo ';
            break;
        case 6:s=s+'junio ';
            break;
        case 7:s=s+'julio ';
            break;
        case 8:s=s+'agosto ';
            break;
        case 9:s=s+'septiembre ';
            break;
        case 10:s=s+'octubre ';
            break;
        case 11:s=s+'noviembre ';
            break;
        case 12:s=s+'diciembre ';
            break;
    } //fin del switch
    s=s+'de '+año;
    return s;
}

document.write(formatearFecha(11,6,2013));

</script>
</body>
</html>
```

Analicemos un poco la función formatearFecha. Llegan tres parámetros con el día, mes y año. Definimos e inicializamos una variable con:

```
var s='Hoy es '+dia+' de ';
```

Luego le concatenamos o sumamos el mes:

```
s=s+'enero ';
```

Esto, si el parámetro mes tiene un uno. Observemos como acumulamos lo que tiene 's' más el string 'enero'. En caso de hacer s='enero ' perderíamos el valor previo que tenía la variable s.

Por último concatenamos el año:

```
s=s+'de '+año;
```

Cuando se llama a la función directamente, al valor devuelto se lo enviamos a la función write del objeto document. Esto último lo podemos hacer en dos pasos:

```
var fec= formatearFecha(11,6,2013);
document.write(fec);
```

Guardamos en la variable 'fec' el string devuelto por la función.

### Ejemplo 3: Cajas de dialogo personalizadas

En lugar de usar los cuadros de alerta con el método alert, se pueden crear cajas de dialogo personalizadas:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style>
#dialogoverlay{
    display: none;
    opacity: .8;
    position: fixed;
    top: 0px;
    left: 0px;
    background: #FFF;
    width: 100%;
    z-index: 10;
}
#dialogbox{
    display: none;
    position: fixed;
    background: #000;
    border-radius:7px;
    width:550px;
    z-index: 10;
}
#dialogbox > div{ background:#FFF; margin:8px; }
#dialogbox > div > #dialogboxhead{ background: #666; font-size:19px; padding:10px;
color:#CCC; }
#dialogbox > div > #dialogboxbody{ background:#333; padding:20px; color:#FFF; }
#dialogbox > div > #dialogboxfoot{ background: #666; padding:10px; text-align:right; }
</style>
<script>
function CustomAlert(){
    this.render = function(dialog){
```

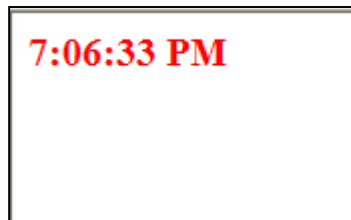
```

        var winW = window.innerWidth;
        var winH = window.innerHeight;
var dialogoverlay = document.getElementById('dialogoverlay');
    var dialogbox = document.getElementById('dialogbox');
dialogoverlay.style.display = "block";
    dialogoverlay.style.height = winH+"px";
    dialogbox.style.left = (winW/2) - (550 * .5)+"px";
    dialogbox.style.top = "100px";
    dialogbox.style.display = "block";
    document.getElementById('dialogboxhead').innerHTML = "Acknowledge This
Message";
document.getElementById('dialogboxbody').innerHTML = dialog;
document.getElementById('dialogboxfoot').innerHTML = '<button
onclick="Alert.ok()">OK</button>';
    }
        this.ok = function(){
            document.getElementById('dialogbox').style.display = "none";
            document.getElementById('dialogoverlay').style.display = "none";
        }
    }
var Alert = new CustomAlert();
</script>
</head>
<body>
<div id="dialogoverlay"></div>
<div id="dialogbox">
<div>
<div id="dialogboxhead"></div>
<div id="dialogboxbody"></div>
<div id="dialogboxfoot"></div>
</div>
</div>
<h1>My web document content ...</h1>
<h2>My web document content ...</h2>
<button onclick="alert('You look very pretty today.')">Default Alert</button>
<button onclick="Alert.render('You look very pretty today.')">Custom Alert</button>
<button onclick="Alert.render('And you also smell very nice.')">Custom Alert 2</button>
<h3>My web document content ...</h3>
</body>
</html>

```

## Laboratorio 4.1

En este laboratorio vamos a insertar un reloj digital en la página web mediante una función de javascript.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema4a y lo guarda en la carpeta Tema4a.
2. Cree un archivo js e inserte el siguiente código:

```
function mostrarclock(){
var nav=document.getElementById;
hora=nav?document.getElementById("reloj"):document.all.reloj
var Digital=new Date()
var hours=Digital.getHours()
var minutes=Digital.getMinutes()
var seconds=Digital.getSeconds()
var dn="PM"
if (hours<12)
dn="AM"
if (hours>12)
hours=hours-12
if (hours==0)
hours=12
if (minutes<=9)
minutes="0"+minutes
if (seconds<=9)
seconds="0"+seconds
var ctime=hours+":"+minutes+":"+seconds+" "+dn
hora.innerHTML="<b style='font-size:20px;'>" +ctime+"</b>"
setTimeout("mostrarclock()",1000)
}
mostrarclock()
```

3. Grabe el archivo en la carpeta js con el nombre códigos.js.
4. Cree un archivo html e inserte el siguiente código:



```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
</head>
<body>
  <span id="reloj"></span>
  <script src="js/codigos.js" type="text/javascript"></script>
</body>
</html>
```

5. Grabe el archivo con el nombre relojdigital.html y luego lo ejecuta en un navegador.

# Resumen

1. Una función no es más que un bloque de enunciados que componen un comportamiento que puede ser invocado las veces que sea necesario.
2. Para ejecutar la función posteriormente no hay más que invocar su nombre en cualquier momento y desde cualquier parte de un código, con una excepción: la función debe haber sido definida anteriormente.
3. Una función puede recibir argumentos o parametros, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas.
4. Si se define una variable dentro de una función, esa variable sólo existe para esa función, y otras funciones no pueden acceder a su valor a menos que lo reciban como un argumento o parametro.
5. Una función con return es un módulo de programa que puede recibir datos de entrada a través de variables locales denominadas argumentos y que retorna un resultado al punto donde es invocado.

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.desarrolloweb.com/articulos/583.php>
- <http://www.desarrolloweb.com/articulos/584.php>
- <http://www.desarrolloweb.com/articulos/585.php>
- [http://librosweb.es/libro/javascript/capitulo\\_4/funciones.html](http://librosweb.es/libro/javascript/capitulo_4/funciones.html)
-

## 2.4. ARREGLOS

### 2.4.1. Definición de arreglos

Una estructura típica en todos los lenguajes es el Array (arreglo), que es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con las variables normales.

Los arrays (arreglos) nos permiten guardar varias variables y acceder a ellas de manera independiente, es como tener una variable con distintos compartimentos donde podemos introducir datos distintos. Para ello utilizamos un índice que nos permite especificar el compartimiento o posición a la que nos estamos refiriendo.

### 2.4.2. Creacion de arreglos

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador. Veremos en adelante un tema para explicar lo que es la orientación a objetos, aunque no será necesario para poder entender el uso de los arrays. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array()
```

Esto crea un array en la página que esta ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el array Javascript especificando el número de compartimentos que va a tener.

```
var miArray = new Array(10)
```

En este caso indicamos que el array va a tener 10 posiciones, es decir, 10 casillas donde guardar datos.

Es importante que nos fijemos que la palabra Array en código Javascript se escribe con la primera letra en mayúscula. Como en Javascript las mayúsculas y minúsculas si que importan, si lo escribimos en minúscula no funcionará.

Tanto se indique o no el número de casillas del array javascript, podemos introducir en el array cualquier dato. Si la casilla está creada se introduce simplemente y si la casilla no estaba creada se crea y luego se introduce el dato, con lo que el resultado final es el mismo. Esta creación de casillas es dinámica y se produce al mismo tiempo que los scripts se ejecutan. Veamos a continuación cómo introducir valores en nuestros arrays.

```
miArray[0] = 290  
miArray[1] = 97  
miArray[2] = 127
```

Se introducen indicando entre corchetes el índice de la posición donde queríamos guardar el dato. En este caso introducimos 290 en la posición 0, 97 en la posición 1 y 127 en la 2.

Los arrays en Javascript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de

un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
var miArray = new Array(3)

miArray[0] = 155
miArray[1] = 4
miArray[2] = 499

for (i=0;i<3;i++){
    document.write("Posición " + i + " del array: " + miArray[i])
    document.write("<br>")
}
```

Hemos creado un array con tres posiciones, luego hemos introducido un valor en cada una de las posiciones del array y finalmente las hemos impreso. En general, el recorrido por arrays para imprimir sus posiciones, o cualquier otra cosa, se hace utilizando bucles. En este caso utilizamos un bucle FOR que va desde el 0 hasta el 2.

### Tipos de datos en los arrays

En las casillas de los arrays podemos guardar datos de cualquier tipo. Podemos ver un array donde introducimos datos de tipo carácter.

```
miArray[0] = "Hola"
miArray[1] = "a"
miArray[2] = "todos"
```

Incluso, en Javascript podemos guardar distintos tipos de datos en las casillas de un mismo array. Es decir, podemos introducir números en unas casillas, textos en otras, booleanos o cualquier otra cosa que deseemos.

```
miArray[0] = "desarrolloweb.com"
miArray[1] = 1275
miArray[1] = 0.78
miArray[2] = true
```

### Declaración e inicialización resumida de Arrays

En Javascript tenemos a nuestra disposición una manera resumida de declarar un array y cargar valores en un mismo paso. Fijémonos en el código siguiente:

```
var arrayRapido = [12,45,"array inicializado en su declaración"]
```

Como se puede ver, se está definiendo una variable llamada arrayRapido y estamos indicando en los corchetes varios valores separados por comas. Esto es lo mismo que haber declarado el array con la función Array() y luego haberle cargado los valores uno a uno.

A continuación se muestran los métodos de la clase array y algunos ejemplos de como pueden mejorar nuestro código notablemente.

**Eliminar**

pop: Elimina el ultimo elemento. Devuelve el elemento eliminado.

shift: Elimina el primer elemento. Devuelve el elemento eliminado.

**Agregar**

push: Agrega al final del arreglo uno o más elementos. Devuelve la nueva longitud.

unshift: Agrega uno o más elementos al inicio. Devuelve la nueva longitud.

splice: Agrega y/o elimina elementos.

**Orden**

reverse: Da vuelta el arreglo.

sort: Ordena los elementos.

**Unión**

concat: Une 2 arreglos.

join: Une los elementos en un string.

**Posición**

indexOf: Devuelve el índice del primer elemento encontrado.

lastIndexOf: Devuelve el índice del último elemento encontrado.

**Recorrer**

forEach: Ejecuta una función para cada elemento del arreglo.

**Ejemplos**

```
var arr = ['durazno', 'pera', 'manzana', 'banana', 'mandarina']
```

```
arr.pop()
```

Resultado: "mandarina" //el arreglo queda: ["durazno", "pera", "manzana", "banana"]

```
arr.shift()
```

Resultado: "durazno" //el arreglo queda: ["pera", "manzana", "banana"]

```
arr.push('naranja')
```

Resultado: 4 //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.unshift('kiwi')
```

Resultado: 5 //el arreglo queda: ["kiwi", "pera", "manzana", "banana", "naranja"]

```
arr.splice(0,1)
```

Resultado: ["kiwi"] //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.splice(2,2)
```

Resultado: ["banana", "naranja"] //el arreglo queda: ["pera", "manzana"]

```
arr.splice(2,0, "banana", "naranja")
```

Resultado: [] //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.splice(0,1, "kiwi")
```

Resultado: ["pera"] //el arreglo queda: ["kiwi", "manzana", "banana", "naranja"]

```
arr.reverse()
```

Resultado: ["naranja", "banana", "manzana", "kiwi"] //el arreglo queda: ["naranja", "banana", "manzana", "kiwi"]

```
arr.sort()
```

Resultado: ["banana", "kiwi", "manzana", "naranja"] //el arreglo queda: ["banana", "kiwi", "manzana", "naranja"]

```
arr.concat(['pera', 'pomelo'])
```

Resultado: ["banana", "kiwi", "manzana", "naranja", "pera", "pomelo"] //el arreglo queda: ["banana", "kiwi", "manzana", "naranja"]

```
arr.join()
```

Resultado: "banana,kiwi,manzana,naranja"

```
arr.indexOf('naranja')
```

Resultado: 3

```
arr.indexOf('kiwi')
```

Resultado: 1

El siguiente código de javascript muestra los índices y los valores de cada elemento del arreglo:

```
<script>
    var dias=["lunes","martes","miercoles","jueves","viernes","sabado","domingo"];
    dias.forEach(function(valorDelElemento,indiceDelElemento){
        document.write(indiceDelElemento+" es "+valorDelElemento+"<br>");
    });
</script>
```

El resultado sería así:

0 es lunes  
1 es martes  
2 es miercoles  
3 es jueves  
4 es viernes  
5 es sabado  
6 es domingo

Otra forma de usar el código anterior sería así:

```
<script>
    function mostrarDias(valorDelElemento,indiceDelElemento){
        document.write(indiceDelElemento+" es "+valorDelElemento+"<br>");
    }
    var dias=["lunes","martes","miercoles","jueves","viernes","sabado","domingo"];
    dias.forEach(mostrarDias);
</script>
```

Ejemplo 1: El siguiente código crea un arreglo de cadenas

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<title></title>
  <style>
    #miElemento{
      width:150px;
      height:250px;
    }
  </style>
</head>
<body>
  <div id="miElemento">Texto del div</div>
</body>
<script>
  function aplicarEstilos(elemento,listaEstilos){
    for(var estilo in listaEstilos)
      elemento.style[estilo]=listaEstilos[estilo];
  }
  var elemento=document.getElementById('miElemento');
  var estilos={'border-top':'solid 1px red',
               'background':'#FF0',
               'font-size':'17px',
               '-webkit-transform':'rotate(90deg)',
               'margin':'40px 50px',
               'position':'absolute'};
  aplicarEstilos(elemento,estilos);
</script>
</html>

```

Ejemplo 2: El siguiente código crea una animación tipo slide de texto:

```

<!DOCTYPE html>
<html>
<head>
<style>
#wss{
  opacity:0;
  -webkit-transition:opacity 1.0s linear 0s;
  transition:opacity 1.0s linear 0s;
}
</style>
<script>
var wss_i = 0;
var wss_array = ["Cute","Happy","<u>Playful</u>","Smart","Loyal"];
var wss_elem;
function wssNext(){
  wss_i++;
  wss_elem.style.opacity = 0;
  if(wss_i > (wss_array.length - 1)){
    wss_i = 0;
  }
  setTimeout('wssSlide()',1000);
}
function wssSlide(){
  wss_elem.innerHTML = wss_array[wss_i];
  wss_elem.style.opacity = 1;
}

```

```

        setTimeout('wssNext()',2000);
    }
</script>
</head>
<body>
<h1>My dog is <span id="wss"></span></h1>
<script>wss_elem = document.getElementById("wss"); wssSlide(); </script>
</body>
</html>

```

### 2.4.3. Trabajando con arreglo de objetos

Javascript tiene acceso a todos los elementos de una página web. Si los objetos son del mismo tipo se agrupan en una colección. Por ejemplo, si en un documento hay cinco párrafos, se puede almacenar en una variable llamada párrafos, los cinco objetos p (de la etiqueta html que representa a los párrafos), de la siguiente manera:

```
var cantidad=document.getElementsByTagName("p");
```

Todo arreglo tiene una propiedad llamada length que devuelve la cantidad de elementos que contiene el arreglo. Por lo tanto, el siguiente código devolverá 5

```
alert("cantidad de párrafos: "+cantidad.length);
```

Asi mismo, el siguiente código devuelve el texto del primer párrafo con la propiedad innerText:

```
alert("texto del párrafo 1: "+document.getElementsByTagName("p")[0].innerText);
```

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```

for(var i=0; i<parrafos.length; i++) {
var parrafo = parrafos[i];
}

```

La función getElementsByTagName() se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```

var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");

```

En el siguiente código se evalúa la cantidad de párrafos y enlaces que hay en un documento html:

```

<script type="text/javascript">
window.onload = function() {

```



```
// Numero de enlaces de la pagina
var enlaces = document.getElementsByTagName("a");

// Direccion del penultimo enlace
var penultimo = enlaces[enlaces.length-2];

// Numero de enlaces que apuntan a http://prueba
var contador = 0;
for(var i=0; i<enlaces.length; i++) {
// Es necesario comprobar los enlaces http://prueba y
// http://prueba/ por las diferencias entre navegadores
if(enlaces[i].getAttribute('href') == "http://prueba" || enlaces[i].getAttribute('href') ==
"http://prueba/") {
contador++;
}
}

// Numero de enlaces del tercer párrafo
var parrafos = document.getElementsByTagName("p");
enlaces = parrafos[2].getElementsByTagName("a");

alert("Numero de enlaces = "+enlaces.length+
"\nEl penultimo enlace apunta a: "+penultimo.getAttribute('href')+
"\n"+contador + " enlaces apuntan a http://prueba"+
"\nNumero de enlaces del tercer parrafo = "+enlaces.length);
}

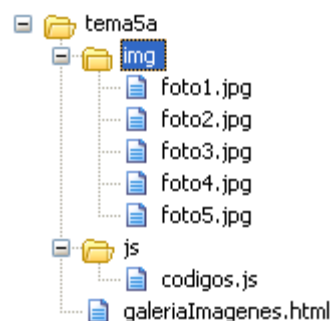
</script>
```

## Laboratorio 5.1

En este laboratorio vamos a insertar una galería de imágenes con arreglos



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes. Necesita 5 imágenes en la carpeta img para este ejercicio.



1. Antes de empezar, debe crear un sitio web con el nombre Tema5a y lo guarda en la carpeta Tema5a.

## 2. Cree un archivo js e inserte el siguiente código:

```
var foto = new Array();

foto[0] = "img/foto1.jpg";
foto[1] = "img/foto2.jpg";
foto[2] = "img/foto3.jpg";
foto[3] = "img/foto4.jpg";
foto[4] = "img/foto5.jpg";

var texto = new Array();
texto[0] = "la primera de las imagenes";
texto[1] = "esta es la segunda";
texto[2] = "esta es al tercera";
texto[3] = "y casi vamos a terminar";
texto[4] = "la ultima de las imagenes";

var cuolvemos = 0;

function mover(direccion) {
    // accedemos al objeto que contiene la imagen
    var laimagen = document.getElementById("misfotos");

    // accedemos al objeto que contiene el texto
    var eltexto = document.getElementById("mistextos");

    // ¿cual es el indice de la ultima imagen en nuestra array?
    var ultima = foto.length - 1; // en el ejemplo, sera el 4

    // antes de cambiar los datos, en un auxiliar, verificamos cual
    // sera la imagen a mostrar
    var auxiliar = cuolvemos + direccion; // se sumara 1 o se
    //restara 1 al indice
    // si el resultado es menor que cero, le decimos que vaya al
    // otro extremo y muestre la ultima
    if(auxiliar < 0) { auxiliar = ultima; }
    // si el resultado es mayor que la ultima, le decimos que
    //vaya al otro extremo y muestre la primera
    if(auxiliar > ultima) { auxiliar = 0; }

    // listo, ahora ya podemos cambiar el dato sin problemas
    cuolvemos = auxiliar;

    // ponemos la direccion URL de la imagen en la etiqueta IMG
    laimagen.src = foto[cuolvemos];

    // ponemos el texto en la etiqueta SPAN
    eltexto.innerHTML = texto[cuolvemos];
}
```

## 3. Grabe el archivo en la carpeta js con el nombre códigos.js

## 4. Cree un archivo html e inserte el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
    <script src="js/codigos.js" type="text/javascript">
    </script>
</head>
<body>
```

```
<div class="demoSW">
  <imgid="misfotos"src="img/foto1.jpg" />
  <span id="mistextos"> la primera de las imagenes </span>
  <div>
    <a href="javascript:mover(-1);"> anterior </a> |
    <a href="javascript:mover(1);"> siguiente </a>
  </div>
</div>
</body>
</html>
```

5. Grabe el archivo html con el nombre galerialimagenes.html y luego ejecute su pagina.

# Resumen

1. Un arreglo es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con las variables normales.
2. La sentencia para crear un objeto array vacío es: `var miArray = new Array()`.
3. Para crear el array especificando el número de compartimentos que va a tener es `var miArray = new Array(10)`.
4. En las casillas de los arrays podemos guardar datos de cualquier tipo.
5. Para declarar un array y cargar valores en un mismo paso se escribe:  
`var arrayRapido = [12.3, 57, "Hilda", "25/11/1990"];`

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.desarrolloweb.com/articulos/631.php>
- [https://msdn.microsoft.com/es-es/library/k4h76zbx\(v=vs.94\).aspx](https://msdn.microsoft.com/es-es/library/k4h76zbx(v=vs.94).aspx)
-

## 2.5. EXPRESIONES REGULARES

### 2.5.1. Definición de expresiones regulares

Las expresiones regulares son secuencia de caracteres que forman un patrón para encontrar o reemplazar una determinada combinación de caracteres dentro de una cadena de texto.

El encontrar una cadena o reemplazar una cadena por otra puede hacerse usando los métodos del objeto String, pero el problema surge cuando no tenemos una subcadena fija y concreta sino que queremos buscar un texto que responda a un cierto esquema, como por ejemplo: buscar aquellas palabras que comienzan con http: y finalizan con una \, o buscar palabras que contengan una serie de números consecutivos, etc.; es en estos casos cuando tenemos que utilizar las expresiones regulares.

En JavaScript, las expresiones regulares también son objetos. Estos patrones son utilizados a través de los métodos exec y test del objeto RegExp, así como los métodos match, replace, search y split del objeto String.

### 2.5.2. Creación de expresiones

La sintaxis básica para crear expresiones regulares es la siguiente:

/patron/modificadores;

Por ejemplo:

```
var patt = /w3schools/i
```

Donde:

/w3schools/i es la expresión regular.

w3schools es el patrón para ser usado en una búsqueda.

i es el modificador insensitive que significa que la búsqueda no considera mayúsculas o minúsculas.

Una expresión regular puede crearse de cualquiera de las siguientes dos maneras:

#### 1. Utilizando una representación literal de la expresión:

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

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

#### 2. Constructor del objeto RegExp:

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

El uso del constructor ofrece una compilación de la expresión regular en tiempo de ejecución. Su uso es recomendable en aquellos escenarios en que el patrón de la expresión regular pueda cambiar durante la ejecución del script, o bien, se

desconoce el patrón, dado que se obtiene desde otra fuente, cuando es suministrado por el usuario, por ejemplo.

### Modificadores

Los modificadores son usados para realizar búsquedas case-insensitive y en toda la cadena:

Modificador	Descripción
i	Realiza comparaciones case-insensitive
g	Realiza una búsqueda global (encuentra todas las coincidencias en lugar de detenerse después de encontrar la primera)
m	Realiza comparaciones en todas las líneas

### Corchetes

Los corchetes se usan para buscar un rango de caracteres:

Expression	Descripción
[abc]	Busca cualquier carácter indicado entre los corchetes
[^abc]	Busca cualquier carácter que no se encuentra entre los corchetes
[0-9]	Busca cualquier carácter dígito
[^0-9]	Busca cualquier carácter que no sea dígito
(x y)	Busca cualquier carácter de las alternativas especificadas

En la tabla que sigue se muestran los caracteres comodín usados para crear los patrones y su significado, junto a un pequeño ejemplo.

Significado	Ejemplo	Resultado
\ Marca de carácter especial	/\$ftp/	Busca la palabra \$ftp
^ Comienzo de una línea	/^-/	Líneas que comienzan por -

\$	Final de una línea	/s\$/	Líneas que terminan por s
.	Cualquier carácter (menos salto de línea)	/\b.\b/	Palabras de una sóla letra
	Indica opciones	/(L l f)ocal/	Busca Local, local, focal
( )	Agrupar caracteres	/(vocal)/	Busca vocal
[ ]	Conjunto de caracteres opcionales	/escrib[aoe]/	Vale escriba, escribo, escribe

La tabla que sigue describe los modificadores que pueden usarse con los caracteres que forman el patrón. Cada modificador actúa sobre el carácter o el paréntesis inmediatamente anterior.

Descripción	Ejemplo	Resultado
* Repetir 0 o más veces	/l*234/	Valen 234, 1234, 11234...
+ Repetir 1 o más veces	/a*mar/	Valen amar, aamar, aaamar...
? 1 o 0 veces	/a?mar/	Valen amar, mar.
{n} Exactamente n veces	/p{2}sado/	Vale ppsado
{n,} Al menos n veces	/(m){2}ala/	Vale mmala, mmmala....
{m,n} entre m y n veces	/tal{1,3}a/	Vale tala, talla, tallla

Los siguientes son caracteres especiales o metacaracteres para indicar caracteres de texto no imprimibles, como puedan ser el fin de línea o un tabulador, o grupos predefinidos de caracteres (alfabéticos, numéricos, etc...)

Significado	Ejemplos	Resultado
\b	Principio o final de palabra	Encuentra ver en "ver de", pero no en "verde"
\B	Frontera entre no-palabras	Empareja ver con "Valverde" pero no con "verde"
\d	Un dígito	No falla en "A4"
\D	Alfabético (no dígito)	Fallaría en "A4"
\O	Carácter nulo	
\t	Caracter ASCII 9 (tabulador)	



<b>\f</b>	Salto de página		
<b>\n</b>	Salto de línea		
<b>\w</b>	Cualquier alfanumérico, [a-zA-Z0-9_ ]	<b>\w+ /</b>	Encuentra <i>frase</i> en " <i>frase.</i> ", pero no el . (punto).
<b>\W</b>	Opuesto a \w ([^a-zA-Z0-9_ ])	<b>\W /</b>	Hallaría sólo el punto (.)
<b>\s</b>	Carácter tipo espacio (como tab)	<b>\sSi\s /</b>	Encuentra <i>Si</i> en " <i>Digo Si</i> ", pero no en " <i>Digo Sientate</i> "
<b>\S</b>	Opuesto a \s		
<b>\cX</b>	Carácter de control X	<b>\c9</b>	El tabulador
<b>\oNN</b>	Carácter octal NN		
<b>\xhh</b>	El hexadecimal hh	<b>\x41 /</b>	Encuentra la A (ASCII Hex41) en " <i>letra A</i> "

### 2.1.1. Manejo de caracteres especiales

Una expresión regular se construye con una cadena de texto que representa el formato que debe cumplir el texto. En javascript se puede hacer de dos formas, bien instanciando una clase RegExp pasándolo como parámetro la cadena de formato, bien poniendo directamente la cadena de formato, en vez de entre comillas, entre /

```
// Son equivalentes
varreg = new RegExp("aa")
varreg = /aa/
```

Ver si un texto tiene una determinada secuencia de letras o números fija es fácil. La expresión simplemente es esa cadena y podemos ver si un texto la tiene usando el método match() de ese texto

```
varreg = /javascript/;
"hola javascript".match(reg);

// devuelve un array de 1 elemento ["javascript"], indicando que sí existe esa
cadena dentro del texto

"adios tu".match(reg);

// devuelve null, indicando que no existe javascript dentro de "adios tu".
```

No es necesario definir la expresión regular antes, podemos hacerlo así

```
"hola javascript".match(/javascript/);  
  
// Devuelve ["javascript"]
```

Y para verificar si existe o no la cadena, podemos poner directamente un if

```
if("hola javascript".match(/javascript/)) {  
    // Pasará por aquí, porque un array con un elemento se evalúa como true en el if  
}  
  
if("adios tu".match(/javascript/)) {  
    // No pasa por aquí, null se evalúa como false en el if.  
}
```

### Caracteres no alfabéticos ni numéricos

Algunos de los caracteres no numéricos ni alfabéticos tienen un significado especial (lo vemos más adelante), como por ejemplo [ ] { } ( ) \* . ^ \$ etc. No podemos ponerlos tal cual si forman parte de nuestro formato, debemos "escaparlos" poniendo \ delante

```
"esto es un *".match(/\*/);  
  
// Devuelve ["*"] indicando que existe un asterisco.
```

### Conjunto opcional de caracteres

A veces nos da igual que una letra, por ejemplo, sea mayúscula o minúscula, o queremos que sea una vocal, o un dígito. Cuando queremos que una de las letras de nuestro texto pueda ser una cualquiera de un conjunto de letras determinado, las ponemos entre [] en nuestra expresión. Por ejemplo, si nos vale "Javascript" y "javascript", podemos poner la expresión como /[Jj]javascript/ para indicar que nos vale J mayúscula o j minúscula

```
"javascript con minuscula".match(/[Jj]javascript/);  
  
// Sí encuentra la cadena  
  
"Javascript con minuscula".match(/[Jj]javascript/);  
  
// También la encuentra.
```

Si los caracteres válidos son varios y van ordenados según el juego de caracteres, podemos poner el primero y el último separados por un -. Por ejemplo, [a-z] vale para cualquier letra minúscula, [0-9] para cualquier dígito y [a-zA-Z] para cualquier letra mayúscula o minúscula

```
"aa2bb".match(/[0-9]/); // Encuentra el 2, devolviendo ["2"]
```

Podemos hacer lo contrario, es decir, que la letra no esté en ese conjunto de caracteres. Se hace poniendo el juego de caracteres que no queremos entre [^ y ]. Por ejemplo, para no dígitos pondríamos [^0-9]

```
"22 33".match(/[^0-9]/); // Encuentra el espacio en blanco, devolviendo [" "]
```

### Conjuntos habituales

Hay varios conjuntos que se usan con frecuencia, como el de letras [a-zA-Z], el de dígitos [0-9] o el de espacios en blanco (espacio, tabulador, etc). Para estos conjuntos la expresión regular define formas abreviadas, como

\w	para letras, equivalente a [a-zA-Z]
\W	para no letras, equivalente a [^a-zA-Z]
\d	para dígitos, equivalente a [0-9]
\D	para no dígitos, equivalente a [^0-9]
\s	para espacios en blanco (espacios, tabuladores, etc).
\S	para no espacios en blanco.

Por ejemplo

```
"aa2bb".match(/\d/); // Encuentra el 2, devolviendo ["2"]
```

### Repetición de caracteres

Podemos querer buscar por ejemplo un conjunto de tres dígitos, podemos hacerlo repitiendo tres veces el \d

```
"aa123bb".match(/\d\d\d/); // Encuentra el 123, devolviendo ["123"]
```

Pero esta forma es un poco engorrosa si hay muchos caracteres y es poco versátil. Las expresiones regulares nos permiten poner entre {} un rango de veces que debe repetirse. Por ejemplo

```
\d{3}/    Busca 3 dígitos en la cadena
\d{1,5}/  Busca entre 1 y 5 dígitos en la cadena.
\d{2,}/   Busca 2 dígitos o más en la cadena.
```

Aplicando estas expresiones regulares tenemos:

```
"1234".match(/\d{2}/);
["12"]

"1234".match(/\d{1,3}/);
["123"]

"1234".match(/\d{3,10}/)
["1234"]
```

También suele haber rangos habituales como 0 o más veces, 1 o más veces, 0 ó 1 vez. Estos rangos habituales tienen caracteres especiales que nos permiten ponerlos de forma más simple.

```
*    equivale a 0 o más veces {0,}
+    equivale a 1 o más veces {1,}
?    equivale a 0 ó 1 vez {0,1}
```

Por ejemplo

```
"a2a".match(/a\d+a/);    // Encuentra a2a
"a234a".match(/a\d+a/);  // Encuentra a234a
```

Cosas como \* o + encuentran el máximo posible de caracteres. Por ejemplo, si nuestro patrón es /a+/ y nuestra cadena es "aaaaa", el resultado será toda la cadena

```
"aaaaa".match(/a+/); // Devuelve ["aaaaa"]
```

Para hacer que se encuentre lo menos posible, se pone un ? detrás. Así por ejemplo, si nuestra expresión regular es /a+?/ y nuestra cadena es "aaaaa", sólo se encontrará una "a"

```
"aaaaa".match(/a+?/); // Devuelve ["a"]
```

El comportamiento inicial se conoce como "greedy" o codicioso, en el que el patrón intenta coger la mayor parte de la cadena de texto posible. El segundo

comportamiento se conoce como "nongreedy" o no codicioso, en el que el patrón coge lo menos posible de la cadena.

### Extraer partes de la cadena

A veces nos interesa no sólo saber si una cadena cumple un determinado patrón, sino extraer determinadas partes de él. Por ejemplo, si una fecha está en el formato "27/11/2012" puede interesarnos extraer los números. Una expresión regular que vale para esta cadena puede ser:

```
\d{1,2}\d{1,2}\d{4}/
```

Suponiendo que el día y el mes puedan tener una cifra y que el año sea obligatoriamente de 4 cifras. En este caso:

```
"27/11/2012".match(\d{1,2}\d{1,2}\d{4}/);
```

Nos devuelve un array con un único elemento que es la cadena "27/11/2012". Para extraer los trozos, únicamente debemos poner entre paréntesis en la expresión regular aquellas partes que nos interesan. Es decir,

```
/(\d{1,2})\d{1,2}\d{4}/
```

Si ahora ejecutamos el método `match()` con la misma cadena anterior, obtendremos un array de 4 cadenas. La primera es la cadena completa que cumple la expresión regular. Los otros tres elementos son lo que cumple cada uno de los paréntesis

```
"27/11/2012".match(/(\d{1,2})\d{1,2}\d{4}/); // Devuelve el array  
["27/11/2012", "27", "11", "2012"]
```

Los paréntesis también nos sirven para agrupar un trozo y poner detrás uno de los símbolos de cantidades. Por ejemplo

```
"xyxyxyxy".match(/(xy)+/); // Se cumple, hay xy una o más veces.
```

### Usar lo encontrado en la expresión

Las partes de la cadena que cumplen la parte de expresión regular entre paréntesis, se pueden reutilizar en la misma expresión regular. Estas partes encontradas se van almacenando en `\1`, `\2`, `\3...` y podemos usarlas. Esta posibilidad es realmente interesante si queremos por ejemplo, verificar que una cadena de texto va cerrada

entre comillas del mismo tipo, es decir, queremos buscar algo como 'esto' o "esto", pero no nos vale 'esto'.

La expresión regular para buscar una cadena entre este tipo de comillas puede ser `/(["']).*\1/` es decir, buscamos una ' o una " con `["']`. Hacemos que lo que se encuentre se guarde metiéndolo entre paréntesis `(["'])` y a partir de ahí nos vale cualquier conjunto de caracteres terminados en `\1`, que es lo que habíamos encontrado al principio.

```
"'hola tu' tururú".match(/(["']).*\1/); // Devuelve ["'hola tu'", ""]  
"\hola tu' tururú".match(/(["']).*\1/); // Devuelve null, la cadena comienza con "y  
termina en '
```

### Ignorar lo encontrado

A veces nos interesa encontrar una secuencia que se repita varias veces seguidas y la forma de hacerlo es con los paréntesis, por ejemplo, si ponemos `/(pa){2}/` estamos buscando "papa". Para evitar que esos paréntesis guarden lo encontrado en `\1`, podemos poner `?:`, tal que así `/(?:pa){2}/`, de esta forma encontraremos "papa", pero se nos devolverá el trozo "pa" encontrado ni lo tendremos disponible en `\1`. Compara las dos siguientes:

```
"papa".match(/(pa){2}/); // Devuelve ["papa", "pa"]  
"papa".match(/(?:pa){2}/); // Devuelve ["papa"]
```

### Posición de la expresión

A veces nos interesa que la cadena busque en determinadas posiciones. Las expresiones regulares nos ofrecen algunos caracteres espaciales para esto.

`^` indica el principio de cadena, por ejemplo, `/^hola/` vale si la cadena "hola" está al principio

```
"hola tu".match(/^hola/); // Devuelve ["hola"]  
"pues hola tu".match(/^hola/); // Devuelve null
```

`$` es el final de la cadena, por ejemplo `/tu$/` vale si la cadena termina en "tu"

```
"hola tu".match(/tu$/); // Devuelve ["tu"]  
"hola tu tururú".match(/tu$/); // Devuelve null
```

\b indica una frontera de palabra, es decir, entre un caracter "letra" y cualquier otra cosa como espacios, fin o principio de linea, etc. De esta forma, por ejemplo, /\bjava\b/ buscará la palabra java, pero ignorará javascript

```
"java es güay".match(/\bjava\b/);    // Devuelve ["java"]  
"javascript es güay".match(/\bjava\b/); // Devuelve null
```

\B es lo contrario de \b, así por ejemplo, /\bjava\b/ buscará una palabra que empiece por "java", pero no sea sólo java sino que tenga algo más

```
"java es güay".match(/\bjava\b/);    // Devuelve null  
"javascript es güay".match(/\bjava\b/); // Devuelve ["java"]
```

(?=expresion) sirve para posicionar el resto de la expresión regular y buscar antes o después. Por ejemplo si queremos buscar un número que vaya delante de km, podemos hacer esto

```
\d+(?= km)/
```

Es decir, uno o más dígitos seguidos de un espacio y las letras km. La diferencia con esta expresión (\d+ km/) es que en el primer caso sólo casan con la expresión los números, mientras que en el segundo caso se incluye también el " km"

```
"11 millas 10 km".match(/\d+(?= km)/); // Devuelve ["10"]  
"11 millas 10 km".match(/\d+ km/);    // Devuelve ["10 km"]
```

Hay que tener cuidado si buscamos detrás, porque como el trozo (?=expresion) no se tiene en cuenta, sigue contando para el resto de la expresión. Por ejemplo, si queremos extraer la parte decimal de "11.22" podríamos pensar en hacer esto

```
/(?=\.)\d+/
```

Pero no funciona porque el . (punto) decimal no se "consume" con (?=\.), así que debemos tenerlo en cuenta y ponerlo detrás, así

```
/(?=\.)\.\d+/
```

Por ejemplo:

```
"11.22".match(/(?!=\.)\d+/); // Devuelve null  
"11.22".match(/(?!=\.)\.\d+/); // Devuelve [".22"]
```

(?!expresion) hace lo contrario que (?=expresion), es decir, busca una posición donde no se cumpla expresión. Por ejemplo, para sacar lo que no sean km de "11 km, 12 km, 14 m" podemos poner

```
\d{2}(?! km)/
```

Por ejemplo:

```
"11 km 12 km 14 m".match(\d{2}(?! km)/); // Devuelve ["14"]
```

### Flags de opciones

Hemos visto que una expresión regular es /expresion/. Podemos poner algunos flags detrás, básicamente unas letras que cambian algo el comportamiento

**i** es para ignorar mayúsculas y minúsculas

```
"hola".match(/HOLA/); // Devuelve null  
"hola".match(/HOLA/i); // Devuelve ["hola"]
```

**g** es para buscar todas las veces posibles la expresión, no sólo una vez

```
"11 223 44 66 77".match(/\d+/); // Devuelve ["11"]  
"11 223 44 66 77".match(/\d+/g); // Devuelve ["11", "223", "44", "66", "77"]
```

**m** busca en cadenas con retornos de carro \n considerando estos como inicios de línea ^ o fin \$

```
"hola\ntu".match(/^tu/); // Devuelve null  
"hola\ntu".match(/^tu/m); // Devuelve ["tu"]  
"hola\ntu".match(/hola$/); // Devuelve null  
"hola\ntu".match(/hola$/m); // Devuelve ["hola"]
```

### Otros métodos de cadena y de expresión regular

Para todos estos ejemplos hemos usado el método match() de la clase String, ya que nos devuelve un array con las cosas que se encuentran y viendo los resultados es la



forma más clara de ver cómo funcionan las distintas partes de la expresión regular. Sin embargo, tanto String como RegExp tienen otros métodos útiles

### **String.search(/expresion/)**

Devuelve la posición donde se encuentra esa expresión dentro de la cadena, o -1 si no se encuentra.

### **String.replace(/expresion/,cadena)**

Busca el trozo de cadena que casa con la expresión y la reemplaza con lo que le pasemos en el parámetro cadena. Este método tiene además un detalle interesante. Cuando en la expresión regular tenemos paréntesis para extraer algunas partes de la cadena, la misma expresión regular recuerda qué ha encontrado. En el método replace, si en la cadena de segundo parámetro aparecen cosas como \$1, \$2, utilizará lo encontrado.

```
"ho3la".replace(/d/,"X"); // Devuelve "hoXla"  
"ho3la".replace(/(\d)/,"-$1-"); // Devuelve "ho-3-la"
```

### **String.split(/expresion/)**

Usa lo que sea que case con la expresión como separador y devuelve un array con la cadena partida en trozos por ese separador

```
"hola, dos tres; cuatro".split(/W+/); // Devuelve ["hola", "dos", "tres", "cuatro"]
```

### **RegExp constructor**

Además de crear las expresiones regulares con /expresion/flags, podemos hacerlo con un new de la clase RegExp, por ejemplo new RegExp("expresion","flags").

```
varreg = newRegExp("\\d+","g");  
"11 22 33".match(reg); // Devuelve ["11","22","33"]
```

Hay que fijarse en este caso que las \ debemos escaparlas, con \\

### **RegExp.exec()**

Es similar a match() de String, pero sólo devuelve un resultado y hace que RegExp guarde la posición en la que lo ha encontrado. Sucesivas llamadas a exec(), nos irán devolviendo los siguientes resultados

```

varreg = newRegExp("\\d+");

reg.exec("11 22 33"); // Devuelve ["11"]
reg.exec("11 22 33"); // Vuelve a devolver ["11"], puesto que no hay flag g

varreg = newRegExp("\\d+", "g");

reg.exec("11 22 33"); // Devuelve ["11"]
reg.exec("11 22 33"); // Vuelve a devolver ["22"], puesto que si hay flag g
reg.exec("11 22 33"); // Vuelve a devolver ["33"], puesto que si hay flag g
reg.exec("11 22 33"); // Devuelve null, ya no hay más resultados.
reg.exec("11 22 33"); // Vuelve a devolver ["11"], despues de devolver null
                        la RegExp se "reinicializa"

```

### RegExp.test()

Similar a exec(), pero en vez de devolver lo encontrado, devuelve true si ha encontrado algo o false si no. Como la expresión regular recuerda las búsquedas anteriores, sucesivas llamadas a test() pueden devolver resultados distintos

```

varreg = newRegExp("\\d+", "g");

reg.test("11 22 33"); // Devuelve true, porque encuentra el ["11"]
reg.test("11 22 33"); // Vuelve a devolver true, porque encuentra el ["22"], puesto
                        que si hay flag g
reg.test("11 22 33"); // Vuelve a devolver true, porque encuentra el ["33"], puesto
                        que si hay flag g
reg.test("11 22 33"); // Devuelve false, ya no hay más resultados.
reg.test("11 22 33"); // Vuelve a devolver true, porque vuelve a encontrar el ["11"],
                        despues de devolver null la RegExp se "reinicializa"

```

### Ejemplo: Probar si existe una cadena

Vamos a comprobar si una cadena está contenida en otra cadena. Utilizar una expresión regular JavaScript para definir un patrón de búsqueda, y luego aplicar el patrón contra de la cadena a buscar, mediante el método RegExp. Haremos que coincida con cualquiera cadena que tenga las dos palabras Cook y Book en ese orden:

Solución.

```

var cookbookString = new Array();
cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript BookCook";

// patrón de búsqueda
var pattern = /Cook.*Book/;
for (var i = 0; i < cookbookString.length; i++)
    alert(cookbookString[i] + " " + pattern.test(cookbookString[i]));

```

La primera y tercera cadenas tiene un resultado positivo, mientras que la segunda y cuarta no.

El método de prueba RegExp toma dos parámetros: la cadena de prueba, y un modificador opcional. Se aplica la expresión regular con la cadena y devuelve true si hay una coincidencia, falso si no hay.

En el ejemplo, el patrón es la palabra Cook y la palabra Book, que aparecen en algún lugar en la cadena, la palabra Book después de la palabra Cook. Puede haber el número de caracteres que sea entre las dos palabras, incluyendo que no haya nada indicados por los caracteres especiales, punto (.) y el asterisco (\*)

El decimal en las expresiones regulares es un carácter especial que coincide con cualquier carácter excepto el carácter de nueva línea. En el patrón de ejemplo, el decimal es seguido por un asterisco, que coincide con el carácter anterior cero o más veces. Combinados, generan un patrón de coincidencia con cero o más de cualquier carácter, excepto una línea nueva.

### **Ejemplo: Encontrar y destacar todas las instancias de un patrón**

Quieres encontrar todas las instancias de un patrón dentro de una cadena. Para ello hay que utilizar el método exec en la expresión regular con el flag (g).

Vamos a decir como ejemplo cualquier palabra que empieza con t y termina con e y nos dará igual el número de caracteres entre ellos.

```
var searchString = "Now is the time and this is the time and that is the time";
var pattern = /tw*e/g;
var matchArray;
var str = "";
while((matchArray = pattern.exec(searchString)) != null) {
    str+="at " + matchArray.index + " we found " + matchArray[0] + " ";
}
document.getElementById("results").innerHTML=str;
```

El método exec de la expresión regular, devuelve null si no se encuentra la coincidencia, o un array con la información encontrada. En el array se encuentra el valor actual que ha encontrado, el índice de la cadena donde se encuentra la coincidencia, cualquier coincidencias de subcadenas entre paréntesis, y la cadena original.

### **Ejemplo: Reemplazar un patrón por una nueva cadena**

Para ello hay que usar el método replace del objeto String con una expresión regular.

```
var searchString = "Now is the time, this is the time";
var re = /tw{2}e/g;
var replacement = searchString.replace(re, "place");
alert(replacement); // Now is the place, this is the place
```

### **Ejemplo: Intercambio de palabras en una cadena usando paréntesis de captura**

Hay que usar paréntesis de captura y una expresión regular para encontrar y recordar los dos nombres de la cadena, y revertirlos.

En el ejemplo vamos a intercambiar el nombre de orden, poniendo el nombre al final y el apellido primero.

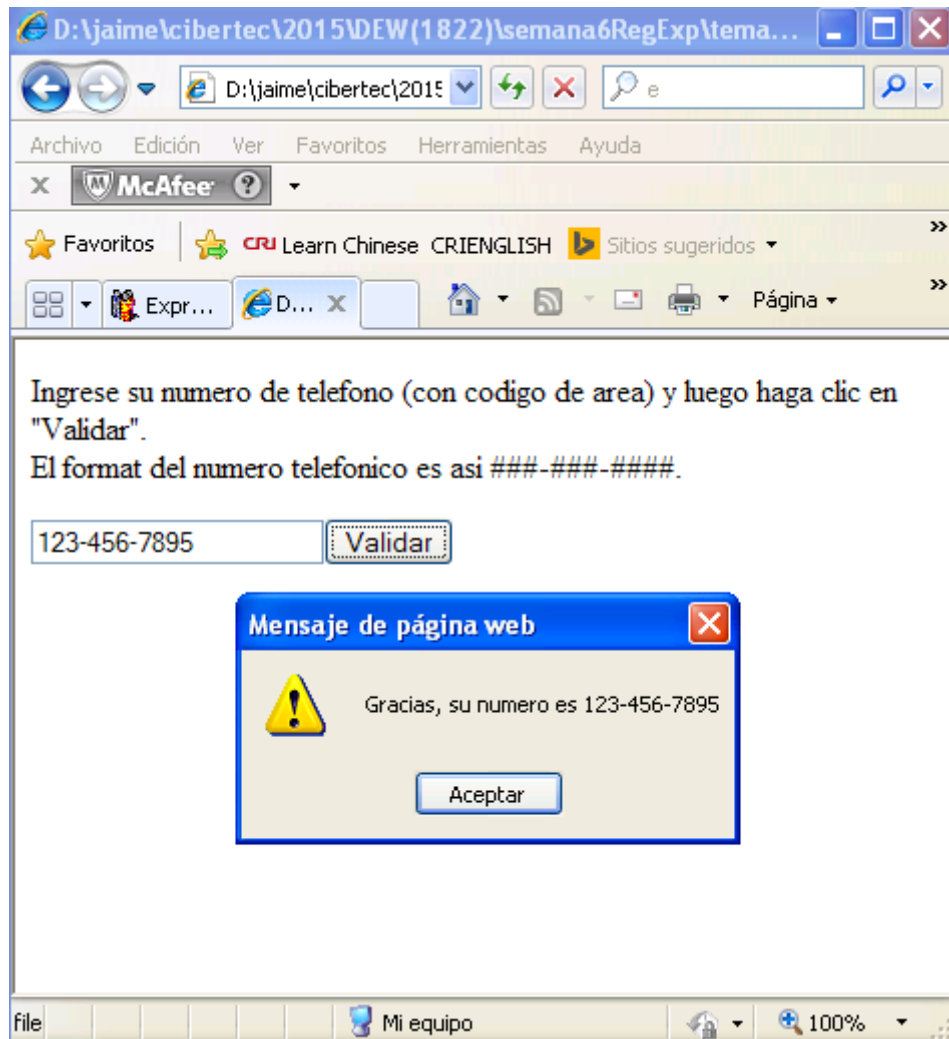
```
var name = "Pedro Ventura";  
var re = /^(w+)s(w+)$/;  
var newname = name.replace(re,"$2, $1");
```

Los paréntesis de captura nos permitirán no sólo para que coincida con los patrones preestablecidos en una cadena, sino que además hace referencia al subcadenas coincidentes. Las cadenas coincidentes son referenciadas numericamente de izquierda a derecha y son representadas con el uso de "\$1" y "\$2" en el método replace de String.

Como las dos palabras están separadas por un espacio, es fácil de crear la expresión regular, que recoge el nombre (las dos palabras), y el nombre será accesible usando "\$1" y el apellido con "\$2".

## Laboratorio 6.1

En el siguiente laboratorio, se espera que el usuario ingrese un número de teléfono. Cuando el usuario presiona el botón Validar, el código valida el número de teléfono. Si el número es correcto (es decir, cumple la secuencia de caracteres especificada por la expresión regular), el código muestra un mensaje agradeciendo al usuario y confirmando el número. De lo contrario, el código muestra un cuadro de alerta que indica que el número es equivocado.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema6a y lo guarda en la carpeta Tema6a.
2. Cree una archivo js e inserte el siguiente código:

```
var re = /(?:\d{3}|\(\d{3}\))([-\/\.]?)\d{3}\1\d{4}/;
function testInfo(phoneInput){
    var OK = re.exec(phoneInput.value);
    if (!OK)
        window.alert(RegExp.input + " no es un numero valido");
    else
        window.alert("Gracias, su numero es " + OK[0]);
}
```

3. Grabe el archivo en la carpeta js con el nombre `códigos.js`.
4. Cree un archivo html e ingrese el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
<script src="js/codigos.js" type="text/javascript"></script>
</head>
<body>
<p>Ingrese su numero de telefono (con codigo de area) y luego
    haga clic en "Validar".
<br>El format del numero telefonico es asi ###-###-####.
</p>
<form action="#">
<input id="phone">
    <button onclick="testInfo(document.getElementById('phone'));">
        Validar
    </button>
</form>
</body>
</html>
```

5. Guarde el archivo con el nombre `regex.html` y luego ejecute su pagina.

# Resumen

1. Las expresiones regulares son secuencia de caracteres que forman un patron para encontrar o reemplazar una determinada combinación de caracteres dentro de una cadena de texto.
2. En JavaScript, las expresiones regulares también son objetos. Estos patrones son utilizados a través de los métodos exec y test del objeto RegExp.
3. Las expresiones regulares son muy utiles para validar formularios.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://www.youtube.com/watch?v=93Ha7gd\\_gvo](https://www.youtube.com/watch?v=93Ha7gd_gvo)
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular\\_Expressions](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions)
- [http://www.w3schools.com/jsref/jsref\\_obj\\_regexp.asp](http://www.w3schools.com/jsref/jsref_obj_regexp.asp)
- [http://chuwiki.chuidiang.org/index.php?title=Expresiones\\_regulares\\_en\\_javascript](http://chuwiki.chuidiang.org/index.php?title=Expresiones_regulares_en_javascript)
-



# PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVASCRIPT

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript, usando el paradigma de programación orientada a objetos, diseña programas, incorporados en una página del Sitio Web

## TEMARIO

### 3.1 Tema 7 : Introducción

- 3.1.1 : Definición de programación orientada a objetos
- 3.1.2 : Definición de clases, Objetos y eventos
- 3.1.3 : Implementando el paradigma de la POO en páginas web

### 3.2 Tema 8 : Clases predefinidas

- 3.2.1 : Array
- 3.2.2 : Date
- 3.2.3 : Math
- 3.2.4 : String

## ACTIVIDADES PROPUESTAS

- Los alumnos diseñan una clase con sus propiedades y funciones necesarias y lo implementan en una página web.
- Los alumnos crean objetos mediante una instancia de clase.
- Los alumnos asignan eventos a cada elemento de una lista.
- Los alumnos crean una aplicación, usando namespace, que valida los campos de un formulario.
- Los alumnos crean un desplazamiento de texto con la clase String
- Los alumnos insertan textos animados que se desplazan en la página
- Los alumnos, usando clases predefinidas, insertan la fecha y la hora en una página web.



## 3.1 INTRODUCCIÓN

### 3.1.1 Definición de programación orientada a objetos

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento. Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos

La programación orientada a objetos puede considerarse como el diseño de software a través de un conjunto de objetos que cooperan, a diferencia de un punto de vista tradicional en el que un programa puede considerarse como un conjunto de funciones, o simplemente como una lista de instrucciones para la computadora. En la programación orientada a objetos, cada objeto es capaz de recibir mensajes, procesar datos y enviar mensajes a otros objetos. Cada objeto puede verse como una pequeña máquina independiente con un papel o responsabilidad definida.

### 3.1.2 Definición de clases, objetos y eventos

#### 3.1.2.1. Clases

La programación basada en prototipos es un estilo de programación orientada a objetos en la que las clases no están presentes y la reutilización de comportamiento (conocido como herencia en lenguajes basados en clases) se lleva a cabo a través de un proceso de decoración de objetos existentes que sirven de prototipos. Este modelo también se conoce como programación sin clases, orientada a prototipos o basada en ejemplos.

JavaScript es un lenguaje basado en prototipos que no contiene ninguna declaración de clase, como se encuentra, por ejemplo, en C++ o Java. Esto es a veces confuso para los programadores acostumbrados a los lenguajes con una declaración de clase. En su lugar, JavaScript utiliza funciones como clases. Definir una clase es tan fácil como definir una función. En el ejemplo siguiente se define una nueva clase llamada Persona.

```
function Persona() { }
```

Una clase es la definición de un objeto. Esta clase tiene propiedades, funciones y eventos. Por ejemplo, la clase Persona se crea de esta manera:

```
function Persona(nombre) {  
    this.nombre = nombre;  
    this.color_pelo = 'negro';  
    this.peso = 75;  
    this.altura = 165;  
    this.sexo = 'varón';  
    this.edad = 26;  
}
```

Vamos a fijarnos bien como se estructura esta función. Se le llama constructor de la clase, y en ella definimos los datos de la clase, los que vamos a poder utilizar al crear objetos con ella. Nótese el uso de la palabra reservada `this`. Esta palabra sirve para identificar al objeto en si mismo dentro de la definición de la clase. Cuando escribimos

```
this.peso = 75;
```

estamos creando la propiedad "peso" de la clase "Persona". Cuando creamos una propiedad en el constructor y le damos un valor, como en este caso, estamos asignándole un valor por defecto. Todos los objetos creados con este constructor contendrán una propiedad "peso" con ese valor inicial, aunque luego podremos cambiarlo al usar el objeto.

### 3.1.2.2. Objetos

Un objeto es la instancia (copia o ejemplar) de una clase. Para definir un objeto de esta clase Persona, sólo tendríamos que llamar a su constructor (function Persona) precedido de la palabra reservada `new` y pasarle entre los paréntesis algún parámetro o dato que necesite la función llamado también constructor:

```
var hombre = new Persona('Pepe');
```

Aquí hemos definido el objeto "hombre", que contendrá todas las propiedades definidas en la función-clase "Persona". Si queremos cambiar su valor, sólo tenemos que hacer algo como esto:

```
hombre.peso = 80;
```

De esta forma, el dato definido para este objeto cambia. Pero si hemos definido más objetos de tipo Persona, cada uno de ellos contendrá las mismas propiedades pero con valores distintos. Ningún objeto tiene el mismo valor que otro objeto de la misma clase a no ser que nosotros se lo asignemos explícitamente.

```
var mujer = new Persona('Ana');  
mujer.peso = 67;  
mujer.sexo = 'mujer';
```

En este caso hemos hecho lo mismo, pero le indicamos su propio peso, independiente del de la variable "hombre". Así, podemos tener tantos objetos de la misma clase como queramos para realizar las operaciones que sean pertinentes. Una última cosa sobre los constructores: como podemos ver, podemos pasarle parámetros, que podemos convertir en los valores de las propiedades de los objetos de esa clase.

### Creación de funciones miembro

Hasta ahora hemos visto como crear propiedades de las clases, pero necesitamos crear código en ese objeto que utilice las propiedades que hemos creado en el constructor. Para crear una función miembro, debemos indicarlo en la propia función de construcción:

```
function Persona(nombre) {  
    this.nombre = nombre;  
    this.color_pelo = 'negro';  
    this.peso = 75;  
    this.altura = 165;  
    this.sexo = 'varón';  
    this.dormir = dormir; // Nueva función miembro  
}
```

Y ahora definimos la función dormir:

```
function dormir() {  
    alert(this.nombre + ' está durmiendo');  
}
```

Fijémonos en la función. Tiene una forma bastante normal. Lo único especial que hemos hecho es añadir la línea

```
this.dormir = dormir;
```

al constructor, con lo que hemos asignado la función dormir como si fuera una propiedad. Recordemos que TODO es un objeto en JavaScript, y esto incluye a las funciones. Ahora, para ejecutar este código, utilizamos el objeto anteriormente creado para ponerlo en marcha:

```
hombre.dormir();
```

Veamos en un ejemplo todo el código que hemos generado hasta ahora:

```
<html>  
<head>  
<script language="javascript">  
function Persona(nombre) {  
    this.nombre = nombre;  
    this.color_pelo = 'negro';  
    this.peso = 75;  
    this.altura = 165;  
    this.sexo = 'varón';  
    this.dormir = dormir;  
}  
  
function dormir() {  
    alert(this.nombre + ' está durmiendo');  
}  
</script>  
</head>  
  
<body>  
<form>  
</form>  
<script>  
    var hombre = new Persona('Pepe');  
    hombre.dormir();  
</script>  
</body>
```

</html>

Como resultado, nos mostrará el mensaje "Pepe está durmiendo". Como vemos, podemos usar las propiedades de los objetos dentro de las funciones miembro, aunque también podríamos construir la misma función de otra manera:

```
function dormir() {  
    with (this)  
        alert(nombre + ' está durmiendo');  
}
```

with es una palabra reservada de JavaScript que permite coger una variable de objeto como this y permite utilizar sus miembros como si fueran variables independientes. Pero tiene sus restricciones: estos nombres abreviados sólo se pueden utilizar dentro del ámbito de with (que si tiene varias líneas, estas deben estar contenidas entre llaves, como for, if, etc.), y además, se pueden confundir fácilmente con variables locales a la función o globales del programa, con lo cual particularmente no recomendamos el uso de with, ya que puede dar lugar a fallos de ejecución difíciles de tratar si no se tienen en cuenta estas restricciones. Se aconseja usar la forma this.nombre. También se recomienda crear cada clase en un archivo diferente para que no haya confusiones de nombres, sobre todo de funciones miembro.

Otra manera de declarar la clase en JavaScript:

```
<html>  
<head>  
<script language="javascript">  
function Persona(nombre) {  
    this.nombre = nombre;  
    this.color_pelo = 'negro';  
    this.peso = 75;  
    this.altura = 165;  
    this.sexo = 'varón';  
    this.dormir = function dormir(){  
        alert(this.nombre + ' está durmiendo');  
    }  
}  
</script>  
</head>  
<body>  
<form>  
</form>  
<script>  
    var hombre = new Persona('Pepe');  
    hombre.dormir();  
</script>  
</body>  
</html>
```

Con este ejemplo se obtiene el mismo resultado que el anterior pero el código queda un poco más complejo. A pesar de esto ya podemos ver que a diferencia

del código anterior este se encuentra encapsulado en la misma función [ `function Persona(){} ]`

Vamos a crear un objeto persona con los atributos “*nombre y apellido*” y el método “*presentar*”.

```
function Persona(nombre, apellido){
    this.nombre=nombre;
    this.apellido=apellido;
}

Persona.prototype={
    presentar:function(){
        console.log("Hola, me llamo "+this.nombre+" "+this.apellido);
    }
};

var p=new Persona("John","Doe");
p.presentar();
```

Vamos a crear un objeto de dos formas, la primera usando un *Literal de Objeto*:

```
var pedroPerez = {
    nombre:'Pedro Pérez',
    nacimiento: new Date(1982, 7, 22),
    casado: false,
    hijos: 0,
    calcularEdad: function(){
        return Math.floor((new Date() - this.nacimiento)/1000/3600/24/365);
    }
};
```

La segunda forma consiste en aprovechar el *Dinamismo* de JavaScript, creando un objeto “vacío” al que le agregamos las propiedades y funciones que necesitemos:

```
var pedroPerez = {}
pedroPerez.nombre = 'Pedro Pérez';
pedroPerez.nacimiento = new Date(1982, 7, 22);
pedroPerez.casado = false;
pedroPerez.hijos = 0;
pedroPerez.calcularEdad = function(){
    return Math.floor((new Date() - this.nacimiento)/1000/3600/24/365);
}
```

En este caso, cuando asignamos un valor a una propiedad de objeto que no existe, ésta se crea inmediatamente para almacenar el nuevo valor.

La primera forma, usando un *Literal de Objeto*, es preferible en lugar de la segunda. La razón es que algunos motores JavaScript (como V8 en Google

Chrome o SpiderMokey en Firefox) intentan optimizar la ejecución del código, y una de las formas de hacerlo consiste en detectar aquellos objetos *que* cambien poco en el código para almacenarlos en memoria de una forma más eficiente. Cuando el motor JavaScript detecta que el objeto es modificado con cierta frecuencia (agregándole nuevas propiedades o borrándolas con Delete), es más difícil de optimizar haciendo el código menos eficiente al momento de ejecutarlo.

### 3.1.2.3. Eventos

Un evento es un proceso que se realiza en respuesta a determinada acción realizada por el usuario. Los eventos son códigos en Javascript que controlan las acciones de los visitantes y definen un comportamiento de la página cuando se produzcan. Cuando un usuario visita una página web e interactúa con ella se producen los eventos y con Javascript podemos definir qué queremos que ocurra cuando se produzcan.

Con javascript podemos definir qué es lo que pasa cuando se produce un evento como podría ser que un usuario pulse sobre un botón, edite un campo de texto o abandone la página.

El manejo de eventos es el caballo de batalla para hacer páginas interactivas, porque con ellos podemos responder a las acciones de los usuarios. Hasta ahora en este manual hemos podido ver muchos ejemplos de manejo de uno de los eventos de Javascript, el evento onclick, que se produce al pulsar un elemento de la página. Hasta ahora siempre hemos aplicado el evento a un botón, pero podríamos aplicarlo a otros elementos de la página.

#### Cómo se define un evento

Los detectores o manejadores de eventos se pueden asociar directamente a cada elemento como si fuera una propiedad adicional, además debemos colocar que acción realizar cuando se detecte el evento.

##### Primera Forma

Se puede hacer que ejecute una serie de acciones en Javascript:

```
<a href="link.html" onclick="alert('Bienvenido!!!');">Entrar</a>
```

##### Segunda Forma

También podemos hacer que llame a una función definida por el usuario:

```
<a href="link.html" onclick="welcome();">Entrar</a>
```

En este caso estamos llamando a una función llamada welcome la cual debemos definir:

```
<script type="text/javascript">
function welcome() {
    alert("Welcome!!!");
}
```

```
}  
</script>
```

En el ejemplo solo hemos mostrado un mensaje de alerta dando la bienvenida al usuario. Obviamente podríamos realizar múltiples acciones como validación de formulario, formateo de datos o llamar a datos AJAX.

### Tercera Forma

Otra forma es asignar los eventos como métodos de los elementos Javascript. Para ello necesitamos asignar identificadores a los elementos que deseamos.

```
<a href="link.html" id="bt">Entrar</a>
```

Luego podemos agregar los eventos como propiedades del elemento Javascript (Hay que tener en cuenta que en esta forma los nombres de los detectores de eventos deben estar todos en minúsculas).

```
<script type="text/javascript">  
document.getElementById("bt").onclick = function () {  
    alert("Welcome!!!");  
}  
</script>
```

### Cuarta Forma

Otra forma de crear los detectores de eventos es haciendo uso de listeners, para ello se utiliza la función `addEventListener` de Javascript, el cual recibe como parámetros el nombre del evento (Sin en prefijo `on`) y la función a ejecutar. Para nuestro ejemplo se tendría:

```
<script type="text/javascript">  
document.getElementById("bt").addEventListener('click', welcome, false);  
function welcome() {  
    alert("Welcome!!!");  
}  
</script>
```

Como se puede ver utilizamos el evento `onclick` pero sin el prefijo lo que significa utilizar la palabra `'click'`.

Con estas dos últimas formas de asignar los detectores de eventos podemos separar el código HTML del Javascript con lo cual es más sencillo dar el mantenimiento a nuestro código. Incluso la asignación de los eventos lo podríamos hacer en un archivo externo.

Todo manual moderno de JavaScript que merezca la pena comienza diciendo que escribir líneas como estas:

```
<a href="javascript:función_que_sea()">Un vínculo</a>  
<a href="función_que_sea()">Un vínculo</a>  
<a href="#" onclick="función_que_sea()">Un vínculo</a>
```

es una pésima idea. ¿Por qué? Bueno, pues porque si el agente de usuario de nuestra visita no soporta JavaScript o no lo tiene activado, la funcionalidad añadida por medio del script simplemente no funciona.

Un paso más allá es crear un marcado como el siguiente:

```
<a href="página_con_funcionalidad_alternativa_no_dependiente_de_JS.htm"
onclick="función_que_sea();return false;">Un vínculo</a>
```

Sin embargo, si queremos acercarnos lo más posible al ideal de separar las capas de contenido, presentación y comportamiento de un documento, deberíamos mantener el marcado limpio de atributos como onclick, que pertenece propiamente al comportamiento.

La pregunta es: ¿cómo hacer esto de manera que las mejoras para la experiencia del usuario que permite JavaScript sigan siendo efectivas? Y la respuesta es: escuchando.

En el nivel 2 del DOM se define el método `addEventListener` (inglés) que nos permite precisamente lo que hemos dicho arriba: indicar al agente de usuario que permanezca atento a la interacción de un usuario sobre un elemento en concreto, sin necesidad de tocar un sólo carácter de nuestro marcado.

La sintaxis de `addEventListener` es muy sencilla:

```
elemento_que_se_escucha.addEventListener('evento',función_a_lanzar,booleano);
```

Veámoslo con más detalle:

**elemento\_que\_se\_escucha** es cualquier elemento presente en un documento, al que accedemos por el medio que elijamos, bien por su id, por su etiqueta o las propiedades de otro nodo.

**evento** es el suceso ocurrido sobre el elemento, con o sin interacción del usuario, como vimos en la sección de sintaxis de JavaScript relativa a los eventos.

**función\_a\_lanzar** es cualquier función definida que queramos que se ejecute cuando ocurra el evento.

**booleano** es un valor que define el orden del flujo de eventos, algo que veremos un poco más abajo.

Pongamos un ejemplo. Si tuviéramos un formulario y quisiéramos que éste se validase antes de ser enviado al servidor, la forma errónea de hacerlo sería ésta:

```
<button onclick="validar()">Enviar formulario</button>
```

Mediante una escucha, en el marcado tendríamos:

```
<button type="submit" id="enviar">Enviar formulario</button>
```

y en el script:



```
document.getElementById('enviar').addEventListener('click', validar, false);
```

Mucho más limpio, y además nos aseguramos de que si JavaScript no está disponible, el formulario podrá ser enviado.

¿Y qué ocurre si necesito escuchar un evento sólo una vez? Para ese caso existe un método complementario de `addEventListener`, que es `removeEventListener`:

```
elemento_que_se_escuchaba.removeEventListener('evento', función_a_anular,
booleano);
```

En los ejemplos he escuchado el evento `click`, pero la mecánica es la misma para cualquier otro, siempre que el elemento acepte el evento.

### El flujo de los eventos: captura y burbuja

Como hemos visto, hay un parámetro en `addEventListener` que es un booleano. ¿Para qué sirve? Bueno, para entenderlo primero hemos de saber lo que es el flujo de eventos.

Supongamos que tenemos este marcado:

```
<body>
<div>
<button>HAZME CLIC</button>
</div>
</body>
```

Cuando hacemos clic en el botón no sólo lo estamos haciendo sobre él, sino sobre los elementos que lo contienen en el árbol del DOM, es decir, hemos hecho clic además sobre el elemento `body` y sobre el elemento `div`. Sí sólo hay una función asignada a una escucha para el botón no hay mayor problema, pero si además hay una para el `body` y otra para el `div`, ¿cuál es el orden en que se deben lanzar las tres funciones?

Para contestar a esa pregunta existe un modelo de comportamiento, el flujo de eventos. Según éste, cuando se hace clic sobre un elemento, el evento se propaga en dos fases, una que es la captura —el evento comienza en el nivel superior del documento y recorre los elementos de padres a hijos— y la otra la burbuja —el orden inverso, ascendiendo de hijos a padres—.

Así, el orden por defecto de lanzamiento de las supuestas funciones sería `body-div-button`.

Una vez visto esto, podemos comprender el tercer parámetro de `addEventListener`, que lo que hace es permitirnos escoger el orden de propagación:

**true:** El flujo de eventos es como el representado, y la fase de captura ocurre al lanzarse el evento. El orden de propagación para el ejemplo sería, por tanto, el indicado, `body-div-button`

**false:** Permite saltar la fase de captura, y la propagación seguiría sólo la burbuja. Así, el orden sería `button-div-body`.

Un ejemplo con el valor true, y otro con el valor false.

En este punto ya podemos crear comportamientos sin ensuciar nuestro marcado con atributos relativos al comportamiento. Sin embargo, ni Internet Explorer 7 ni ninguno de sus predecesores soportan addEventListener. Lo que sí soportan las versiones 6 y 7 son dos métodos propietarios similares: attachEvent y detachEvent:

```
elemento_que_se_escucha.attachEvent('manejador_de_evento',función_a_lanzar);
```

```
elemento_que_se_escuchaba.detachEvent('manejador_de_evento',función_a_anular);
```

Como se ve, su sintaxis es muy similar a las de addEventListener y removeEventListener, salvo por dos detalles:

No se hace referencia al evento, sino al manejador de evento, que sin meternos en cuestiones técnicas es lo mismo pero con un prefijo on; así, al evento click le corresponde el manejador onclick, a mouseover onmouseover, y así sucesivamente.

No hay un tercer parámetro: en Explorer el flujo se identifica con la burbuja, y no hay fase previa de captura.

Así, siempre que se quieran establecer escuchas, habrá que emplear unos métodos para navegadores con un soporte de los métodos de escucha estándar, y los otros para Explorer. Éste es el motivo por el que todos antes o después todos empezamos a utilizar una función para establecer escuchas:

```
function ev (x,y,z) {  
    if (document.addEventListener){  
        x.addEventListener(y,z,false);  
    } else {  
        x.attachEvent('on'+y,z);  
    }  
}
```

donde x es el elemento sobre el que se quiere establecer la escucha, y el evento y z la función a ejecutar ;).

En el siguiente ejemplo, se añaden eventos a los elementos de tipo input=text de un formulario complejo:

```
function resalta() {  
    // Código JavaScript  
}
```

```
window.onload = function() {  
    var formulario = document.getElementById("formulario");  
    var camposInput = formulario.getElementsByTagName("input");
```

```
    for(var i=0; i<camposInput.length; i++) {  
        if(camposInput[i].type == "text") {
```

```
camposInput[i].onclick = resalta;
    }
}
}
```

En el siguiente ejemplo vamos a tener una lista de textos. Lo que vamos a hacer es asociarles un evento onClick. De tal manera, que cuando se produzca dicho evento, mostraran el contenido del texto.

Lo primero es crear la lista de elementos:

```
<ul id="frases">
<li>Avila</li>
<li>Salamanca</li>
<li>Zamora</li>
<li>León</li>
<li>Soria</li>
<li>Valladolid</li>
<li>Burgos</li>
<li>Segovia</li>
<li>Palencia</li>
</ul>
```

Ahora, ejecutaremos el código JavaScript para añadir un evento a cada uno de los elementos.

En dicho código diferenciaremos dos partes. La primera lo que hace es recuperar una referencia a la lista y recorrer todos los elementos que la contienen. Por cada elemento llamaremos al método **crearEvento**, el cual veremos en detalle más adelante.

Para poder acceder a la lista nos apoyaremos en los métodos `.getElementById`, que nos posiciona en un elemento y en `.getElementsByTagName`, que nos devuelve una lista de elementos dada una etiqueta. En nuestro caso la etiqueta `li`.

El código JavaScript será el siguiente:

```
// Obtenemos los elementos DIV a los que queremos añadir nuestro evento
onclick
var elementos =
document.getElementById("frases").getElementsByTagName("li");
// Recorremos todos los elementos
for (var i=0; i<elementos.length; i++) {
    // Añadimos el evento onclick al div
    crearEvento(elementos[i], "click", function(){
        // Hacemos que muestre el contenido del DIV
        alert(this.innerHTML);
    });
}
```

Como vemos por cada elemento llamamos a la función **crearEvento**, pasándole el elemento, el evento a crear y la función a asociarle.

Para crear un evento sobre un elemento tenemos el método `.addEventListener`. Esta es la función utilizada por el DOM. Pero para el caso del Internet Explorer deberemos de utilizar el método `.attachEvent`. Ambos métodos los ejecutaremos sobre el elemento.

Hay que indicar que para `.addEventListener` el nombre del evento será sin anteponerle el "on", mientras que para `.attachEvent` si que le antepondremos el "on".

De esta forma, la función JavaScript para crear un evento sobre un método quedará de la siguiente forma:

```
function crearEvento(elemento, evento, funcion) {
    if (elemento.addEventListener) {
        elemento.addEventListener(evento, funcion, false);
    } else {
        elemento.attachEvent("on" + evento, funcion);
    }
}
```

El código completo se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
<script>
function crearEvento(elemento, evento, función) {
if (elemento.addEventListener) {
    elemento.addEventListener(evento, función, false);
}else{
    elemento.attachEvent("on" + evento, función);
}
}
</script>
</head>
<body>
    <ul id="frases">
        <li>Avila</li>
        <li>Salamanca</li>
        <li>Zamora</li>
        <li>León</li>
        <li>Soria</li>
        <li>Valladolid</li>
        <li>Burgos</li>
        <li>Segovia</li>
        <li>Palencia</li>
    </ul>

    <script>
        // Obtenemos los elementos DIV a los que queremos añadir nuestro evento onclick
        var elementos =
        document.getElementById("frases").getElementsByTagName("li");
```

```
// Recorremos todos los elementos
for (var i=0; i<elementos.length; i++) {
  // Añadimos el evento onclick al div
  crearEvento(elementos[i], "click", function(){
    // Hacemos que muestre el contenido del DIV
    alert(this.innerHTML);
  });
}
</script>
</body>
</html>
```

### 3.1.3 Implementando el paradigma de la POO en páginas web

El manejo de espacios de nombres en JavaScript ha sido ignorado durante muchos años porque los programas eran relativamente sencillos y, normalmente, una página web no se componía de diferentes piezas de código desarrolladas por programadores o empresas diferentes. El código se incluía como un script en la página y la mayoría de las variables y funciones eran globales.

Actualmente, en JavaScript se programan aplicaciones muy complejas (enormes a veces) y se han desarrollado cientos de librerías que podemos importar en nuestro código. Esto hace necesario organizar de alguna manera nuestro código para asegurarnos de que nuestras variables, objetos y funciones sean únicas y no se estén sobrescribiendo (o siendo sobrescritas) por una variable del mismo nombre de alguna de las librerías externas que utilicemos.

Nombres como getElement, counter, checkValue, showMessage, etc. son muy comunes y podrían existir en casi cualquier script que importemos.

Para poder utilizar cualquier nombre con seguridad, cada programa debe definir sus variables dentro de un contenedor único. Este contenedor es el espacio de nombres (namespace).

Cada librería externa que utilicemos tendrá su propio espacio de nombres y nosotros debemos definir también el nuestro para nuestro proyecto.

#### Namespace

Un espacio de nombres es un contenedor que permite asociar toda la funcionalidad de un determinado objeto con un nombre único. En JavaScript un espacio de nombres es un objeto que permite a métodos, propiedades y objetos asociarse. La idea de crear espacios de nombres en JavaScript es simple: Crear un único objeto global para las variables, métodos, funciones convirtiéndolos en propiedades de ese objeto. El uso de los namespace permite minimizar el conflicto de nombres con otros objetos haciéndolos únicos dentro de nuestra aplicación.

Un espacio de nombres es un objeto:

Vamos a crear un objeto global llamado MIAPLICACION

```
// namespace global
```

```
var MIAPLICACION = MIAPLICACION || {};
```

Nota: Para continuar con las mejores prácticas vamos a utilizar mayúsculas para los namespace.

En el código de ejemplo anterior comprobamos si MIAPLICACION ya se encuentra definida. Si es así utilizamos el objeto global MIAPLICACION que existe, si este no existe creamos un objeto vacío llamado MIAPLICACION que encapsulará métodos, funciones, variables y otros objetos que vayamos a crear.

También podemos crear Sub-espacios de nombres:

```
// Sub-namespace
```

```
MIAPLICACION.event = {};
```

A continuación se muestra el código para la creación de un espacio de nombre y como agregar variables, funciones y métodos:

```
// Creación del contenedor llamado MIAPLICACION.metodoComun de método y propiedades comunes.
```

```
MIAPLICACION.metodoComun = {  
  regexParaNombre: "", // define regex para la validación del nombre  
  regexParaTelefono: "", // define regex para validación del teléfono  
  validaNombre: function(nombre){  
    // Hace algo con el nombre que usted ingresa a la variable reExParaNombre  
    // usando "this.regexParaNombre"  
  },
```

```
  validaNroTelefono: function (numTelefono){  
    // Hace algo con el número de teléfono  
  }  
}
```

```
// Objeto junto a la declaración del método
```

```
MIAPLICACION.event = {  
  addListener: function(el, type, fn){  
    // código de relleno  
  },  
  removeListener: function(el, type, fn){  
    // código de relleno  
  },  
  getEvent: function(e) {  
    // código de relleno  
  }  
}
```

```
// Puedes agregar otras propiedades y métodos  
}
```

```
// Sintaxis de utilización del método addListener:
```

```
MIAPLICACION.event.addListener("turl", "tipo", callback);
```

Veremos otra forma muy utilizada en Javascript para definir objetos. Esta forma se la llama **Objetos literales**

Esta metodología consiste en definir una lista de propiedades y sus valores. Veamos con un ejemplo esta técnica:

```
<html>
<head>
</head>
<body>

<script type="text/javascript">

  var cliente1= {
nombre: 'Juan',
  deposito: 0,
  imprimir: function ()
  {
    document.write(this.nombre+'<br>');
    document.write(this.deposito+'<br>');
  },
  depositar: function(monto) {
    this.deposito=this.deposito+monto;
  },
  extraer: function(monto) {
    this.deposito=this.deposito-monto;
  }
  };

  cliente1.imprimir();
  cliente1.depositar(1000);
  document.write('Estado luego de depositar 1000 pesos</br>');
  cliente1.imprimir();
  cliente1.extraer(200);
  document.write('Estado luego de extraer 200 pesos</br>');
  cliente1.imprimir();

</script>

</body>
</html>
```

En este ejemplo hemos creado un objeto literal llamado cliente1, la sintaxis mínima para crear un objeto vacío sería:

```
var cliente1= {};
```

Es decir creamos una variable llamada cliente1 y le asignamos un bloque encerrado entre llaves vacío. Es importante notar el punto y coma al final de la llave de cerrado (como ocurre cuando asignamos un valor a una variable)

Veamos ahora si decimos que el objeto cliente1 define la propiedad nombre, luego nuestro objeto quedará definido con la sintaxis:

```
var cliente1= {
  nombre: 'Juan'
};
```

Decimos que la propiedad nombre almacena el string 'Juan', del lado izquierdo indicamos el nombre de la propiedad y del lado derecho de los dos puntos indicamos

el valor de la propiedad del objeto (el valor puede ser de cualquier tipo, en este caso es de tipo string pero podría ser de tipo number, boolean, object, Array etc.)

Ahora si agregamos una segunda propiedad a nuestro objeto cliente1 llamada deposito (que representa la cantidad de dinero que tiene depositado el cliente1) la sintaxis queda:

```
var cliente1= {  
    nombre: 'Juan',  
    deposito: 0  
};
```

Como podemos observar separamos por coma cada inicialización de propiedad del objeto (menos para la última propiedad donde aparece la "}").

Las funciones del objeto también definimos una sintaxis similar a la declaración de sus propiedades:

```
var cliente1= {  
    nombre: 'Juan',  
    deposito: 0,  
    imprimir: function ()  
    {  
        document.write(this.nombre+'<br>');  
        document.write(this.deposito+'<br>');  
    },  
    depositar: function(monto) {  
        this.deposito=this.deposito+monto;  
    },  
    extraer: function(monto) {  
        this.deposito=this.deposito-monto;  
    }  
};
```

Del lado izquierdo de los dos puntos indicamos el nombre de la función y del lado derecho utilizamos la palabra clave function junto con los parámetros.

En la función podemos acceder a las propiedades del objeto antecediendo la palabra clave this.

Ahora solo nos falta hacer la llamada a las funciones del objeto cliente1:

```
cliente1.imprimir();  
cliente1.depositar(1000);  
document.write('Estado luego de depositar 1000 pesos</br>');  
cliente1.imprimir();  
cliente1.extraer(200);  
document.write('Estado luego de extraer 200 pesos</br>');  
cliente1.imprimir();
```

La propiedad prototype es accesible para todos los objetos y nos da una referencia a la clase con la que fue creada. Con el prototipo podemos modificar las propiedades y



métodos de una clase, de tal forma que todos los objetos creados anterior y posteriormente incorporarán esas modificaciones.

Veamos esto en acción. Primero, la clase coche:

```
function coche(unaMarca, unModelo, unColor) {
  this.marca = unaMarca;
  this.modelo = unModelo;
  this.color = unColor;

  this.seleccionarMarca = function seleccionarMarca(otraMarca){
    this.marca = otraMarca
  };

  this.seleccionarModelo = function seleccionarModelo(otroModelo){
    this.modelo = otroModelo
  };

  this.seleccionarColor = function seleccionarColor(otroColor){
    this.color = otroColor
  };

  this.verComoEs = function verComoEs(){
    return "Marca: " + this.marca + "<br />" +
    "Modelo: " + this.modelo + " <br />" +
    "Color: " + this.color;
  };
}
```

Con este ejemplo de la clase coche creamos una instancia de objeto que llamamos primerCoche.

```
<b><i>"Así es..."</i></b>
<div id="div1" class="verde"></div>
<script>
  var primerCoche = new coche("Volkswagen", "Golf", "Rojo");
  document.getElementById("div1").innerHTML = primerCoche.verComoEs();
</script>
```

Ejemplo:

***"Así es el primerCoche:"***

Marca: Volkswagen  
Modelo: Golf  
Color: Rojo

Luego accedemos al prototipo de la clase con coche.prototype para modificar cosas. Ponemos una nueva propiedad matricula (la incluimos en la clase, no en el objeto

como hacíamos con la función **modificadora**), y dos nuevos métodos `ponerMatricula()` y `verMatricula()`.

```
<b><i>"Modificamos..."</i></b>
<div id="div2" class="verde"></div>
<script>
    coche.prototype.matricula = "";
    coche.prototype.verMatricula = function verMatricula() {
        return this.matricula;
    }
    coche.prototype.ponerMatricula = function ponerMatricula(unaMatricula) {
        this.matricula = unaMatricula;
    }
    primerCoche.ponerMatricula("ABC1234");
    document.getElementById("div2").innerHTML = primerCoche.verComoEs() + "<br
/>" +
    "Matrícula: " + primerCoche.verMatricula();
</script>
```

Ejemplo:

***"Modificamos la clase coche para añadir propiedades y métodos:"***

Marca: Volkswagen  
Modelo: Golf  
Color: Rojo  
Matrícula: ABC1234

De ahora en adelante todos los objetos instanciados de la clase `coche` ya tienen esa nueva propiedad y métodos. Por ejemplo, podemos crear un nuevo objeto `segundoCoche`:

```
<b><i>"Este es..."</i></b>
<div id="div3" class="verde"></div>
<script>
    var miSegundoCoche = new coche("Fiat", "Punto", "Azul");
    miSegundoCoche.ponerMatricula("MAT0001");
    document.getElementById("div3").innerHTML = miSegundoCoche.verComoEs() +
    "<br />" +
    "Matrícula: " + miSegundoCoche.verMatricula();
</script>
```

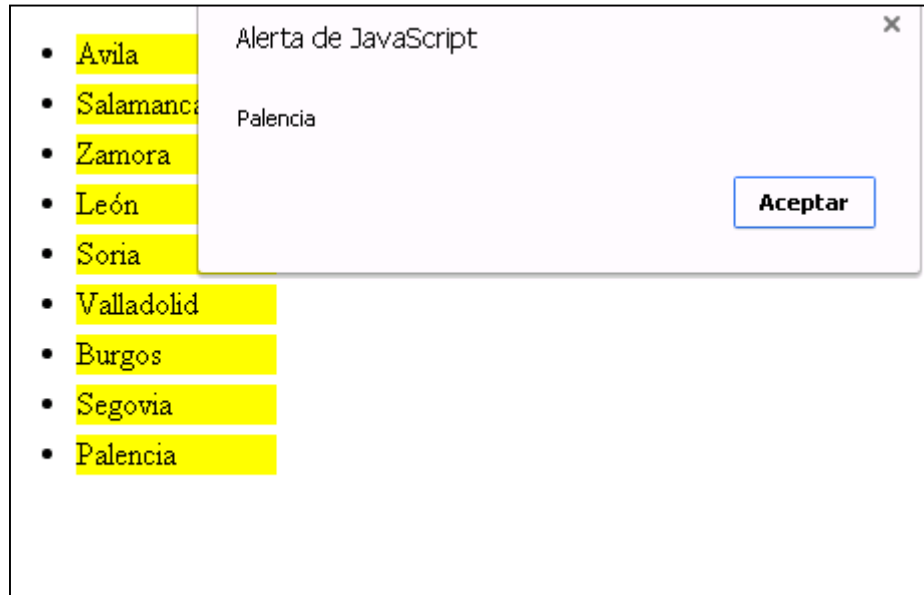
Ejemplo:

***"Este es un segundoCoche de la clase coche modificada:"***

Marca: Fiat  
Modelo: Punto  
Color: Azul  
Matrícula: MAT0001

## Laboratorio 7.1

En el siguiente ejemplo vamos a aplicar el uso de eventos en javascript. Lo que vamos a hacer es asociarles un evento onClick a una lista de textos. De tal manera, que cuando se produzca dicho evento, mostrarán el contenido del texto.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema7a y lo guarda en la carpeta Tema7a.
2. Cree una archivo js e inserte el siguiente código:

```
//esta función elige el navegador adecuado para ejecutar el evento
function crearEvento(elemento, evento, funcion) {
    if (elemento.addEventListener) {
        elemento.addEventListener(evento, funcion, false);
    } else {
        elemento.attachEvent("on" + evento, funcion,true);
    }
}
```

```
// Obtenemos los elementos LI para añadir evento onclick
var uls=document.getElementById("frases");
var ele = uls.getElementsByTagName("li");
```

```
// Recorremos todos los elementos
for (var i=0; i<ele.length; i++) {
// Añadimos el evento onclick al li
  crearEvento(ele[i], "click", function(){
    // Hacemos que muestre el contenido del LI
    alert(this.innerHTML);
  });
}
```

3. Grabe el archivo en la carpeta js con el nombre códigos.js.
4. Cree un archivo en la carpeta css con el nombre estilos.css e inserte el siguiente código:

```
li{
width:100px;
height:20px;
margin-top:5px;
background-color:yellow;
cursor:pointer;
}
```

5. Cree un archivo html e ingrese el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
  <link href="css/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <ul id="frases">
<li>Avila</li>
<li>Salamanca</li>
<li>Zamora</li>
<li>León</li>
<li>Soria</li>
<li>Valladolid</li>
<li>Burgos</li>
<li>Segovia</li>
<li>Palencia</li>
</ul>
<script src="js/codigos.js" type="text/javascript"></script>
</body>
</html>
```

6. Guarde el archivo con el nombre regex.html y luego ejecute su pagina.

# Resumen

1. La programación orientada a objetos puede considerarse como el diseño de software a través de un conjunto de objetos que cooperan.
2. JavaScript es un lenguaje basado en prototipos que no contiene ninguna declaración de clase.
3. JavaScript utiliza funciones como clases. Una clase es la definición de un objeto. Esta clase tiene propiedades, funciones y eventos
4. Un objeto es la instancia (copia o ejemplar) de una clase. Para definir un objeto sólo tendríamos que llamar a su constructor
5. Un evento es un proceso que se realiza en respuesta a determinada acción realizada por el usuario
6. Un espacio de nombres es un contenedor que permite asociar toda la funcionalidad de un determinado objeto con un nombre único.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://youtu.be/jVzwKPtm9Jo?list=PL962XzQ8DeRHRikZxVd8itD-Fcg51KykQ>
- <http://notasjs.blogspot.com/2012/06/espacios-de-nombres-en-javascript.html>
- <http://manuales.dgsca.unam.mx/javascript/Objetos.html>
- [http://librosweb.es/libro/javascript/capitulo\\_6/modelo\\_basico\\_de\\_eventos\\_2.html](http://librosweb.es/libro/javascript/capitulo_6/modelo_basico_de_eventos_2.html)
- <http://web.ontuts.com/tutoriales/namespacing-en-javascript/>
- <https://jherax.wordpress.com/2014/07/08/javascript-poo-1/>

## 3.2 CLASES PREDEFINIDAS

### 3.2.1 Array

En javascript un arreglo es una colección ordenada de elementos no homogéneos, cada elemento puede ser de un tipo de dato diferente. En javascript los arreglos empiezan con el subíndice 0.

Los arrays simples, también llamados "tablas" o "matrices", son el método más común y simple de almacenar datos. Técnicamente, un Array es un objeto intrínseco de JavaScript. Admite datos del tipo cadenas, números, objetos, arrays, true, false, null.

Un array en javascript se puede inicializar de las siguientes formas

```
a = []; // array vacío
a = [1, 'hola', true]; // array con elementos
a = new Array(); // array vacío
```

Se pueden asignar elementos al array independientemente que existan o no

```
a = [];
a[5] = 23;
// Esto da un array [undefined, undefined, undefined, undefined, undefined, 23]
a[10] // devuelve undefined
```

#### Usar el array como una pila

Se puede añadir un elemento al final del array

```
a = [1,2,3];
a.push('hola');
// esto da un array [1, 2, 3, "hola"]
```

y se pueden eliminar elementos del final

```
a = [1,2,3];
b = a.pop();
// ahora b tiene el elemento 3 y a vale [1,2]
```

#### Recorrer el array

Se puede recorrer un array con un bucle normal

```
a = [1,2,3];
for (i=0;i<a.length;i++) {
// hacer algo con a[i];
}
```

Con **every()**, se puede recorrer un array hasta encontrar algo en él

```
a = [1,2,3];
a.every ( function (elemento) {
if (elemento no cumple una condición) {
```

```
    return true; // seguir con el siguiente elemento
  } else {
    return false; // terminar el bucle en este elemento
  }
});
```

O se puede recorrer completo con **foreach()** de esta otra forma

```
a = [1,2,3];
a.forEach ( function (elemento) {
// hacer algo con elemento
});
```

### Otras funciones de los arrays

Todos los arrays de javascript heredan de la clase Array, así que todos nuestros arrays tendrán disponibles los métodos de la clase Array.

**concat()** concatena arrays y nos devuelve un array con todos los elementos

```
var a = [1,2];
var b = a.concat([3,4],[5,6]);
// b vale [1,2,3,4,5,6]
```

**indexOf()** nos devuelve la posición de un elemento en el array, -1 si no existe.

```
var a = ['hola','adios'];
a.indexOf('hola'); // devuelve 0
a.indexOf('adios'); // devuelve 1
a.indexOf('tres'); // devuelve -1
```

**lastIndexOf()** hace lo mismo, pero empezando a buscar por detrás

```
var a=["a","b","a"];
a.indexOf("a"); // Devuelve 0
a.lastIndexOf("a"); // Devuelve 2
```

**join()** devuelve un string con todos los elementos del array concatenados. Admite un parámetro para poner el separador que queremos entre los elementos, por defecto una coma

```
var a = ['hola','adios'];
a.join(); // Devuelve "hola,adios"
a.join("---"); // Devuelve "hola---adios"
```

**reverse()** le da la vuelta al array

```
var a=['hola','adios'];
a.reverse(); // Devuelve ["adios", "hola"] y a vale ahora eso.
```

**shift()** nos devuelve el primer elemento del array, eliminándolo del mismo

```
var a=['hola','adios'];  
a.shift(); // Devuelve "hola"  
a; // Vale ahora ["adios"]
```

**unshift()** añade elementos al principio del array. Hay que fijarse que los añade de forma que queden en el mismo orden que los hemos puesto en los parámetros.

```
var a = [0,1,2,3,4];  
a.unshift(22,23,24);  
a; // Ahora vale [22,23,24,0,1,2,3,4]
```

**slice()** nos devuelve un trozo del array, debemos indicar dos parámetros, el primer índice (que se incluye en el array resultante) y el último índice (que no se incluye).

```
var a=['a','e','i','o','u'];  
a.slice(1,3); // Devuelve ["e", "i"]
```

Si omitimos el segundo parámetro, se devuelve hasta el final del array

```
a.slice(3); // Devuelve ["o", "u"]
```

Si algún índice es negativo, se empieza a contar desde el último elemento del array

```
a.slice(-4,-1); // Devuelve ["e", "i", "o"]
```

**sort()** ordena el array. El orden siempre se hace convirtiendo a texto y ordenando alfabéticamente. En el caso de números, el resultado es extraño

```
var a = [22,2,33,3];  
a.sort();  
a; // vale ahora [2, 22, 3, 33] ("22" alfabéticamente va antes de "3")
```

Para ordenar número o si simplemente no queremos orden alfabético sino otro orden, debemos pasar una función que recibe dos parámetros a y b, que serán elementos del array. La función debe devolver un número negativo si consideramos que a es menor que b, positivo si a es mayor que b y 0 si da igual el orden de a y b. Para ordenar números, debemos pasar una función como esta

```
var a = [22,2,33,33];  
a.sort(function(a,b) {return a-b}); // Si a es mayor que b, a-b es positivo.  
a; // ahora vale [2, 3, 22, 33]
```

**splice()** permite añadir o quitar elementos del array en cualquier posición. El primer parámetro que se pasa es el índice del array donde se quieren insertar o borrar elementos. El segundo parámetro es cuántos elementos queremos borrar (0 si queremos insertar elementos) y el resto de parámetros que pongamos son elementos a añadir al array



```
var a = [0,1,'b',4,5];
a.splice(2,1,2,3); // Borramos la 'b' y la reemplazamos por los 2 y 3 que faltan en el
array
           // A partir del índice 2 (corresponde a la 'b'), borramos un elemento (la 'b')
           // y añadimos el 2 y el 3.
a; // vale ahora [0,1,2,3,4]
```

### Ejemplo 1

En el siguiente ejemplo se muestra el uso de algunos de sus métodos. Se utilizan los métodos `sort()` y `reverse()` para mostrar los elementos de un array en orden alfabético y en orden inverso.

```
<html>
<head><title> Ejemplo del Objeto Array</title></head>
<body>
<script language=JavaScript>
var myShopping = new Array("Huevos","Leche","Papas","Cereales","Plátanos");
var ord = prompt("Escriba 1 para ordenar alfabéticamente y para el orden inverso -1",
1);
if (ord == 1) {
    myShopping.sort();
    document.write(myShopping.join("<BR>"));
}
else if (ord == -1) {
    myShopping.sort();
    myShopping.reverse();
    document.write(myShopping.join("<BR>"));
}
else {
    document.write("No ha tecleado una entrada válida");
}
</script>
</body>
</html>
```

### 3.2.2 Date

Sobre la clase `Date` recae todo el trabajo con fechas en Javascript, como obtener una fecha, el día la hora actuales y otras cosas. Para trabajar con fechas necesitamos instanciar un objeto de la clase `Date` y con él ya podemos realizar las operaciones que necesitemos.

Un objeto de la clase `Date` se puede crear de dos maneras distintas. Por un lado podemos crear el objeto con el día y hora actuales y por otro podemos crearlo con un día y hora distintos a los actuales. Esto depende de los parámetros que pasemos al construir los objetos.

Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase `Date`.

```
miFecha = new Date()
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis el momento con que inicializar el objeto. Hay varias maneras de expresar un día y hora válidas, por eso podemos construir una fecha guiándonos por varios esquemas. Estos son dos de ellos, suficientes para crear todo tipo de fechas y horas.

```
miFecha = new Date(año,mes,día,hora,minutos,segundos)
miFecha = new Date(año,mes,día)
```

Los valores que debe recibir el constructor son siempre numéricos. Un detalle, el mes comienza por 0, es decir, enero es el mes 0. Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

Los objetos de la clase Date no tienen propiedades pero si un montón de métodos, vamos a verlos ahora.

`getDate()`  
Devuelve el día del mes.

`getDay()`  
Devuelve el día de la semana.

`getHours()`  
Retorna la hora.

`getMinutes()`  
Devuelve los minutos.

`getMonth()`  
Devuelve el mes (atención al mes que empieza por 0).

`getSeconds()`  
Devuelve los segundos.

`getTime()`  
Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje.

`getYear()`  
Retorna el año, al que se le ha restado 1900. Por ejemplo, para el 1995 retorna 95, para el 2005 retorna 105. Este método está obsoleto en Netscape a partir de la versión 1.3 de Javascript y ahora se utiliza `getFullYear()`.

`getFullYear()`  
Retorna el año con todos los dígitos. Usar este método para estar seguros de que funcionará todo bien en fechas posteriores al año 2000.

`setDate()`  
Actualiza el día del mes.

`setHours()`  
Actualiza la hora.

`setMinutes()`  
Cambia los minutos.

setMonth()

Cambia el mes (atención al mes que empieza por 0).

setSeconds()

Cambia los segundos.

setTime()

Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.

setYear()

Cambia el año recibe un número, al que le suma 1900 antes de colocarlo como año de la fecha. Por ejemplo, si recibe 95 colocará el año 1995. Este método está obsoleto a partir de Javascript 1.3 en Netscape. Ahora se utiliza setFullYear(), indicando el año con todos los dígitos.

setFullYear()

Cambia el año de la fecha al número que recibe por parámetro. El número se indica completo ej: 2005 o 1995. Utilizar este método para estar seguros que todo funciona para fechas posteriores a 2000.

### Ejemplo 1

El siguiente ejemplo escribe el día, mes y año actuales en el documento HTML que lo contiene. Para ello se utilizan los métodos: getFullYear(), getMonth(), getDate().

```
<html>
<head><title> Ejemplo del Objeto Date</title></head>
<body>
<script language=JavaScript>
var months = new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio",
                        "Agosto","Septiembre","Octubre","Noviembre","Diciembre");
var dateNow = new Date();
var yearNow = dateNow.getFullYear();
var monthNow = months[dateNow.getMonth()];
var dayNow = dateNow.getDate();
var daySuffix;
switch (dayNow){
case 1: case 21: case 31:
    daySuffix = "er";
    break;
case 2: case 22:
    daySuffix = "do";
    break;
case 3: case 23:
    daySuffix = "er";
    break;
default:
    daySuffix = "to";
break;
}
document.write("Es el " + dayNow + daySuffix + " día ");
document.write("del mes de " + monthNow);
document.write(" del año " + yearNow);
```

```
</script>
</body>
</html>
```

## Ejemplo 2

Usando los métodos `getHours()`, `getMinutes()` y `getSeconds()`, se escribe en la página web la hora actual.

```
<html>
<head><title> Ejemplo del Objeto Date</title></head>
<body>
<script language=JavaScript>
var greeting;
var nowDate = new Date();
var nowHour = nowDate.getHours();
var nowMinute = nowDate.getMinutes();
var nowSecond = nowDate.getSeconds();
if (nowMinute < 10)
    nowMinute = "0" + nowMinute;
if (nowSecond < 10)
    nowSecond = "0" + nowSecond;
if (nowHour < 12) {
    greeting = "Buenos Días";
}
else if (nowHour < 17) {
    greeting = "Buenas Tardes";
}
else {
    greeting = "Buenas Noches";
}
document.write("<H4>" + greeting + ", bienvenido a my página web </H4>")
document.write("De acuerdo con su reloj son las ");
document.write(nowHour + ":" + nowMinute + ":" + nowSecond);
</script>
</body>
</html>
```

### 3.2.3 Math

La clase `Math` es una de las clases nativas de Javascript. Proporciona los mecanismos para realizar operaciones matemáticas en Javascript. Algunas operaciones se resuelven rápidamente con los operadores aritméticos que ya conocemos, como la multiplicación o la suma, pero hay una serie de operaciones matemáticas adicionales que se tienen que realizar usando la clase `Math` como pueden ser calcular un seno o hacer una raíz cuadrada.

De modo que para cualquier cálculo matemático complejo utilizaremos la clase `Math`, con una particularidad. Hasta ahora cada vez que queríamos hacer algo con una clase debíamos instanciar un objeto de esa clase y trabajar con el objeto y en el caso de la clase `Math` se trabaja directamente con la clase. Esto se permite por que las

propiedades y métodos de la clase Math son lo que se llama propiedades y métodos de clase y para utilizarlos se opera a través de la clase en lugar de los objetos. Dicho de otra forma, para trabajar con la clase Math no deberemos utilizar la instrucción new y utilizaremos el nombre de la clase para acceder a sus propiedades y métodos.

### Propiedades de Math

Las propiedades guardan valores que probablemente necesitemos en algún momento si estamos haciendo cálculos matemáticos. Es probable que estas propiedades resulten un poco raras a las personas que desconocen las matemáticas avanzadas, pero los que las conozcan sabrán de su utilidad.

E

Número E o constante de Euler, la base de los logaritmos neperianos.

LN2

Logaritmo neperiano de 2.

LN10

Logaritmo neperiano de 10.

LOG2E

Logaritmo en base 2 de E.

LOG10E

Logaritmo en base 10 de E.

PI

Conocido número para cálculo con círculos.

SQRT1\_2

Raíz cuadrada de un medio.

SQRT2

Raíz cuadrada de 2.

### Métodos de Math

Así mismo, tenemos una serie de métodos para realizar operaciones matemáticas típicas, aunque un poco complejas. Todos los que conozcan las matemáticas a un buen nivel conocerán el significado de estas operaciones.

abs()

Devuelve el valor absoluto de un número. El valor después de quitarle el signo.

acos()

Devuelve el arcocoseno de un número en radianes.

asin()

Devuelve el arcoseno de un numero en radianes.

atan()

Devuelve un arcotangente de un numero.

ceil()

Devuelve el entero igual o inmediatamente siguiente de un número. Por ejemplo, ceil(3) vale 3, ceil(3.4) es 4.

cos()

Retorna el coseno de un número.

exp()

Retorna el resultado de elevar el número E por un número.

floor()

Lo contrario de ceil(), pues devuelve un número igual o inmediatamente inferior.

log()

Devuelve el logaritmo neperiano de un número.

max()

Retorna el mayor de 2 números.

min()

Retorna el menor de 2 números.

pow()

Recibe dos números como parámetros y devuelve el primer número elevado al segundo número.

random()

Devuelve un número aleatorio entre 0 y 1. Método creado a partir de Javascript 1.1.

round()

Redondea al entero más próximo.

sin()

Devuelve el seno de un número con un ángulo en radianes.

sqrt()

Retorna la raíz cuadrada de un número.

tan()

Calcula y devuelve la tangente de un número en radianes

## Ejemplo 1

El código que se muestra a continuación le solicita al usuario que introduzca un número y muestra el resultado de aplicarle los métodos parseInt(), ceil(), floor() y round().

```
<html>
<head><title> Ejemplo del Objeto Math</title></head>
<body>
<script language=JavaScript>
var myNumber = prompt("Introduzca el número que quiere redondear","166.386");
document.write("<H3>El número que escribió fue " + myNumber + "</H3><BR>");
document.write("<P>Los redondeos para este número son</P>");
document.write("<TABLE WIDTH=150 BORDER=1>");
document.write("<TR><TH>Método</TH><TH>Resultado</TH></TR>");
```

```
document.write("<TR><TD>parseInt()</TD><TD>" + parseInt(myNumber)
+ "</TD></TR>");
document.write("<TR><TD>ceil()</TD><TD>" + Math.ceil(myNumber) +
"</TD></TR>");
document.write("<TR><TD>floor()</TD><TD>" + Math.floor(myNumber) +
"</TD></TR>");
document.write("<TR><TD>round()</TD><TD>" + Math.round(myNumber)
+ "</TD></TR>");
document.write("</TABLE>")
</script>
</body>
</html>
```

## Ejemplo 2

Mediante el uso de los métodos `pow()`, `round()` del objeto `Math` se escribe un función que fija el número de lugares decimales en un número.

```
<html>
<head><title> Ejemplo del Objeto Math</title></head>
<script language=JavaScript>
function fix(fixNumber, decimalPlaces)
{
    var div = Math.pow(10,decimalPlaces);
    fixNumber = Math.round(fixNumber * div) / div;
    return fixNumber;
}
</script>
</head>
<body>
<script language=JavaScript>
    var number1 = parseFloat(prompt("Inserte un número en coma flotante","166.386"));
    var number2 = parseFloat(prompt("¿Con cuántos decimales lo quiere?",""));
    document.write(number1 + " con " + number2 + " decimales es: ");
    document.write(fix(number1,number2));
</script>
</body>
</html>
```

### 3.2.4 String

En javascript las variables de tipo texto son objetos de la clase `String`. Esto quiere decir que cada una de las variables que creamos de tipo texto tienen una serie de propiedades y métodos. Recordamos que las propiedades son las características, como por ejemplo longitud en caracteres del string y los métodos son funcionalidades, como pueden ser extraer un substring o poner el texto en mayúsculas.

Para crear un objeto de la clase `String` lo único que hay que hacer es asignar un texto a una variable. El texto va entre comillas, como ya hemos visto en los capítulos de sintaxis. También se puede crear un objeto string con el operador `new`, que veremos más adelante. La única diferencia es que en versiones de Javascript 1.0 no funcionará `new` para crear los Strings.

## Propiedades de String

### Length

La clase String sólo tiene una propiedad: length, que guarda el número de caracteres del String.

## Métodos de String

Los objetos de la clase String tienen una buena cantidad de métodos para realizar muchas cosas interesantes. Primero vamos a ver una lista de los métodos más interesantes y luego vamos a ver otra lista de métodos menos útiles.

`charAt(indice)`

Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.

`indexOf(carácter,desde)`

Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.

`lastIndexOf(carácter,desde)`

Busca la posición de un carácter exactamente igual a como lo hace la función `indexOf` pero desde el final en lugar del principio. El segundo parámetro indica el número de caracteres desde donde se busca, igual que en `indexOf`.

`replace(substring_a_buscar,nuevoStr)`

Implementado en Javascript 1.2, sirve para reemplazar porciones del texto de un string por otro texto, por ejemplo, podríamos utilizarlo para reemplazar todas las apariciones del substring "xxx" por "yyy". El método no reemplaza en el string, sino que devuelve un resultante de hacer ese reemplazo. Acepta expresiones regulares como substring a buscar.

`split(separador)`

Este método sólo es compatible con javascript 1.1 en adelante. Sirve para crear un vector a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.

`substring(inicio,fin)`

Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.

`toLowerCase()`

Pone todas los caracteres de un string en minúsculas.



toUpperCase()

Pone todos los caracteres de un string en mayúsculas.

toString()

Este método lo tienen todos los objetos y se usa para convertirlos en cadenas.

### Ejemplo 1

En el ejemplo que se muestra a continuación se detecta si el primer caracter de una cadena dada es una letra mayúscula, una letra minúscula, o un número para ello se utiliza el método: `charCodeAt()`.

```
<html>
<head><title> Ejemplo del Objeto String</title></head>
<body>
<script language=JavaScript>
function checkCharType(charToCheck){
    var returnValue = "O";
    var charCode = charToCheck.charCodeAt(0);

    if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0)){
        returnValue = "U"; // Upper Case -- Letra Mayúscula
    }
    else if (charCode >= "a".charCodeAt(0) && charCode <= "z".charCodeAt(0)){
        returnValue = "L"; // Lower Case -- Letra Minúscula
    }
    else if (charCode >= "0".charCodeAt(0) && charCode <= "9".charCodeAt(0)){
        returnValue = "N"; // Es un número
    }
    return returnValue;
}
var myString = prompt("Inserte un texto","Aquí debe escribir algo");
switch (checkCharType(myString)){
    case "U":
        document.write("El primer carácter es una letra mayúscula");
        break;
    case "L":
        document.write("El primer carácter es una letra minúscula");
        break;
    case "N":
        document.write("El primer carácter es un número");
        break;
    default:
        document.write("El primer carácter no es ni un letra ni un número");
}
</script>
</body>
</html>
```

## Ejemplo 2

En el siguiente ejemplo se detecta el número de veces que aparece la subcadena 'toto' en una cadena usando el método `indexOf()`.

```
<html>
<head><title> Ejemplo del Objeto String</title></head>
<body>
<script language=JavaScript>
var myString = "Esto es un ejemplo importante por eso se llama Toto. "
myString = myString + "La página web del programa Toto es www.toto.org. "
myString = myString + "Visite la página web de Toto, no lo deje para mañana. "
myString = myString + "Gracias por usar Toto."

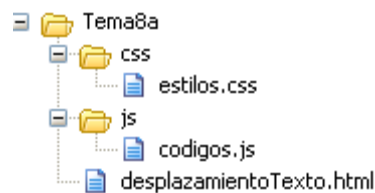
var foundAtPosition = 0;
var totoCount = 0;
while ( foundAtPosition != -1) {
    foundAtPosition = myString.indexOf("Toto",foundAtPosition);
    if (foundAtPosition != -1) {
        totoCount++;
        foundAtPosition++;
    }
}
document.write("La palabra Toto ha aparecido " + totoCount + " veces en el texto.");
</script>
</body>
</html>
```

## Laboratorio 8.1

En el siguiente laboratorio, se muestra un desplazamiento de texto en una capa div de una pagina web:

Bienvenidos a mi pagina web desarrollado con Javascript

En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Antes de empezar, debe crear un sitio web con el nombre Tema8a y lo guarda en la carpeta Tema8a.
2. Cree una archivo js e inserte el siguiente código:

```
//texto del mensaje
var texto = " Bienvenidos a mi pagina web desarrollado con
Javascript";
var pos = 0
//funcion para mover el texto de la barra de estado
function mueve_texto(){
  if (pos < texto.length)
    pos ++;
  else
    pos = 1;
  string_actual = texto.substring(pos) + texto.substring(0,pos)
  document.getElementById("texto").innerText = string_actual;
  setTimeout("mueve_texto()",150)
}
mueve_texto()
```

3. Grabe el archivo en la carpeta js con el nombre códigos.js.
4. Cree una archivo css en la carpeta css y lo graba como estilos.css. Inserte el siguiente código:

```
#texto{  
width:350px;  
height:40px;  
background-color:#ff0;  
}
```

5. Cree un archivo html e ingrese el siguiente código:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<title></title>  
<link href="css/estilos.css" rel="stylesheet" type="text/css" />  
</head>  
<body>  
<div id="texto"></div>  
<script src="js/codigos.js" language="javascript"></script>  
</body>  
</html>
```

6. Guarde el archivo con el nombre desplazamientoTexto.html y luego ejecute su pagina.

# Resumen

1. Los arrays sirven para guardar varias variables y acceder a ellas de manera independiente, es como tener una variable con distintos compartimentos donde podemos introducir datos distintos. Para crear una array se usan las siguientes sintaxis:

```
a = [];           // array vacío  
a = [1, 'hola', true]; // array con elementos  
a = new Array();  // array vacío
```

2. La clase Date permite representar y manipular valores relacionados con fechas y horas. Para obtener la representación de la fecha actual, sólo es necesario instanciar la clase sin parámetros:

```
var fecha = new Date();
```

3. La fecha y la hora se almacena como el número de milisegundos que han transcurrido desde el 1 de Enero de 1970 a las 00:00:00. Por este motivo, se puede construir una fecha cualquiera indicando el número de milisegundos a partir de esa referencia temporal.
4. La clase Math proporciona los mecanismos para realizar operaciones matemáticas en Javascript. Para trabajar con la clase Math no deberemos utilizar la instrucción new y utilizaremos el nombre de la clase para acceder a sus propiedades y métodos
5. En javascript las variables de tipo texto son objetos de la clase String. Esto quiere decir que cada una de las variables que creamos de tipo texto tienen una serie de propiedades y métodos. Para crear un objeto de la clase String lo único que hay que hacer es asignar un texto a una variable. otambién con el operador new.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.desarrolloweb.com/articulos/745.php>
- <http://www.desarrolloweb.com/articulos/762.php>
- <http://www.desarrolloweb.com/articulos/726.php>
- <http://www.desarrolloweb.com/articulos/630.php>
-



# JQUERY

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript, usando la librería JQuery, diseña programas, incorporados en una página del Sitio Web

## TEMARIO

### 4.1 Tema 9 : Introducción

- 4.1.1 : Definición
- 4.1.2 : Breve referencia CSS
- 4.1.3 : Software necesario
- 4.1.4 : Selectores

### 4.2 Tema 10 : Características principales

- 4.2.1 : Constructor
- 4.2.2 : Iteración implícita
- 4.2.3 : Consultas a través del DOM
- 4.2.4 : Encadenamiento (chaining)

### 4.3 Tema 11 : Manejo de eventos

- 4.3.1 : Introducción a los eventos
- 4.3.2 : Evento click
- 4.3.3 : Comportamiento en cola
- 4.3.4 : Evento hover
- 4.3.5 : Eventos del teclado
- 4.3.6 : Borrar eventos

### 4.4 Tema 12 : Efectos y modificaciones sobre el DOM

- 4.4.1 : Introducción a los efectos
- 4.4.2 : Efectos incorporados
- 4.4.3 : Efectos personalizados
- 4.4.4 : Control de efectos

## ACTIVIDADES PROPUESTAS

- Los alumnos aplican estilos con códigos JQuery.
- Los alumnos aplican eventos con códigos JQuery.
- Los alumnos aplican efectos con códigos JQuery.

## 3.1 INTRODUCCIÓN

### 3.1.1 Definición

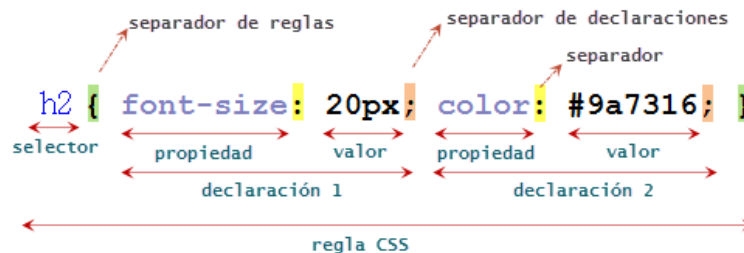
jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privados.<sup>2</sup> jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

jQuery es un conjunto de librerías JavaScript que han sido diseñadas específicamente para simplificar el desplazamiento de un documento HTML, la animación, la gestión de eventos y las interacciones Ajax.

### 3.1.2 Breve referencia CSS

Con CSS podemos hacer declaraciones de estilo sobre los elementos HTML, para esto el W3C: World Wide Web Consortium(<http://www.w3.org/Style/CSS/>) ha definido una serie de selectores los cuales siguen el siguiente formato:



jQuery hace uso de estos selectores para interactuar con el DOM, por esta razón es importante que el alumno comprenda CSS.

La futura versión CSS 3 incluye todos los selectores de CSS 2.1 y añade otras decenas de selectores, pseudo-classes y pseudo-elementos. La lista provisional de novedades y su explicación detallada se puede encontrar en el módulo de selectores de CSS 3.

En primer lugar, CSS 3 añade tres nuevos selectores de atributos:

**elemento[atributo^="valor"]**, selecciona todos los elementos que disponen de ese atributo y cuyo valor comienza exactamente por la cadena de texto indicada.

**elemento[atributo\$="valor"]**, selecciona todos los elementos que disponen de ese atributo y cuyo valor termina exactamente por la cadena de texto indicada.

**elemento[atributo\*="valor"]**, selecciona todos los elementos que disponen de ese atributo y cuyo valor contiene la cadena de texto indicada.

De esta forma, se pueden crear reglas CSS tan avanzadas como las siguientes:

```
/* Selecciona todos los enlaces que apuntan a una dirección de correo electrónico */
```

```
a[href^="mailto:"] { ... }
```

```
/* Selecciona todos los enlaces que apuntan a una página HTML */
```

```
a[href$=".html"] { ... }
```

```
/* Selecciona todos los títulos h1 cuyo atributo title contenga la palabra "capítulo" */
```

```
h1[title*="capítulo"] { ... }
```

Otro de los nuevos selectores de CSS 3 es el "selector general de elementos hermanos", que generaliza el selector adyacente de CSS 2.1. Su sintaxis es `elemento1 ~ elemento2` y selecciona el `elemento2` que es hermano de `elemento1` y se encuentra detrás en el código HTML. En el selector adyacente la condición adicional era que los dos elementos debían estar uno detrás de otro en el código HTML, mientras que ahora la única condición es que uno esté detrás de otro.

Si se considera el siguiente ejemplo:

```
h1 + h2 { ... } /* selector adyacente */
```

```
h1 ~ h2 { ... } /* selector general de hermanos */
```

```
<h1>...</h1>  
<h2>...</h2>  
<p>...</p>  
<div>  
<h2>...</h2>  
</div>  
<h2>...</h2>
```

El primer selector (`h1 + h2`) sólo selecciona el primer elemento `<h2>` de la página, ya que es el único que cumple que es hermano de `<h1>` y se encuentra justo detrás en el código HTML. Por su parte, el segundo selector (`h1 ~ h2`) selecciona todos los elementos `<h2>` de la página salvo el segundo. Aunque el segundo `<h2>` se encuentra detrás de `<h1>` en el código HTML, no son elementos hermanos porque no tienen el mismo elemento padre.

Los pseudo-elementos de CSS 2.1 se mantienen en CSS 3, pero cambia su sintaxis y ahora se utilizan `::` en vez de `:` delante del nombre de cada pseudo-elemento:

`::first-line`, selecciona la primera línea del texto de un elemento.

`::first-letter`, selecciona la primera letra del texto de un elemento.

`::before`, selecciona la parte anterior al contenido de un elemento para insertar nuevo contenido generado.

`::after`, selecciona la parte posterior al contenido de un elemento para insertar nuevo contenido generado.

CSS 3 añade además un nuevo pseudo-elemento:

`::selection`, selecciona el texto que ha seleccionado un usuario con su ratón o teclado.



Las mayores novedades de CSS 3 se producen en las pseudo-clases, ya que se añaden 12 nuevas, entre las cuales se encuentran:

`elemento:nth-child(numero)`, selecciona el elemento indicado pero con la condición de que sea el hijo enésimo de su padre. Este selector es útil para seleccionar el segundo párrafo de un elemento, el quinto elemento de una lista, etc.

`elemento:nth-last-child(numero)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

`elemento:empty`, selecciona el elemento indicado pero con la condición de que no tenga ningún hijo. La condición implica que tampoco puede tener ningún contenido de texto.

`elemento:first-child` y `elemento:last-child`, seleccionan los elementos indicados pero con la condición de que sean respectivamente los primeros o últimos hijos de su elemento padre.

`elemento:nth-of-type(numero)`, selecciona el elemento indicado pero con la condición de que sea el enésimo elemento hermano de ese tipo.

`elemento:nth-last-of-type(numero)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

Algunas pseudo-clases como `:nth-child(numero)` permiten el uso de expresiones complejas para realizar selecciones avanzadas:

```
li:nth-child(2n+1) { ... } /* selecciona todos los elementos impares de una lista */
li:nth-child(2n) { ... } /* selecciona todos los elementos pares de una lista */
```

```
/* Las siguientes reglas alternan cuatro estilos diferentes para los párrafos */
p:nth-child(4n+1) { ... }
p:nth-child(4n+2) { ... }
p:nth-child(4n+3) { ... }
p:nth-child(4n+4) { ... }
```

Empleando la pseudo-clase `:nth-of-type(numero)` se pueden crear reglas CSS que alternen la posición de las imágenes en función de la posición de la imagen anterior:

```
img:nth-of-type(2n+1) { float: right; }
img:nth-of-type(2n) { float: left; }
```

Otro de los nuevos selectores que incluirá CSS 3 es `:not()`, que se puede utilizar para seleccionar todos los elementos que no cumplen con la condición de un selector:

```
:not(p) { ... } /* selecciona todos los elementos de la página que no sean párrafos */
:not(#especial) { ... } /* selecciona cualquier elemento cuyo atributo id no sea "especial" */
```

A continuación, algunos ejemplos:

```
$('div.foo').has('p'); // el elemento div.foo contiene elementos <p>
$('h1').not('.bar'); // el elemento h1 no posee la clase 'bar'
$('ul li').filter('.current'); // un ítem de una lista desordenada
// que posee la clase 'current'
```

```
$('#ul li').first();      // el primer item de una lista desordenada
$('#ul li').eq(5);        // el sexto item de una lista desordenada
```

Aunque todavía faltan muchos años hasta que la versión CSS 3 sustituya a la actual versión CSS 2.1, los navegadores que más se preocupan por los estándares (Opera, Safari y Firefox) incluyen soporte para varios o casi todos los selectores de CSS 3.

Existe una herramienta llamada CSS Selectors test que permite comprobar los selectores que soporta el navegador con el que se hace la prueba. Se encuentra en el siguiente enlace: <http://www.css3.info/selectors-test/>

### 3.1.3 Software necesario

Para desarrollar sitios de Internet con jQuery básicamente necesitamos 4 cosas:

1. Editor de texto
2. Navegador web.
3. La librería jQuery.
4. Servidor web (para ajax y json).

Para descargar la librería más actual de jquery, vaya a la siguiente dirección url y luego lo guarda en el directorio de su sitio web:

<http://code.jquery.com/jquery-2.1.4.js>

Una vez que tenemos la librería jQuery, lo siguiente es incluir la librería en nuestros documentos HTML, esto lo hacemos, agregando el script dentro de la cabecera del documento.

```
<head>
<!-- agregando librería jQuery -->
    <script type='text/javascript' src='js/jquery1.9.js'></script>
</head>
```

### 3.1.4 Selectores

Por un selector entendemos en jQuery lo mismo que en CSS: una forma de permitirnos elegir un elemento (o varios) entre todos los que tenemos en nuestro documento HTML. ¿Para qué? Para luego poder aplicar sobre los elementos seleccionados diversas funciones.

La sintaxis para seleccionar un elemento de la página web es la siguiente:

```
$('#selector')
```

Es decir, jQuery utiliza el poder de los selectores para acceder de una manera rápida y sencilla a un elemento o grupo de elementos del DOM (Document Object Model) y luego poder aplicar sobre los mismos cualquier tipo de instrucción, evento, animación, etc. Por ejemplo, para aplicar la clase “enlace” a todos los elementos “a” que se encuentren dentro de un elemento “p”. Haríamos:

```
$('#p a').addClass('enlace');
```

No importa qué tipo de selector usemos en jQuery: siempre comenzaremos con `$()`. Prácticamente todo lo que se pueda usar en CSS se puede también incluir entre esos paréntesis de esta forma `$('selectores')`. De tal manera que por ejemplo podríamos hacer:

```
$('p')           //selecciona todos los párrafos del documento
$('#nombre-id')  //selecciona el elemento con un id
$('.nombre-clase') //selecciona el elemento con un class
```

## Selectores CSS

jQuery soporta prácticamente todos los selectores de CSS, con la ventaja de que podemos utilizar selectores de CSS3 que funcionen con Internet Explorer 6 gracias a jQuery. Lo vemos mejor con estos ejemplos:

```
$('div')  Selecciona todos los DIV del documento
$('a')    Selecciona todos los '<a >' del documento
$('p a')  Selecciona todos los '<a>' descendientes de un '<p >'
$('p, a') Selecciona todos los '<p>' y todos los '<a >' del documento
$('li.nombreClase') Selecciona todos los '<li >' con clase 'nombreClase'
$('fieldset a') Selecciona todos los '<a >' dentro de '<fieldset >'
$('li>p') Selecciona todos los '<p >' hijos directos de '<li >'
$('h1+p') Selecciona todos los '<p >' inmediatamente precedidos por
           un '<h1>' hermano

$('div~p') Selecciona todos los 'div' precedidos por un elemento '<p >'
$('p:has(b)') Selecciona todos los '<p>' que contienen un elemento '<b >'
$('div.nombreClase') Selecciona todos los 'div' con la clase 'nombreClase'
$('.nombreClase') Selecciona todos los elementos con la clase 'nombreClase'
$('#nombreID') Selecciona el elemento con un id 'nombreID'
$('img[alt]') Selecciona todos los elementos '<img >' con atributo 'alt'
$('button[id*=boton]') Selecciona todos los botones con atributo 'id'
                       que contenga la palabra boton
$('a[href$=.pdf]') Selecciona todos los '<a >' con atributo 'href'
                   acabado en .pdf
$('a[title^=lr]') Selecciona todos los '<a >' cuyo atributo 'title'
                  comienza por 'lr'
```

## Selectores propios de jQuery

A la amplia variedad de selectores propios de CSS jQuery añade sus propios selectores. Como característica que les distingue decir que siempre comienzan por dos puntos (:). Vamos a distinguir entre ellos:

### Selectores Posicionales

Estos selectores están basados en las relaciones posicionales entre elementos (como veíamos antes en ejemplo de la estructura del DOM). Como antes, los vamos a ver a través de ejemplos:

`$('p:first')` *Selecciona el primer elemento '`<p>`' de la página*  
`$('img[src$=.png]:first')` *Selecciona el primer '`<img>`' de la página que tiene un atributo src acabado en .png*  
`$('p:last')` *Selecciona el último '`<p>`' de la página*  
`$('li:first-child')` *Selecciona todos los '`<li>`' que son primeros hijos*  
`$('li:last-child')` *Selecciona todos los '`<li>`' que son últimos hijos*  
`$('li:only-child')` *Selecciona todos los '`<li>`' que sean hijos únicos*  
`$('li:nth-child(3)')` *Selecciona todos los '`<li>`' que sean el tercer elemento de su lista*  
`$('tr:nth-child(odd)')` *Selecciona todos los '`<tr>`' que sean impares*  
`$('tr:nth-child(even)')` *Selecciona todos los '`<tr>`' que sean pares*  
`$('div:nth-child(3n)')` *Selecciona cada tercer elemento '`<div>`'*  
`$('div:nth-child(3n+1)')` *Selecciona el elemento tras cada tercer '`<div>`'*  
`$('p:odd')` *Selecciona los '`<p>`' impares*  
`$('p:even')` *Selecciona los '`<p>`' pares*  
`$('p:eq(1)')` *Selecciona el segundo '`<p>`' - Comienza a contar desde cero*  
`$('p:gt(1)')` *Selecciona todos los '`<p>`' excepto los dos primeros*  
`$('p:lt(1)')` *Selecciona los dos primeros '`<p>`' - Comienza a contar desde 0*

### Selectores de Formularios

Cuando trabajemos con formularios jQuery nos ofrece una serie de selectores propios que nos permiten seleccionar de manera sencilla el elemento preciso. Vamos a verlos con ejemplos:

\$(':text')  
 \$(':checkbox')  
 \$(':radio')  
 \$(':image')  
 \$(':submit')  
 \$(':reset')  
 \$(':password')  
 \$(':file')

*Selecciona todos los elementos '<input >' con un tipo de atributo igual al nombre del selector (excluyendo los dos puntos). Por ejemplo, ':text' selecciona <input type="text">*

\$('input') *Selecciona los elementos input, textarea, select y button.*

\$('button') *Selecciona los elementos button e input con atributo 'type' igual a 'button'*

\$('enabled') *Selecciona los elementos del formulario activados*

\$('disabled') *Selecciona los elementos del formulario desactivados*

\$('checked') *Selecciona los radio buttons y checkboxes que están pulsados*

\$('selected') *Elementos de una lista de opciones que estén seleccionados*

*Estos selectores se pueden combinar, por ejemplo:*

\$('radio:checked'), \$(':text:disabled'),  
 \$('button:hidden'), \$('select[name=nombre]:selected'),  
 \$('input[name=nombre]:radio:checked')

### Crear Nuevos Elementos

jQuery provee una forma fácil y elegante para crear nuevos elementos a través del mismo método `$()` que se utiliza para realizar selecciones.

```
$('<p>Un nuevo párrafo</p>');
```

```
$('<li class="new">nuevo item de la lista</li>');
```

Crear un nuevo elemento con atributos utilizando un objeto

```

$('<a/>', {
  html: 'Un <strong>nuevo</strong> enlace',
  'class': 'new',
  href: 'foo.html'
});

```

Note que en el objeto que se pasa como argumento, la propiedad `class` está entre comillas, mientras que la propiedad `href` y `html` no lo están. Por lo general, los nombres de propiedades no deben estar entre comillas, excepto en el caso que se utilice como nombre una palabra reservada (como es el caso de `class`).

Cuando se crea un elemento, éste no es añadido inmediatamente a la página, sino que se debe hacerlo en conjunto con un método.

Crear un nuevo elemento en la página

```
var $myNewElement = $('<p>Nuevo elemento</p>');
    $myNewElement.appendTo('#content');

    $myNewElement.insertAfter('ul:last'); // eliminará al elemento <p>
                                         // existente en #content

    $('ul').last().after($myNewElement.clone()); // clonar al elemento <p>
                                                // para tener las dos versiones
```

Estrictamente hablando, no es necesario guardar al elemento creado en una variable — es posible llamar al método para añadir el elemento directamente después de `$()`. Sin embargo, la mayoría de las veces se deseará hacer referencia al elemento añadido, por lo cual, si se guarda en una variable no es necesario seleccionarlo después.

Crear y añadir al mismo tiempo un elemento a la página

```
$('ul').append('<li>item de la lista</li>');
```

Nota

La sintaxis para añadir nuevos elementos a la página es muy fácil de utilizar, pero es tentador olvidar que hay un costo enorme de rendimiento al agregar elementos al DOM de forma repetida. Si está añadiendo muchos elementos al mismo contenedor, en lugar de añadir cada elemento uno por vez, lo mejor es concatenar todo el HTML en una única cadena de caracteres para luego anexarla al contenedor. Una posible solución es utilizar un vector que posea todos los elementos, luego reunirlos utilizando `join` y finalmente anexarla.

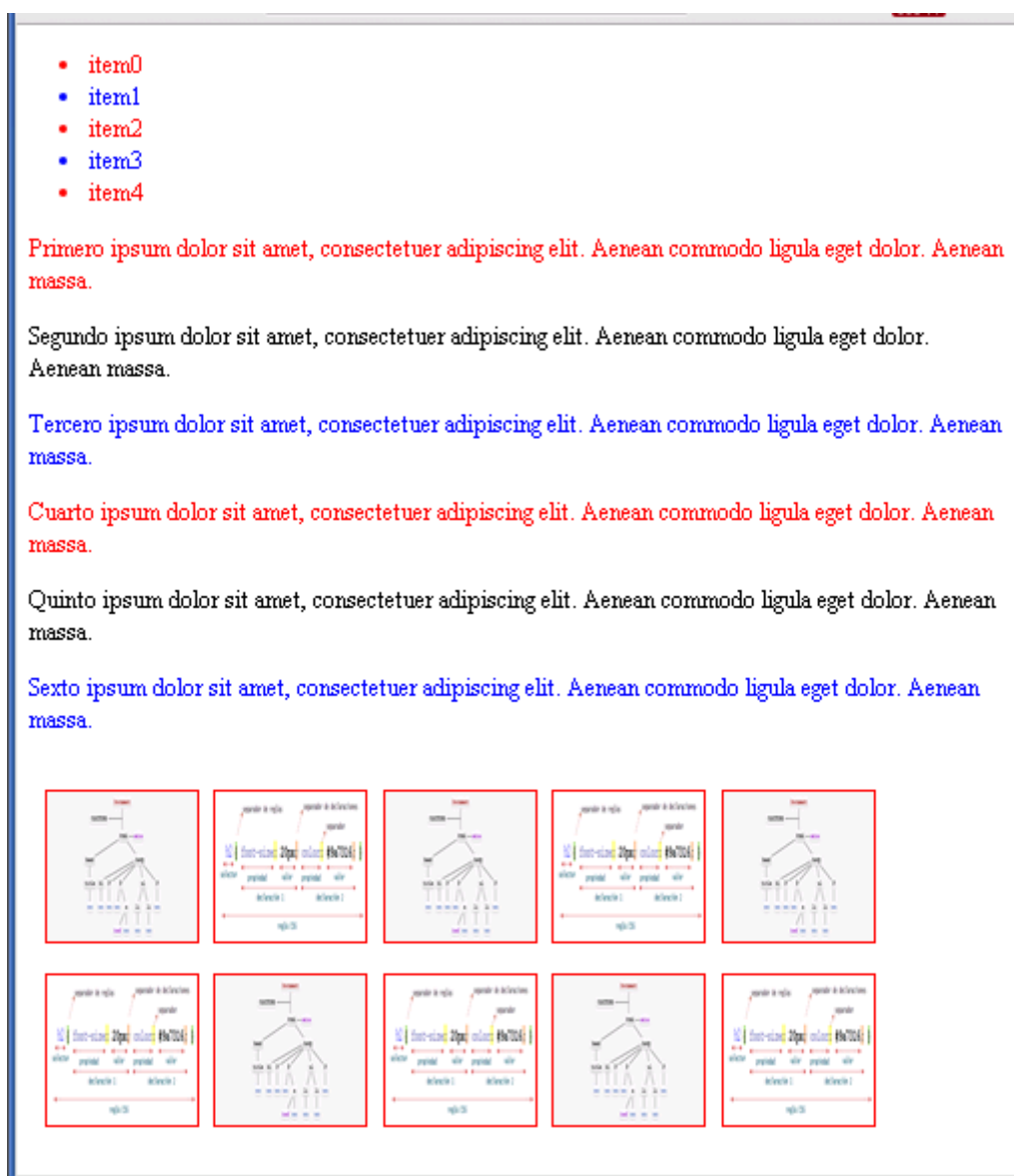
```
var myItems = [], $myList = $('#myList');

for (var i=0; i<100; i++) {
    myItems.push('<li>item ' + i + '</li>');
}

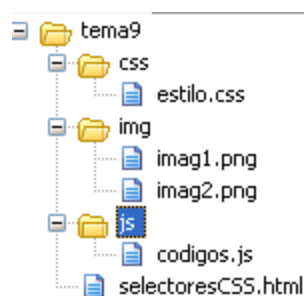
$myList.append(myItems.join(""));
```

## Laboratorio 9.1

En este laboratorio vamos a aplicar estilos, con selectores avanzados CSS, a una lista ul, a 6 párrafos y a 10 imágenes para obtener el siguiente resultado:



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Para empezar, debe crear una carpeta donde grabara todos sus archivos para realizar este ejercicio. La carpeta principal se llamará **tema9**. Luego, cree un sitio web con el nombre **tema9** en la carpeta principal.
2. Abra una hoja de estilo CSS e inserte el siguiente código:

```
p:nth-child(3n+2) { color:red; } /*primero asigna formato y luego cuenta 3*/
p:nth-child(3n+1) { color:blue; } /*cuenta 3 y asigna formato*/
```

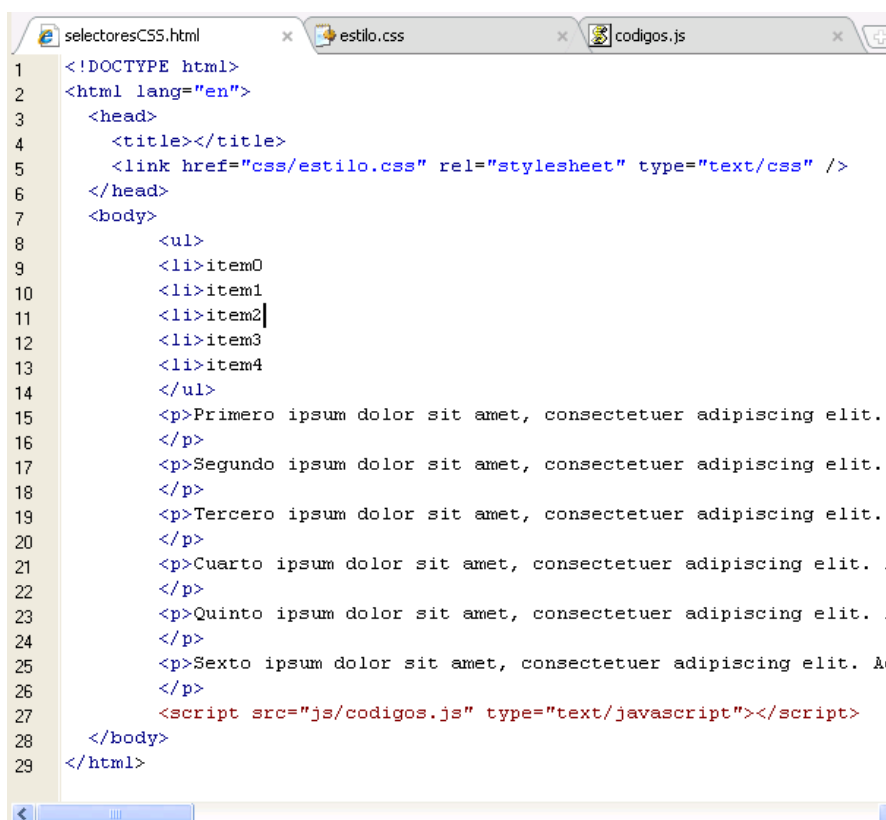
```
img{float:left;margin-top:20px;margin-left:10px;border:red solid 1px;}
img:nth-of-type(5n+1) {clear:left;}
```

```
li:nth-child(2n+1) {color:red; } /*color a todos los elementos impares de una lista */
li:nth-child(2n) { color:blue; } /*color a todos los elementos pares de una lista */
```

3. Grabe su archivo como **estilo.css** en la carpeta **css**
4. Cree una carpeta con el nombre **img** y grabe dos imágenes con el nombre **imag1.png** e **imag2.png**
5. Abra un archivo **js** e inserte el siguiente código:

```
for(var i=0;i<5;i++){
document.write("<img src='img/imag1.png' width='100' height='100'/>");
document.write("<img src='img/imag2.png' width='100' height='100'/>");
}
```

6. Grabe el archivo como **códigos.js** en la carpeta **js**
7. Abra una archivo **html** e insert el siguiente código:



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title></title>
5     <link href="css/estilo.css" rel="stylesheet" type="text/css" />
6   </head>
7   <body>
8     <ul>
9       <li>item0
10      <li>item1
11      <li>item2
12      <li>item3
13      <li>item4
14    </ul>
15    <p>Primero ipsum dolor sit amet, consectetur adipiscing elit.
16    </p>
17    <p>Segundo ipsum dolor sit amet, consectetur adipiscing elit.
18    </p>
19    <p>Tercero ipsum dolor sit amet, consectetur adipiscing elit.
20    </p>
21    <p>Cuarto ipsum dolor sit amet, consectetur adipiscing elit. A
22    </p>
23    <p>Quinto ipsum dolor sit amet, consectetur adipiscing elit. A
24    </p>
25    <p>Sexto ipsum dolor sit amet, consectetur adipiscing elit. Ae
26    </p>
27    <script src="js/codigos.js" type="text/javascript"></script>
28  </body>
29 </html>
```

8. Grabe el archivo como **selectoresCSS.html** y luego lo ejecuta para ver los resultados



# Resumen

1. jQuery es un conjunto de librerías JavaScript que han sido diseñadas específicamente para simplificar el desplazamiento de un documento HTML, la animación, la gestión de eventos.
2. Si queremos utilizar jQuery en nuestra página web lo primero será declarar el uso de la librería:

```
<script src="http://code.jquery.com/jquery-2.1.4.min.js"></script>
```

3. CSS 3 añade tres nuevos selectores de atributos:

```
elemento[atributo^="valor"]  
elemento[atributo$="valor"]  
elemento[atributo*="valor"]
```

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.arquitecturajava.com/jquery-find-vs-filter/>
- <http://api.jquery.com/filter/>

## 3.2 CARACTERISTICAS PRINCIPALES

### 3.2.1 Constructor

Generalmente, cuando se desea ejecutar Javascript después de la carga de la página, si no utilizamos ningún framework, lo más normal será utilizar un código como este:

```
window.onload = function () {  
    alert("cargado...");  
}
```

Pero esta sentencia, que carga una funcionalidad en el evento onload del objeto window, sólo se ejecutará cuando el navegador haya descargado completamente TODOS los elementos de la página, lo que incluye imágenes, iframes, banners, etc. lo que puede tardar bastante, dependiendo de los elementos que tenga esa página y su peso.

Pero en realidad no hace falta esperar todo ese tiempo de carga de los elementos de la página para poder ejecutar sentencias Javascript que alteren el DOM de la página. Sólo habría que hacerlo cuando el navegador ha recibido el código HTML completo y lo ha procesado al renderizar la página. Para ello, jQuery incluye una manera de hacer acciones justo cuando ya está lista la página, aunque haya elementos que no hayan sido cargados del todo. Esto se hace con la siguiente sentencia.

```
$(document).ready(function(){  
    //código a ejecutar cuando el DOM está listo para recibir instrucciones.  
});
```

Por dar una explicación a este código, digamos que con `$(document)` se obtiene una referencia al documento (la página web) que se está cargando. Luego, con el método `ready()` se define un evento, que se desata al quedar listo el documento para realizar acciones sobre el DOM de la página.

Existe una forma abreviada para `$(document).ready()` la cual podrá encontrar algunas veces; sin embargo, es recomendable no utilizarla en caso que este escribiendo código para gente que no conoce jQuery.

Forma abreviada para `$(document).ready()`

```
$(function() {  
    console.log('el documento está preparado');  
});
```

Además es posible pasarle a `$(document).ready()` una función nombrada en lugar de una anónima:

Pasar una función nombrada en lugar de una función anónima

```
function readyFn() {  
    // código a ejecutar cuando el documento este listo  
}  
  
$(document).ready(readyFn);
```

### 3.2.2 Iteración implícita

Los metodos de jQuery que realizan consultas trabajan con iteración implícita, por ejemplo, la consulta.

```
$('#h2')
```

Nos devuelve el conjunto de los titulos nivel 2 (h2), ahora si por ejemplo quisieramos cambiarle el color a un rojo (#F00), esto lo podriamos hacer de la siguiente manera.

```
$.each($('#h2'), function() {  
    $(this).css("color", "#F00");  
});
```

En este caso estamos ocupando el metodo each, en el cual para cada titulo en la selección se ejecuta una función la cual le cambia el color.

Existe una forma de hacer esto más "fácil" y es ocupando el comportamiento de iteración implícita de jQuery (o acción sobre el conjunto), en este caso la función css puede trabajar sobre toda una selección:

```
$('#h2').css('color', '#F00');
```

Recuerde que la etiqueta <script> debe estar insertada en el <body> después de insertar las etiquetas h1 y h2. De lo contrario no funcionará.

### 3.2.3 Consultas a través del DOM

jQuery básicamente nos permite hacer 3 tipos de consultas:

- Consultas CSS
- Consultas XPath
- Consultas Trasversales

Este manual explica las consultas CSS y algunos metodos transversales, dejando un poco de lado las consultas XPath ya que casi no se usan, pero es bueno que el alumno sepa que existen y que también son una forma de acceder a los elementos.

#### Trasversatilidad

Se ha explicado como funciona la anatomia CSS, falta explicar la transversatilidad, la cual es un conjunto de métodos definidos por jQuery para refinamiento de selectores, para explicar esto ocuparemos 2 funciones.

find: Nos permite realizar una consulta sobre un objeto jQuery.

filter: Nos permite realizar un filtrado, descartando elementos, sobre un objeto jQuery.

Imaginemos que tenemos una tabla que tiene como atributo id el valor idTabla, el cual queremos accecer a todos sus elementos th, una forma de hacerlo seria:

```
$("#idTabla th").css('background', '#ddd');
```

Otra alternativa es usar el metodo find para buscar en un objeto de jQuery el cual me representa una sub-rama del DOM en idTabla.

```
$("#idTabla").find('th').css('background', '#ddd');
```

Otro ejemplo, dado el siguiente código html:

```
<div id="padre">
<div id="hijo1">
<div id="nieto1"></div>
</div>
<div id="hijo2"></div>
</div>
```

El siguiente código jquery

```
$("#padre").find("div")
```

Encuentra “hijo1”, “hijo2” y “nieto1”, (y no el div “padre” contenido en el objeto \$.)

Este tipo de métodos denominados “transversales”, son los que nos ampliarán el campo de acción cuando excedamos las posibilidades de los selectores.

Considere una página con una lista simple ul:

```
<ul>
<li>list item 1</li>
<li>list item 2</li>
<li>list item 3</li>
<li>list item 4</li>
<li>list item 5</li>
<li>list item 6</li>
</ul>
```

Se puede aplicar el metodo filter al conjunto de items de la lista, de esta manera:

```
$( "li" ).filter( ":even" ).css( "background-color", "red" );
```

El resultado de este codigo jquery es aplicar fondo rojo a los items 1, 3, y 5, porque ellos coinciden con el selector even. Recuerde que :even (par) y :odd (impar) usan indices que empiezan en 0.

En el siguiente ejemplo, el código jquery busca todos los elementos p que son hijos de un elemento div y le aplica bordes

```
$( "div > p" ).css( "border", "1px solid gray" );
```

Si el codigo html fuera el siguiente:

```
<p>one</p>
<div><p>two</p></div>
<p>three</p>
```

Aplicaria borde al segundo elemento p porque esta insertada en una etiqueta div

El siguiente ejemplo cambia el color de todas las capas div, agrega color de borde a aquellos que tengan la clases middle.

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>filter demo</title>
<style>

div {
width: 60px;
height: 60px;
margin: 5px;
float: left;
border: 2px white solid;
}

</style>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
</head>
<body>

<div></div>
<div class="middle"></div>
<div class="middle"></div>
<div class="middle"></div>
<div class="middle"></div>
<div></div>

<script>

$( "div" )
.css( "background", "#c8ebcc" )
.filter( ".middle" )
.css( "border-color", "red" );

</script>

</body>
</html>
```

Resultado final:



### 3.2.4 Encadenamiento (chaining)

La habilidad de utilizar métodos en cadena nos permite realizar una gran cantidad de actividades en una sola sentencia, ahorrando líneas y redundancia de lógica en nuestro código.

Esta capacidad de encadenamiento no sólo nos facilita escribir y realizar de manera concisa operaciones poderosas que llegan a tener gran alcance, sino que también nos concede una mejora en la eficiencia, porque los conjuntos envueltos no tienen que ser recalculados cada vez que deseemos aplicar múltiples comandos o realizar operaciones sobre ellos.

La utilización del encadenamiento puede desembocar en la producción de muchas facilidades, dependiendo de los métodos utilizados en una cadena de mando, varios conjuntos envueltos se pueden generar o podemos aplicar operaciones sobre lo que acabamos de afinar.

En el siguiente código, se repite el objeto \$('h2') dos veces:

```
$('h2').css('color','red');  
$('h2').text('Cambiando el contenido a todos los elementos h2');
```

Para evitar esta repetición, se puede usar encadenamiento y el nuevo código sería así:

```
$('h2')  
    .css('color','red')  
    .text('Cambiando el contenido a todos los elementos h2');
```

Tenemos otro ejemplo de encadenamiento:

```
$("#box_prueba")  
    .css("backgroundColor", "red")  
    .slideUp(2000)  
    .slideDown(2000);
```

slideUp y slideDown son metodos para hacer animaciones de 2 segundos.

El siguiente ejemplo aplica diversos formatos css con encadenamiento a todas las etiquetas h1 ubicadas dentro del div con id content, selecciona tercer elemento h1 con el método eq(2), y le cambia el texto con el método html y el color azul:

```
$('#content')  
    .find('h1')  
    .css('color','red')  
    .eq(2)  
    .css('color','blue')  
    .html('nuevo texto para el tercer elemento h1');
```

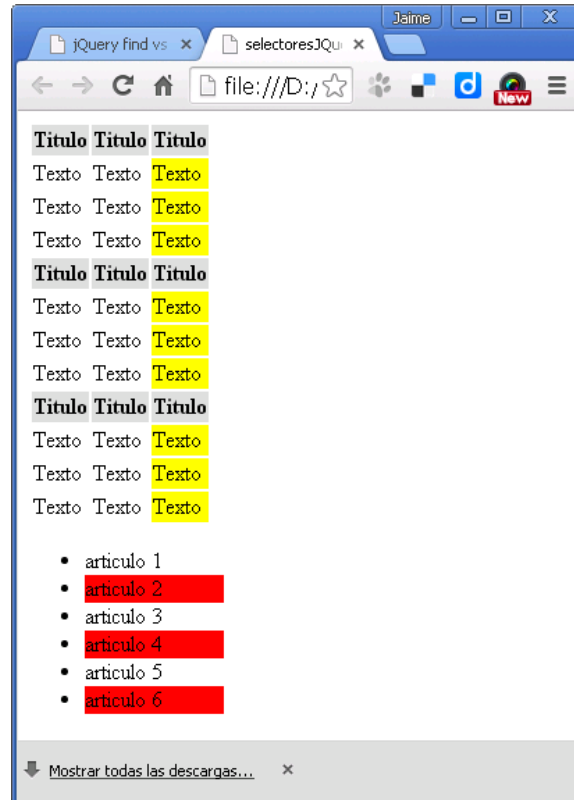
El encadenamiento es muy poderoso y es una característica que muchas bibliotecas JavaScript han adoptado desde que jQuery se hizo popular. Sin embargo, debe ser utilizado con cuidado. Un encadenamiento de métodos extensivo puede hacer un código extremadamente difícil de modificar y depurar. No existe una regla que indique que tan largo o corto debe ser el encadenado — pero es recomendable que tenga en cuenta este consejo.

**Restablecer la selección original utilizando el método \$.fn.end**

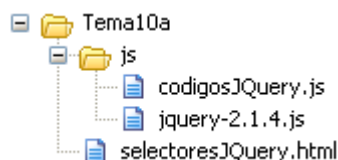
```
$('#content')
    .find('h1')
    .css('color','orange')// cambia el color a todos los h1
    .eq(2) // selecciona el tercer elemento h1
    .html('nuevo texto para el tercer elemento h1')
    .css('color','#F0F')
    .end() // reestablece la selección a todos los elementos h1
    .eq(0) // selecciona el primer elemento h1
    .html('nuevo texto para el primer elemento h1')
    .css('color','red')
```

## Laboratorio 10.1

En este laboratorio vamos a aplicar consultas con selectores JQuery para aplicar formato a todos los th y a la última columna de la tabla. También, aplicaremos formatos a los ítems de un listado ul para luego, obtener el siguiente resultado:



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Copie el archivo jquery-2.1.4.js en su carpeta js
2. Cree una archivo css e inserte el siguiente código:

```

//Crea una tabla
document.write("<table id='idTabla'>");
for(var i=0;i<3;i++){
  with(document){
    write("<tr><th>Titulo<th>Titulo<th>Titulo");
    write("<tr><td>Texto<td>Texto<td>Texto");
    write("<tr><td>Texto<td>Texto<td>Texto");
    write("<tr><td>Texto<td>Texto<td>Texto");
  }
}
document.write("</table>");
  
```



```
//Crea una lista ul con 6 elementos
document.write("<ul>");
for(var i=1;i<7;i++)
document.write("<li>articulo "+i);
document.write("</ul>");

//aplica estilos css a la lista usando jquery
$("li").filter(":odd").css("background-color", "red")
$("li").filter(":odd").css("width", "100px");

//aplica estilos css a la tabla usando jquery
$("#idTabla").find('th').css('background', '#ddd');
$("tr").find("td:last").css("background-color", "yellow");
```

3. Grabe el archivo con el nombre codigosjQuery.css.
4. Abra un archivo html y escriba el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>

<title></title>
<!-- agregando libreria jQuery -->
<script type='text/javascript' src='js/jquery-2.1.4.js'></script>
<!-- agregando archivo js con codigos jQuery -->
<script type='text/javascript' src='js/codigosjQuery.js'></script>
</head>

<body>

</body>
</html>
```

5. Grabe el archivo selectoresjQuery.html y luego ejecute la pagina web para ver los resultados

# Resumen

1. Es una buena practica cuando se trabaja con JavaScript esperar a que el documento termine de cargar para ejecutar nuestro codigo, jQuery tiene una funcion especifica para esto:

```
$(document).ready(function(){  
    //código a ejecutar cuando el DOM está listo para recibir instrucciones.  
});
```

2. La iteración implícita de jQuery (o acción sobre el conjunto), trabajaa sobre toda una selección. Por ejemplo el siguiente código aplica color rojo a todos los textos insertados con la etiqueta h2:

```
$('h2').css('color', '#F00');
```

3. jQuery básicamente nos permite hacer 3 tipos de consultas:

- Consultas CSS
- Consultas xPath
- Consultas Trasversales

.

4. Los métodos para realizar consultas o selección de tipo transversal son find y filter

- find: Nos permite realizar una consulta sobre un objeto jQuery.
- filter: Nos permite realizar un filtrado, descartando elementos, sobre un objeto jQuery.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://librojquery.com/#conceptos-básicos-de-jquery>
- <http://www.emenia.es/curso-de-jquery-3-selectores-segunda-parte/>

## 3.3 MANEJO DE EVENTOS

### 3.3.1 Introduccion a los eventos

jQuery define una lista de eventos y funciones para la administración de los mismos, la sintaxis por defecto para un manejador de evento es de la siguiente forma  
\$.fn.nombreEvento.

```
$('#Selector').nombreEvento( function(event){  
    //funcion que administra el evento.  
    //this es el elemento que disparo el evento.  
})
```

O también, esta otra forma:

```
$('#Selector').on( function(event){  
    //funcion que administra el evento.  
    //this es el elemento que disparo el evento.  
})
```

Aquí es importante resaltar que this contendrá la instancia del elemento que disparó el evento, por ejemplos si a los enlaces(a) le agregamos el evento click, this contendrá la instancia del enlace específico sobre el cual hallamos hecho click, analice el siguiente código.

jQuery define varios eventos. Para agregar un manejador de dicho evento basta con agregar una función en la llamada de dicho evento. Esta función puede recibir un argumento event el cual es útil para obtener información de dicho evento ó para cambiar el comportamiento del mismo. Por ejemplo, el siguiente código al hacer clic en cualquier enlace con nombre de class testClick, muestra su mismo texto de enlace en un elemento con class llamado log

```
$('#a.testClick').click(function(event){  
    $('#log').html( '<strong>Resultado:</strong> ' + $(this).text());  
    $('#log').css('color','F00');  
    event.preventDefault();  
});
```

El código html sería así:

```
<a class='testClick' href='http://adsl.org.mx'>enlace 1</a>  
<a class='testClick' href='http://www.cibertec.edu.pe'>enlace 2</a>  
<a class='testClick' href='js/codigos.js'>enlace 3</a>  
<p class="log"></p>
```

Al hacer clic en el enlace 1, se muestra en el párrafo llamado log el texto del enlace, o sea enlace 1. El método preventDefault() evita que el enlace abra la dirección url.

También se puede usar un método reducido el cual tiene la siguiente forma:

```
$('#p').click(function(){  
    console.log('click'); //muestra mensaje en la consola  
});
```

Por lo tanto, el ejemplo anterior, se reduciría así:

```
$('#a.testClick').click(function(){
    $('#.log').html( '<strong>Resultado:</strong> ' + $(this).text());
    $('#.log').css('color','#F00');
    event.preventDefault();
});
```

O de esta otra forma:

```
$('#a.testClick').on('click',function(){
    $('#.log').html( '<strong>Resultado:</strong> ' + $(this).text());
    $('#.log').css('color','#F00');
    event.preventDefault();
});
```

Otra forma de vincular un evento es usando el método bind, de esta manera:

```
$('#a.testClick').bind('click',function(){
    $('#.log').html( '<strong>Resultado:</strong> ' + $(this).text());
    $('#.log').css('color','#F00');
    event.preventDefault();
});
```

### Ejecutar una sola vez el evento

A veces puede necesitar que un controlador particular se ejecute solo una vez — y después de eso, necesite que ninguno más se ejecute, o que se ejecute otro diferente. Para este propósito jQuery provee el método `$.fn.one`.

### Cambiar controladores utilizando el método `$.fn.one`

El método `$.fn.one` es útil para situaciones en que necesita ejecutar cierto código la primera vez que ocurre un evento en un elemento, pero no en los eventos sucesivos.

```
$('#p').one('click',function(){
    console.log('Se clickeó al elemento por primera vez');
    $(this).click(function(){ console.log('Se ha clickeado nuevamente');});
});
```

El siguiente código usa el método `one` para que la primera vez que se haga clic en un enlace, se cambie todos los enlaces a color verde. A partir del segundo clic en cada enlace aparecerá el texto en el elemento log con el color rojo:

```
$('#a.testClick').one('click', function(){
    $(this).css('color','#0F0');
    event.preventDefault();
    $(this).click(function() {
        $('#.log').html( '<strong>Resultado:</strong> ' + $(this).text() );
    });
});
```

```
    $('log').css('color', '#F00');  
    event.preventDefault();  
  });  
});
```

### Desvincular eventos

Para desvincular (en inglés *unbind*) un controlador de evento, puede utilizar el método `$.fn.unbind` pasándole el tipo de evento a desconectar. Si se pasó como adjunto al evento una función nombrada, es posible aislar la desconexión de dicha función pasándola como segundo argumento.

El siguiente código, desvincula todos los controladores del evento click en una selección:

```
$('.p').unbind('click');
```

### Manejadores de eventos

jQuery maneja un listado con los distintos manejadores eventos ordenados por los tipos eventos de ratón, eventos de teclado o cualquiera de los **dos**.

#### Eventos del Mouse

A continuación podemos ver una lista de los eventos que se pueden definir en jQuery que tienen que ver con el mouse. Es decir, cómo definir eventos cuando el usuario realiza diferentes acciones con el ratón sobre los elementos de la página.

`click()`

Sirve para generar un evento cuando se produce un clic en un elemento de la página.

`dblclick()`

Para generar un evento cuando se produce un doble clic sobre un elemento.

`hover()`

Esta función en realidad sirve para manejar dos eventos, cuando el ratón entra y sale de encima de un elemento. Por tanto espera recibir dos funciones en vez de una que se envía a la mayoría de los eventos.

`mousedown()`

Para generar un evento cuando el usuario hace clic, en el momento que presiona el botón e independientemente de si lo suelta o no. Sirve tanto para el botón derecho como el izquierdo del ratón.

`mouseup()`

Para generar un evento cuando el usuario ha hecho clic y luego suelta un botón del ratón. El evento `mouseup` se produce sólo en el momento de soltar el botón.

`mouseenter()`

Este evento se produce al situar el ratón encima de un elemento de la página.

`mouseleave()`

Este se desata cuando el ratón sale de encima de un elemento de la página.

`mousemove()`

Evento que se produce al mover el ratón sobre un elemento de la página.

`mouseout()`

Este evento sirve para lo mismo que el evento `mouseout` de JavaScript. Se desata cuando el usuario sale con el ratón de la superficie de un elemento.

`mouseover()`

Sirve para lo mismo que el evento `mouseover` de Javascript. Se produce cuando el ratón está sobre un elemento, pero tiene como particularidad que puede producirse varias veces mientras se mueve el ratón sobre el elemento, sin necesidad de haber salido.

`toggle()`

Sirve para indicar dos o más funciones para ejecutar cosas cuando el usuario realiza clics, con la particularidad que esas funciones se van alternando a medida que el usuario hace clics.

## Eventos del teclado

A continuación se muestran los eventos que pueden modelizarse como respuesta a la pulsación de teclas del teclado.

`keydown()`

Este evento se produce en el momento que se presiona una tecla del teclado, independientemente de si se libera la presión o se mantiene. Se produce una única vez en el momento exacto de la presión.

`keypress()`

Este evento ocurre cuando se digita un carácter, o se presiona otro tipo de tecla. Es como el evento `keypress` de Javascript, por lo que se entiende que `keypress()` se ejecuta una vez, como respuesta a una pulsación e inmediata liberación de la tecla, o varias veces si se pulsa una tecla y se mantiene pulsada.

`keyup()`

El evento `keyup` se ejecuta en el momento de liberar una tecla, es decir, al dejar de presionar una tecla que teníamos pulsada.

**Nota:** a través del objeto evento, que reciben las funciones que indiquemos como parámetro de estos métodos, podemos saber qué tecla se está pulsando, aparte de otras muchas informaciones.

## Eventos combinados teclado o mouse

Ahora mostramos varios eventos que pueden producirse tanto por el ratón como por el teclado, es decir, como resultado de una acción con el ratón o como resultado de presionar teclas en el teclado.

`focusin()`

Evento que se produce cuando el elemento gana el foco de la aplicación, que puede producirse al hacer clic sobre un elemento o al presionar el tabulador y situar el foco en ese elemento.

focusout()

Ocurre cuando el elemento pierde el foco de la aplicación, que puede ocurrir cuando el foco está en ese elemento y pulsamos el tabulador, o nos movemos a otro elemento con el ratón.

focus()

Sirve para definir acciones cuando se produce el evento focus de Javascript, cuando el elemento gana el foco de la aplicación.

### 3.3.2 Evento click

El evento click es disparado cuando con el mouse le damos click sobre un elemento seleccionado, la forma que tiene este evento es la siguiente:

```
$('#Selector'). click ( function(event){
    //funcion que administra el evento.
    //this es el elemento que disparo el evento.
})
```

Por ejemplo

```
$("#a.button").click(
    function(event) {
        console.log('Manejador para el evento click del enlace(a) con clase(.) button');
        event.preventDefault();
    });
```

En el ejemplo anterior vemos como al evento click le agregamos una funcion como manejador de dicho evento, esta recibe el objeto event el cual ocupamos para ejecutar el metodo preventDefault este metodo detiene el comportamiento por defecto de dicho evento, en este caso tenemos un link(a) que al darle click lo comun seria que siguiera en enlace definido en el atributo href.

El siguiente código detecta si se hizo clic en la imagen:

```
$(document).ready(function(){
    $("#img[name=imagen]").click(function () {
        //$("#img").click(function () {      Estos 4 selectores también se aplican
        //$("#idimagen").click(function () {
        //$("#img[class=claseimagen]").click(function () {
        //$("#.claseimagen").click(function () {
        alert("has hecho click en la imagen");
    });
});
```

El siguiente ejemplo muestra el evento clic sobre un botón:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script>
$(document).ready(function(){
```

```
    $("button").click(function(){
        $("#p1").css("color", "red").slideUp(2000).slideDown(2000);
    });
});
</script>
</head>
<body>
<p id="p1">jQuery is fun!!</p>
<button>Click me</button>
</body>
</html>
```

### 3.3.3 Comportamiento en cola

jQuery tiene un comportamiento en cola para los eventos (el primero en entrar es el primero en ejecutarse), para explicar esto observe el siguiente código:

```
    $("a")
        .click(function() {
            console.log('Ejecución del manejador1 para el evento click del enlace');
        })
        .click(function() {
            console.log('Ejecución del manejador2 para el evento click del enlace');
        });
```

¿Que cree que suceda cuando le demos click a un enlace (a) en la página?.

La respuesta la podemos encontrar al reflexionar que sucede cuando mandamos a llamar multiples veces el evento `document.onReady`.

Lo que hace jQuery es ocupar una cola para administrar los eventos, de esta forma se pueden agregar multiples funciones sobre el mismo evento/elemento, por lo cual el resultado en la consola seria:

```
Ejecución del manejador1 para el evento click del enlace
Ejecución del manejador2 para el evento click del enlace
```

El siguiente código muestra el comportamiento de multiples eventos en cola. Al hacer clic en el elemento `target` se mostrará el primer mensaje, luego el segundo y por ultimo el tercero.

```
$('#target')
    .bind('click',function(event) {
        alert('Hello!');
    })
    .bind('click',function(event) {
        alert('Hello again!');
    })
    .bind('click',function(event) {
        alert('Hello yet again!');
    });
```



### 3.3.4 Evento hover

El evento hand over o hover es disparado cuando pasamos el cursor por encima de algun elemento, la sintaxis basica es la siguiente.

```
$("#a").hover(function() {  
    console.log('Ejecución del manejador para el evento hover del enlace');  
});
```

Cabe mencionar que este evento puede soportar 2 manejadores, el primero es ejecutado cuando el cursor pase por encima del elemento y el segundo es ejecutado cuando el cursor se quita de dicho elemento, p.e.:

```
$("#a").hover(  
    function() {  
        $(this).css('color', 'red');  
    },  
    function() {  
        $(this).css('color', 'blue');  
    }  
);
```

En este caso estamos agregando 2 manejadores el resultado sera que cuando pasemos el mouse por encima de un enlace el color de la fuente se cambiara a rojo y cuando quitemos el cursor el color sera azul, veamos otro ejemplo:

#### código HTML:

```
<div id='box'></div>
```

#### CSS

```
#box{  
    width: 100px;  
    height: 100px;  
    background: orange;  
    display: block;  
    border-radius: 5px;  
}
```

#### código JS:

```
$("#box").hover(  
    function () {  
        $(this).animate({ 'width': "300px", "height" : "300px"});  
    },  
    function () {  
        $(this).animate({ 'width': "100px", "height" : "100px"});  
    }  
);
```

En este caso estamos agregando 2 manejadores el resultado será que cuando pasemos el mouse por encima de la capa div, el ancho y alto de la capase agrandará y cuando quitemos el cursor el tamaño de la caja volverá a su valor inicial.

### 3.3.5 Eventos del teclado

Los eventos de teclado, en principio, son tres: keydown, keypress y keyup. Realmente no actúan por separado, sino que se produce una combinación de éstos al ir presionando y soltando las teclas.

Si pulsamos y soltamos una tecla, primero se produce un evento keydown, al presionar la tecla, luego un keypress y por último un keyup al soltarla.

Si hacemos una pulsación prolongada de una tecla este esquema varía, pues se produce un keydown y luego un keypress. Mientras se mantiene pulsada la tecla en bucle se van produciendo eventos keydown y keypress, repetidas veces hasta que finalmente se suelta la tecla y se produce un keyup.

En el caso de las teclas CTRL, Mayúsculas o ALT, se producen múltiples keydown hasta que se suelta la tecla y se produce un keyup. Es decir, al pulsar una de estas teclas no se produce el evento keypress.

#### Secuencia de eventos de teclado

Vamos a aprender cuál es la secuencia con la que se producen los eventos de teclado, con un pequeño ejemplo práctico. Se trata de hacer una función que detecte cualquier evento de teclado, muestre el tipo de evento que ha ocurrido y lo muestre en la página. Así podremos ver los eventos que se producen, sean cuales sean, y en qué orden.

Primero podríamos definir la función que va a procesar los eventos:

```
function operaEvento(evento){  
    $("#loescrito").html($("#loescrito").html() + evento.type + ": " + evento.which + ", ")  
}
```

Esta función recibe el evento y escribe en una capa el tipo de evento, que se consigue con la propiedad type del objeto evento, y luego un código de la tecla pulsada, que se consigue con la propiedad which del objeto evento.

El tipo de evento type es otra de las propiedades que encontramos en el objeto evento que recibe la función que tiene el código a ejecutar por el evento. Esta propiedad type simplemente es un string con la cadena que identifica el tipo de evento que se está procesando ("keydown", "keyup", "click" o cualquier otro). La tecla pulsada se obtiene con la propiedad which.

Ahora podríamos hacer que cualquier evento de teclado invoque esta función con el código:

```
$(document).keypress(operaEvento);  
$(document).keydown(operaEvento);  
$(document).keyup(operaEvento);
```

Como hemos asociado los eventos al objeto document de Javascript, estos eventos se pondrán en marcha cada vez que se pulse una tecla, independientemente de dónde esté el foco de la aplicación (o donde esté escribiendo el usuario).

Esto se puede ver en marcha en una página aparte.

Se presenta el código completo del anterior ejemplo:

```
<html>
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script src="../../jquery-1.4.1.min.js"></script>
<script>
function operaEvento(evento){
    $("#loescrito").html($("#loescrito").html() + evento.type + ": " + evento.which + ", ")
}
$(document).ready(function(){
    $(document).keypress(operaEvento);
    $(document).keydown(operaEvento);
    $(document).keyup(operaEvento);
})
</script>
</head>
<body>
<h1>Eventos de teclado en jQuery</h1>
<div id="loescrito"></div>
</body>
</html>
```

### Averiguar qué tecla fue pulsada

A través de la propiedad which del objeto evento de jQuery podemos saber qué tecla ha sido pulsada cuando se produce el evento de teclado. Esta propiedad contiene un número entero con el código Unicode la tecla pulsada. Haremos un ejemplo para explicarlo.

Tenemos un textarea y escribiendo algo en él, mostraremos la tecla pulsada en una capa, independiente del textarea. Este será el código HTML que necesitaremos para el ejemplo:

```
<form>
<textarea cols=300 rows=2 id="mitexto">Escribe algo aquí!</textarea>
<br>
<b>Tecla pulsada:</b>
<br>
<div id="loescrito"></div>
</form>
```

Ahora definiremos con jQuery el evento keypress, para mostrar la tecla pulsada.

```
$("#mitexto").keypress(function(e){
    e.preventDefault();
    $("#loescrito").html(e.which + ": " + String.fromCharCode(e.which));
});
```

Con `e.preventDefault()`; hacemos que no se escriba nada en el textarea, osea, estamos inhibiendo el comportamiento habitual del evento, que es escribir las teclas en el textarea, que no tiene mucho que ver con nuestro ejemplo, pero que está bien para ver cómo funciona.

Luego escribimos en la capa con id "loescrito" el código de Unicode de esa tecla y luego su conversión a un carácter normal, a través de la función estática de la clase `String.fromCharCode()`.

El código completo del ejercicio es el siguiente.

```
<html>
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script src="../jquery-1.4.1.min.js"></script>
<script>
$(document).ready(function(){
    $("#mitexto").keypress(function(e){
        e.preventDefault();
        $("#loescrito").html(e.which + ": " + String.fromCharCode(e.which))
    });
})
</script>
</head>
<body>
<h1>Eventos de teclado en jQuery</h1>
<h2>Averiguar qué tecla se está pulsando</h2>
<form>
<textarea cols=300 rows=2 id="mitexto">Escribe algo aquí!</textarea>
<br>
<b>Tecla pulsada:</b>
<br>
<div id="loescrito"></div>
</form>
</body>
</html>
```

### 3.3.6 Borrar Eventos

Para esta acción, contamos con varios métodos como `unbind()`, `die()`, pero la nueva instrucción es `off`. La sintaxis de `off()` es la siguiente:

```
$(elements).off( [ events ] [, selector] [, handler] );
```

Con `off()`, todos los parámetros son opcionales. Cuando se utiliza en su forma más simple, `$(elements).off()`, se eliminan todos los eventos asociados al conjunto seleccionado.

Por ejemplo para eliminar el evento click en H1 podríamos hacer.

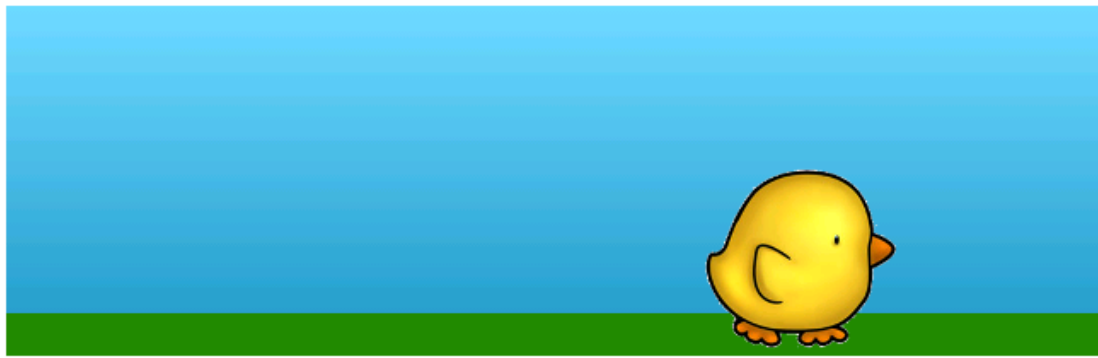
```
$("#h1").off('click');
```

Con esto eliminaríamos todos los eventos click del h1, si quisieramos eliminar un unico evento click, podríamos hacer uso de los espacios de nombres.

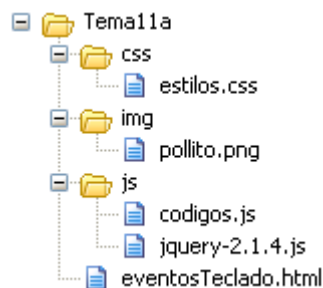
## Laboratorio 11.1

En este laboratorio vamos a aplicar eventos de teclado con JQuery para realizar una animación de tal forma que al hacer clic con la tecla derecha el pollito se mueva a la derecha y así por el contrario.

**Utilice las flechas de su teclado para darle animación al pollito :**



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Copie el archivo jquery-2.1.4.js en su carpeta js
2. Cree un archivo css e inserte el siguiente código:

```
#escenario {  
  width: 100%;  
  min-width: 500px;  
  height: 180px;  
  background: #6FD9FF;  
  background-repeat: repeat-x;  
  background-image: -moz-linear-gradient(top, #6FD9FF, #289FCB);  
  background-image: -ms-linear-gradient(top, #6FD9FF, #289FCB);  
  background-image: -webkit-gradient(linear, left top, left bottom, from(#6FD9FF),  
to(#289FCB));  
  background-image: -webkit-linear-gradient(top, #6FD9FF, #289FCB);  
  background-image: -o-linear-gradient(top, #6FD9FF, #289FCB);  
  background-image: linear-gradient(top, #6FD9FF, #289FCB);
```

```

    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#6FD9FF',
endColorstr='#289FCB',GradientType=0 );
    border-bottom: 25px solid #208A00;
}
#pollito {
    position: absolute;
    margin-left: 400px;
    margin-top: 70px;
    background: URL('../img/pollito.png')no-repeat;
    width: 161px;
    height: 134px;
}

```

3. Guarde su archivo en la carpeta css con el nombre estilos.css.
4. Cree una archivo js en la carpeta js e inserte el siguiente código:

```

$(function() {
    $pollito = $("#pollito");
    $escenario = $("#escenario");
    pos_x = 400;

    $('body').keydown(function(event) {
        switch(event.which) {
            case 39:
                $pollito.css("background-position", "0 0");
                if(pos_x < ($escenario.width() - $pollito.width()) ) {
                    pos_x += 10;
                    $pollito.css("margin-left", pos_x + "px");
                }
                break;
            case 37:
                $pollito.css("background-position", "-161 0");
                if(pos_x > 0) {
                    pos_x -= 10;
                    $pollito.css("margin-left", pos_x + "px");
                }
                break;
            default:
                console.log('Tecla: '+ event.which);
        }
    });
});

```

5. Guarde su archivo en la carpeta js con el nombre codigos.js.
6. Cree una archivo html e inserte el siguiente código:

```
<div id="escenario"><div id="pollito"></div></div>
```

Utilice las flechas de su teclado para darle animación al pollito.

# Resumen

1. jQuery define una lista de eventos y funciones para la administración de los mismos, la sintaxis por defecto para un manejador de evento es de la siguiente forma `$.fn.nombreEvento`.

```
$('#Selector').nombreEvento( function(event){  
    //funcion que administra el evento.  
    //this es el elemento que disparo el evento.  
})
```

O también, esta otra forma:

```
$('#Selector').on('nombreEvento',function(){  
    //funcion que administra el evento.  
    //this es el elemento que disparo el evento.  
});
```

2. El evento hand over o hover es disparado cuando pasamos el cursor por encima de algun elemento, y puede soportar 2 manejadores, el primero es ejecutado cuando el cursor pase por encima del elemento y el segundo es ejecutado cuando el cursor se quita de dicho elemento la sintaxis basica es la siguiente.

```
$("a").hover(  
function() {  
    console.log('Ejecución del manejador para el evento hover del enlace');},  
function() {  
    console.log('Ejecución del manejador para el evento hover del enlace');}  
);
```

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- o <http://fernetjs.com/2012/04/manejando-eventos-con-jquery/>
- o <http://codehero.co/jquery-desde-cero-eventos/>
- o [http://librosweb.es/libro/fundamentos\\_jquery/capitulo\\_5/vincular\\_eventos\\_a\\_elementos.html](http://librosweb.es/libro/fundamentos_jquery/capitulo_5/vincular_eventos_a_elementos.html)
- o <https://www.codecademy.com/courses/web-beginner-en-1NG7i/0/1>





## 3.4 EFECTOS Y MODIFICACIONES SOBRE EL DOM

### 3.4.1 Introduccion a los efectos

Las funciones de efectos son los métodos jQuery que realizan un cambio en los elementos de la página de manera suavizada, es decir, que alteran las propiedades de uno o varios elementos progresivamente, en una animación a lo largo de un tiempo.

Por poner un ejemplo, tenemos el método `fadeOut()`, que realiza un efecto de opacidad sobre uno o varios elementos, haciendo que éstos desaparezcan de la página con un fundido de opaco a transparente. El complementario método `fadeIn()` hace un efecto de fundido similar, pero de transparente a opaco. Como éstos, tenemos en jQuery numerosos métodos de efectos adicionales como `animate()`, `slideUp()` y `slideDown()`, etc. En la propia documentación del framework, en el apartado Effects de la referencia del API, podremos ver una lista completa de estas funciones de efectos.

#### Cola de efectos por defecto

Cuando invocamos varias funciones de efectos de las disponibles en jQuery, éstas se van introduciendo en una cola de efectos predeterminada, llamada "fx". Cada elemento de la página tiene su propia cola de efectos predeterminada y funciona de manera automática. Al invocar los efectos se van metiendo ellos mismos en la cola y se van ejecutando automáticamente, uno detrás de otro, con el orden en el que fueron invocados.

```
capa = $("#micapa");  
capa.fadeOut();  
capa.fadeIn();  
capa.slideUp();  
capa.slideDown();
```

Las funciones de efectos, una detrás de otra, se invocan en un instante, pero no se ejecutan todas a la vez, sino que se espera que acabe la anterior antes de comenzar la siguiente. Por suerte, jQuery hace todo por su cuenta para gestionar esta cola.

Como decimos, cada elemento de la página tiene su propia cola de efectos y, aunque incluso podríamos crear otras colas de efectos para el mismo elemento, en la mayoría de los casos tendremos suficiente con la cola por defecto ya implementada

### 3.4.2 Efectos incorporados

Antes de mencionar los diferentes efectos que podemos encontrar en jQuery hay que recalcar que existen plugins que podemos utilizar con efectos ya creados e implementarlos, sin embargo ahora mencionaremos solamente el código nativo y su implementación para que puedas agregarlo a tus páginas a tu conveniencia.

Para utilizarlos, debemos indicar dos cosas, el elemento sobre el cual se va a aplicar el efecto y el enlace que ejecutará la función. Ambos deberían ser identificados con un ID único. Por ejemplo:

```
<div id="nombreContenido"> ..... el contenido ..... </div>  
<a href="javascript:void(0);" id="nombreEnlaceMostrar"> MOSTRAR </a>  
<a href="javascript:void(0);" id="nombreEnlaceOcultar"> OCULTAR </a>
```

La función sería algo así:

```
<script type='text/javascript'>
//
    $(document).ready(function(){
        $("#nombreEnlaceMostrar").click(function () { $("#nombreContenido").show(); });
        $("#nombreEnlaceOcultar").click(function () { $("#nombreContenido").hide(); });
    });
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="138 263 862 324" data-label="Text"><p>Recordemos que CSS tiene una propiedad para alterar la opacidad de los elementos. Todos los valores de Opacity se expresan con números de 0 al 1. Con un valor de cero haría que el elemento fuera totalmente transparente y opacity con un valor de 1 sería totalmente opaco.</p></div><div data-bbox="138 336 832 369" data-label="Text"><p>Con los métodos de fading de jQuery se puede cambiar esa propiedad. Existen tres métodos para crear efectos de fundido, los siguientes:</p></div><div data-bbox="138 382 297 399" data-label="Section-Header"><h3><b>Método fadeOut()</b></h3></div><div data-bbox="138 397 862 443" data-label="Text"><p>Este método hace que el elemento que lo recibe desaparezca de la página a través del cambio de su opacidad, haciendo una transición suavizada que acaba con el valor de opacity cero.</p></div><div data-bbox="138 457 280 474" data-label="Section-Header"><h3><b>Método fadeIn()</b></h3></div><div data-bbox="138 473 852 549" data-label="Text"><p>El método fadeIn() hace que el elemento que lo recibe aparezca en la página a través del cambio de su opacidad, haciendo una transición suavizada que acaba con el valor de opacity 1. Este método sólo podremos observarlo si el elemento sobre el que lo invocamos era total o parcialmente transparente, porque si era opaco al hacer un fadeIn() no se advertirá ningún cambio de opacidad.</p></div><div data-bbox="138 561 287 578" data-label="Section-Header"><h3><b>Método fadeTo()</b></h3></div><div data-bbox="138 577 842 639" data-label="Text"><p>El tercer método para hacer efectos de fundidos es fadeTo() y es el más versátil de todos, puesto que permite hacer cualquier cambio de opacidad, a cualquier valor y desde cualquier otro valor. Este método recibe la duración deseada para el efecto, el valor de opacidad al que queremos llegar y una posible función callback.</p></div><div data-bbox="138 666 699 684" data-label="Section-Header"><h3><b>Ejemplos con efectos de fundido fadeOut() y fadeIn() en jQuery</b></h3></div><div data-bbox="138 683 855 729" data-label="Text"><p>Para ilustrar el modo en el que se pueden hacer efectos de fundido con el cambio de opacidad hemos hecho un ejemplo de página donde se pueden ver todos los métodos de fading en funcionamiento, con algunas variantes interesantes.</p></div><div data-bbox="138 742 547 759" data-label="Text"><p>En el ejemplo vamos a tener una lista como esta:</p></div><div data-bbox="138 772 443 847" data-label="Text"><pre>&lt;ul id="milista"&gt;
  &lt;li id="e1"&gt;Elemento 1&lt;/li&gt;
  &lt;li id="e2"&gt;Segundo elemento&lt;/li&gt;
  &lt;li id="e3"&gt;Tercer LI&lt;/li&gt;
&lt;/ul&gt;</pre></div><div data-bbox="138 861 813 894" data-label="Text"><p>Como vemos, tanto la lista (etiqueta UL) como los elementos (etiquetas LI) tienen identificadores (atributos id) para poder referirnos a ellos desde jQuery.</p></div><div data-bbox="138 950 440 964" data-label="Page-Footer">CARRERA DE COMPUTACIÓN E INFORMÁTICA</div><div data-bbox="782 950 862 964" data-label="Page-Footer">CIBERTEC</div>
```

Ahora veamos cómo hacer que la lista desaparezca con un fundido hacia transparente, a partir de una llamada a `fadeOut()`.

```
$("#milista").fadeOut();
```

Como se puede ver, `fadeOut()` en principio no recibe ningún parámetro. Aunque luego veremos que le podemos pasar un parámetro con una función callback, con código a ejecutarse después de finalizado el efecto.

Este sería el código para que la lista vuelva a aparecer, a través de la restauración de su opacidad con una llamada a `fadeIn()`.

```
$("#milista").fadeIn();
```

### Ejemplo con `fadeTo()`

El método `fadeTo` es bastante más versátil, como ya se había adelantado. Para hacer un ejemplo interesante con este método tenemos que ver cómo se le pueden pasar distintos valores de opacidad y para ello hemos creado un campo select con distintos valores.

```
<select name="opacidad" id="selopacidad">
  <option value="0.2">20%</option>
  <option value="0.5">50%</option>
  <option value="0.8">80%</option>
  <option value="1">100%</option>
</select>
```

Como se puede ver, este SELECT tiene diferentes OPTION con algunos valores de opacidad. Los valores (atributos value de los OPTION) son números entre 0 y 1. Ahora vamos a mostrar el código de un evento que asociaremos a este campo SELECT, para ejecutar acciones cuando el usuario cambia el valor que aparece en él. Cuando el SELECT cambie, queremos actualizar el valor de opacity de los elementos H1 de la página.

```
$("#selopacidad").change(function(e){
  var opacidad_deseada = e.target.options[e.target.selectedIndex].value
  $("h1").fadeTo(1000,opacidad_deseada);
});
```

En este código estamos definiendo un evento "onchange" sobre el SELECT anterior. En la primera línea de la función se está extrayendo la opacidad deseada y para ello se accede a la propiedad target del objeto evento que se recibe en la función que enviamos al método `change()`.

En el objeto evento, target es una referencia al objeto del DOM sobre el que se está codificando el evento. Es decir, en este ejemplo, `e.target` es una referencia al campo SELECT sobre el que estamos definiendo el evento.

Con `e.target.options[]` tengo el array de options que hay dentro de ese SELECT. Con `e.target.selectedIndex` obtengo el índice del elemento seleccionado, para poder acceder a la opción seleccionada a través del array de options.

Con `e.target.options[e.target.selectedIndex].value` estamos accediendo a la propiedad `value` del `OPTION` que se encontraba seleccionado. Así accedemos a la opacidad deseada que queríamos aplicar.

Una vez tenemos esa opacidad deseada, recogida del `value` del `OPTION` seleccionado, podemos ver la siguiente línea de código, en la que hacemos el `fadeTo()`.

Veamos que `fadeTo()` recibe en principio dos métodos. El primero es la duración en milisegundos del ejemplo. El segundo es el valor de opacidad que queremos aplicar.

### Enviando funciones callback

Los tres métodos que estamos viendo para hacer fading, como cualquiera de los existentes en jQuery, permiten el envío de un parámetro como función callback. Con este código conseguimos que se ejecute un `fadeIn()` después de un `fadeOut()`, para conseguir un efecto de parpadeo, en el que primero se oculta el elemento y cuando desaparece se vuelve a mostrar restaurando su opacidad.

```
$("#milista").fadeOut(function(){
    $(this).fadeIn();
});
```

Como vemos, se está indicando una función callback y dentro de la misma, `this` es una referencia al objeto jQuery que recibió el anterior método. Osea, con `$("#milista").fadeOut()` se hace un efecto de fundido para que desaparezca el elemento `"#milista"`. Luego la función callback se ejecutará cuando ese elemento termine de desaparecer. Dentro de esa función callback se accede a `$(this)` para tener una referencia a `"#milista"` y sobre ese elemento invocamos al método `fadeIn()` para hacer que aparezca de nuevo la lista.

Ahora vamos a mostrar otro ejemplo de callback un poco más adelantado, en el que se encadenan varias funciones callback, que se ejecutarían una detrás de la otra.

```
var opacidad_deseada = $("#selopacidad").attr("value");

$("#e1").fadeTo(500, opacidad_deseada, function(){
    $("#e2").fadeTo(500, opacidad_deseada, function(){
        $("#e3").fadeTo(500, opacidad_deseada);
    });
});
```

En este código hacemos un efecto de `fadeTo()` sobre cada uno de los elementos de la lista. Para definir qué opacidad queremos aplicar a esos elementos utilizamos de nuevo el campo `SELECT` que habíamos visto anteriormente en este artículo. Pero en esta ocasión utilizamos una manera distinta de acceder al valor de opacidad que hay seleccionado, a través del método `attr()` de jQuery.

En el código anterior primero se ejecuta el cambio de opacidad en el primer elemento, luego en el segundo y por último en el tercero, siempre hacia la misma `"opacidad_deseada"` que se había recuperado en el `SELECT`.

### Código completo del ejemplo de fading en jQuery

A continuación podemos ver el código completo de trabajo con los métodos de fading disponibles en jQuery.

```

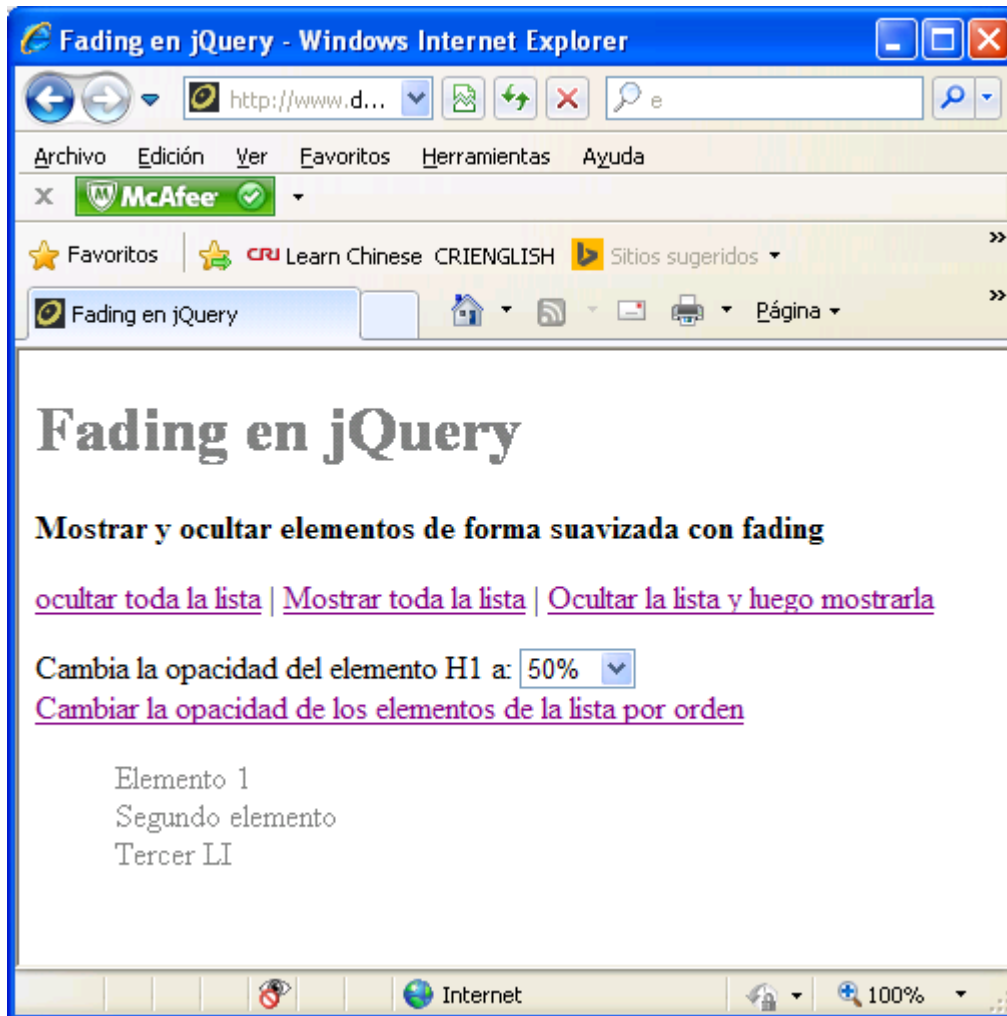
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="es">
<head>
<title>Fading en jQuery</title>
  <script src="../jquery-1.4.2.min.js"></script>
<script>
$(document).ready(function(){
  $("#ocultartoda").click(function(e){
    $("#milista").fadeOut();
  });
  $("#mostrartoda").click(function(e){
    $("#milista").fadeIn();
  });
  $("#ocultarmostrar").click(function(e){
    $("#milista").fadeOut(function(){
      $(this).fadeIn();
    });
  });
  $("#selopacidad").change(function(e){
    var opacidad_deseada = e.target.options[e.target.selectedIndex].value
    $("h1").fadeOut(1000,opacidad_deseada);
  });
  $("#pororden").click(function(e){
    var opacidad_deseada = $("#selopacidad").attr("value");
    $("#e1").fadeOut(500, opacidad_deseada, function(){
      $("#e2").fadeOut(500, opacidad_deseada, function(){
        $("#e3").fadeOut(500, opacidad_deseada);
      });
    });
  });
})
</script>
</head>
<body>
  <h1>Fading en jQuery</h1>
  <b>Mostrar y ocultar elementos de forma suavizada con fading</b>
  <p>
    <a href="#" id="ocultartoda">ocultar toda la lista</a> |
    <a href="#" id="mostrartoda">Mostrar toda la lista</a> |
    <a href="#" id="ocultarmostrar">Ocultar la lista y luego mostrarla</a>
  </p>
  <form name="f1">
    Cambia la opacidad del elemento H1 a: <select name="opacidad"
id="selopacidad">
      <option value="0.2">20%</option>
      <option value="0.5">50%</option>
      <option value="0.8">80%</option>
      <option value="1">100%</option>
    </select>
    <br>
    <a href="#" id="pororden">Cambiar la opacidad de los elementos de la lista por
orden</a>
  </form>

```

```
<ul id="milista">
  <li id="e1">Elemento 1</li>
  <li id="e2">Segundo elemento</li>
  <li id="e3">Tercer LI</li>
</ul>
```

```
</body>
</html>
```

A continuación, se muestra la pantalla de este ejercicio:



### 3.4.3 Efectos personalizados

Es posible realizar animaciones en propiedades CSS utilizando el método `$.fn.animate`. Dicho método permite realizar una animación estableciendo valores a propiedades CSS o cambiando sus valores actuales.

Efectos personalizados con `$.fn.animate`

```
$('#div.funtimes').animate(
{
  left : "+=50",
  opacity : 0.25
```

```
},  
300, // duration  
function() { console.log('realizado'); // función de devolución de llamada  
});
```

Las propiedades relacionadas al color no pueden ser animadas utilizando el método \$.fn.animate, pero es posible hacerlo a través de la extensión color plugin.

### Easing

El concepto de Easing describe la manera en que un efecto ocurre — es decir, si la velocidad durante la animación es constante o no. JQuery incluye solamente dos métodos de easing: swing y linear. Si desea transiciones más naturales en las animaciones, existen varias extensiones que lo permiten.

A partir de la versión 1.4 de la biblioteca, es posible establecer el tipo de transición por cada propiedad utilizando el método \$.fn.animate.

Transición de easing por cada propiedad

```
$('div.funtimes').animate(  
{  
  left : [ "+=50", "swing" ],  
  opacity : [ 0.25, "linear" ]  
},  
300  
);
```

### Animacion de colores

Con animate() no podemos hacer animaciones de color, es decir, hacer un gradiente suavizado para pasar de un color a otro con una animación. Quizás nunca encontrarás un inconveniente en esa carencia del framework, pero si algún día decides hacer una animación de color, tendrás que teclear bastante código para conseguirlo por cuenta propia.

Sin embargo, como en muchas otras ocasiones, los plugins de terceras personas nos pueden ahorrar mucho trabajo y horas de ingeniería. En este caso comentamos una de esos plugins que nos permitirá hacer animaciones de color directamente con el método animate().

El plugin de jQuery que vamos a mostraros a continuación se llama Color animation jQuery-plugin y lo puedes encontrar en la ruta: <http://www.bitstorm.org/jquery/color-animation/>

### Uso del plugin para animar colores

Lo cierto es que hay poco que explicar sobre este plugin, pues simplemente se trata de incluirlo en las páginas y a partir de ese momento simplemente utilizar el conocido método animate(), no obstante, hacemos una descripción paso por paso sobre cómo se utilizaría.

Las propiedades CSS que podrás animar usando este plugin son las siguientes:

- color
- backgroundColor
- borderColor

- borderBottomColor
- borderLeftColor
- borderRightColor
- borderTopColor
- outlineColor

### 1) Incluir jQuery y el plugin

Comenzamos por incluir los scripts del framework jQuery y del plugin para animación de colores.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"
type="text/javascript"></script>
<script src="jquery.animate-colors.js" type="text/javascript"></script>
```

### 2) Crear la animación

Ahora es tan sencillo como invocar a animate() con los parámetros necesarios, que fueron explicados en el artículo del método animate().

```
$("#h1").animate({
  color: "#f86"
}, 3000);
```

Con esto estamos haciendo que los elementos H1 de la página tengan una animación que durará 3 segundos en la que pasarán del color que tuvieran definido normalmente hasta el color #f86.

### Ejemplo de animación de un fondo con un patrón definido por imagen

Como hemos comprobado, no tiene mucho misterio, pero el efecto puede resultar interesante. Si tenemos un poco de creatividad todavía podemos conseguir efectos un poco más atractivos, como el que vamos a ver a continuación.

Se trata de hacer una animación de color de un fondo (atributo background-color), pero donde estamos utilizando un patrón de imagen que se repite en un mosaico. Al haber un fondo de imagen, da la sensación que la animación se realiza cambiando esa imagen, pero realmente solo está cambiando el color del fondo. Esto lo conseguimos gracias a una imagen de fondo que tiene transparencia.

Lo mejor para darse cuenta de lo que estamos haciendo es ver un ejemplo en marcha. Si hacemos clic en el titular veremos el efecto que estamos queriendo explicar.

Veamos el estilo que hemos definido para los elementos H2:

```
h2{
  padding: 30px;
  background-color: #ffc;
  background-image: url("fondo-h2.png");
  color: #009;
}
```

La imagen de fondo que hemos colocado "fondo-h2.png" es parcialmente transparente, para obtener el efecto deseado.

Ahora este pequeño código nos servirá para iluminar y oscurecer el fondo del H2 al hacer clic sobre él.



```
var iluminado = false;
$("h2").click(function(){
    var elem = $(this);
    if(iluminado){
        elem.animate({
            "background-color": "#ffc"
        }, 500);
    }else{
        elem.animate({
            "background-color": "#9f9"
        }, 500);
    }
    iluminado = !iluminado;
})
```

Como se puede comprobar, se ha utilizado una variable "iluminado" para saber cuando el elemento está encendido y cuando apagado. Luego creamos un evento click, para colocar la funcionalidad descrita. Si estaba iluminado, hago una animación del atributo background-color hacia un color y si estaba oscurecido paso el background-color hacia otro color.

El efecto no es nada del otro mundo, pero es bastante versátil y si tenéis un bonito fondo con un patrón interesante, más atractivo será el resultado.

#### 3.4.4 Control de efectos

jQuery provee varias herramientas para el manejo de animaciones.

`$.fn.stop`

Detiene las animaciones que se están ejecutando en el elemento seleccionado.

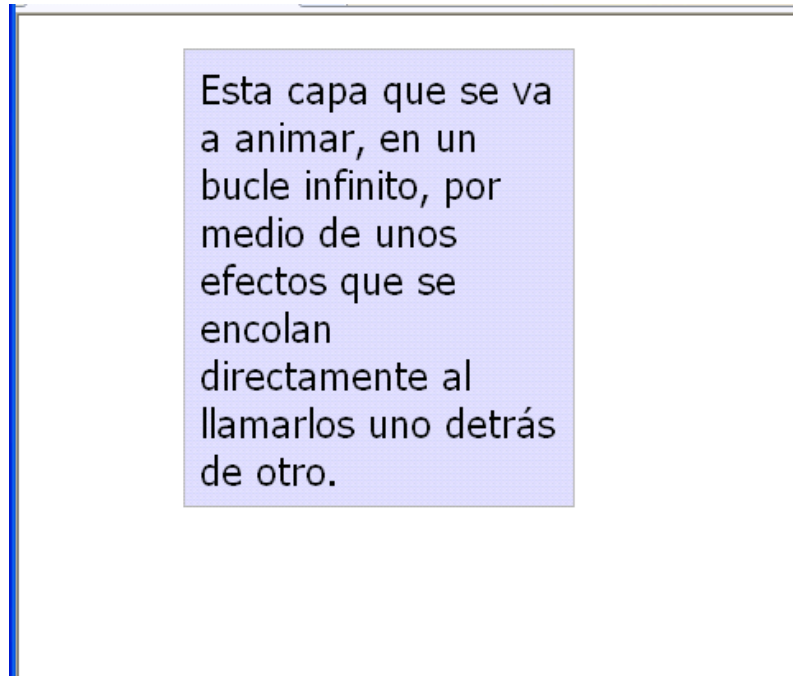
`$.fn.delay`

Espera un tiempo determinado antes de ejecutar la próxima animación.

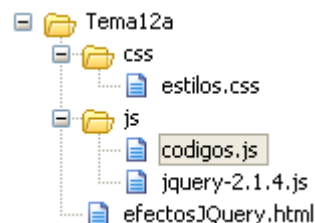
```
$('#h1').show(300).delay(1000).hide(300);
```

## Laboratorio 12.1

En este laboratorio vamos a aplicar efectos con JQuery para realizar una animación de tal forma que una capa div tenga diferentes animaciones de manera infinita en la página.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Copie el archivo jquery-2.1.4.js en su carpeta js
2. Cree un archivo css e inserte el siguiente código:

```
#micapa{
  left: 200px;
  top: 20px;
  position: absolute;
  font-size: 0.75em;
  font-family: tahoma, verdana, sans-serif;
  width: 420px;
  height: 220px;
  background-color: #ddf;
  padding: 10px;
  border: 1px solid #bbb;
}
```

3. Guarde su archivo en la carpeta css con el nombre estilos.css.
4. Cree una archivo js en la carpeta js e inserte el siguiente código:

```
function colaEfectos(){
    capa = $("#micapa");
    capa.animate({
        "font-size": "1.5em"
    }, 2000);
    capa.hide(1000);
    capa.show(1000);
    capa.animate({
        "left": "350px",
        "top": "50px"
    }, 1500);
    capa.animate({
        "font-size": "0.75em"
    }, 2000);
    capa.animate({
        "left": "100px",
        "top": "20px"
    }, 1500, colaEfectos);
}
$(document).ready(function(){
    colaEfectos();
});
```

5. Guarde su archivo en la carpeta js con el nombre codigos.js.
6. Cree una archivo html e inserte el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title></title>
    <script type='text/javascript' src='js/jquery-2.1.4.js'></script>
    <script src="js/codigos.js" type="text/javascript"></script>
    <link href="css/estilos.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <div id="micapa">
      Esta capa que se va a animar, en un bucle infinito, por medio
      de unos efectos que se encolan directamente al llamarlos uno
      detrás de otro.
    </div>

  </body>
</html>
```

7. Grabe su archivo con el nombre efectosJQuery.html.

# Resumen

1. Las funciones de efectos son los métodos jQuery que realizan un cambio en los elementos de la página de manera suavizada, es decir, que alteran las propiedades de uno o varios elementos progresivamente, en una animación a lo largo de un tiempo.
2. Para utilizarlos los efecto, debemos indicar dos cosas, el elemento sobre el cual se va a aplicar el efecto y el enlace que ejecutará la función. La función sería algo así:

```
<script type='text/javascript'>
    $(document).ready(function(){
        $("#nombreEnlaceMostrar").click(function () { $("#nombreContenido").show(); });
        $("#nombreEnlaceOcultar").click(function () { $("#nombreContenido").hide(); });
    });
</script>
```

3. Es posible realizar animaciones en propiedades CSS utilizando el método \$.fn.animate. Dicho método permite realizar una animación estableciendo valores a propiedades CSS o cambiando sus valores actuales.
4. Con animate() no podemos hacer animaciones de color, es decir, hacer un gradiente suavizado para pasar de un color a otro con una animación. Sin embargo, como en muchas otras ocasiones, los plugins de terceras personas nos pueden ahorrar mucho trabajo y horas de ingeniería.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.taringa.net/post/hazlo-tu-mismo/17453454/Mejores-tutoriales-de-JQuery---2013.html>
- [http://mundosica.github.io/tutorial\\_hispano\\_jQuery/sesion04/index.html](http://mundosica.github.io/tutorial_hispano_jQuery/sesion04/index.html)
- <http://codehero.co/jquery-desde-cero-animaciones-y-efectos/>
- <http://www.batanga.com/tech/13342/conociendo-los-efectos-de-jquery>
- <http://vagabundia.blogspot.com/2010/01/efectos-elementales-con-jquery.html>
- <http://api.jquery.com/animate/>



# JSON

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript y usando la notación JSON, JQuery, crea páginas con contenido dinámico

## TEMARIO

### 5.1 Tema 13 : Introducción

- 5.1.1 Definición
- 5.1.2 Estructura de un archivo JSON
- 5.1.3 Creación de un archivo JSON

## ACTIVIDADES PROPUESTAS

- Los alumnos identifican la estructura de un archivo JSON.
- Los alumnos crean un archivo JSON.

## 5.1 INTRODUCCIÓN

### 5.1.1 Definición

Actualmente las aplicaciones Web son cada vez más interactivas centradas en los datos, y existen nuevas técnicas para hacer estas aplicaciones de forma más eficiente. Un gran avance en esta área fue la llegada de AJAX (Asynchronous JavaScript y XML). Desde entonces, los desarrolladores han conseguido nuevas herramientas para exprimir aún más el rendimiento y la eficiencia de cada byte. Una forma de lograrlo ha sido el uso de JSON.

Primero vamos a entender lo que es JSON y lo que no lo es. El estándar formal internacional para JSON es RFC 4627. Douglas Crockford, el autor de la JSON (JavaScript Object Notation) describe JSON como “un intercambio de datos en formato ligero” y se basa en dos ideas clave: El uso de pares de nombre / valor y de una lista ordenada de valores. Dado que estas características existen en casi cualquier lenguaje de programación importante (lo que conocemos como arreglos), esto hace que JSON es una buena solución para muchas plataformas de desarrollo. JSON es a menudo descrito como un subconjunto de JavaScript, pero JSON en sí no es un lenguaje. Tampoco es un formato de documento tan completo como un XML. JSON es una forma de almacenar información en forma organizada, de fácil acceso así. Es a la vez humana y legible por las computadoras y se analiza con facilidad. Lo más importante es JSON es un proceso abierto, basado en texto formato de intercambio de datos que proporciona simplicidad e independencia del lenguaje de programación.

Al momento de considerar las nuevas tecnologías para sus aplicaciones, es posible que desee saber si la tecnología está siendo utilizada por otras personas. ¿Quién está utilizando JSON actualmente? En primer lugar el creador de esta tecnología Douglas Crockford es el arquitecto senior de Yahoo JavaScript. Twitter ha cambiado recientemente el uso de XML a JSON para su API. The Google Web Toolkit también trabaja con JSON. La norma oficial para JSON se encuentra bien documentada y actualmente esta creciendo rápidamente su uso en toda la industria, lo cual da mucha seguridad al momento de considerarla como una tecnología útil para nuevos desarrollo

Por supuesto, nadie debe utilizar una tecnología sólo porque este de moda. ¿Por qué debe usted utilizar JSON? La mayoría de las definiciones de JSON enfatizan que es “ligera”. ¿Qué significa eso? En comparación con XML, el porcentaje de contenido / volumen con JSON es mucho menor. Esto significa que es más pequeño y menos complejo de generar o recuperar y fácil de leer. Eso se traduce en tiempos de carga más rápidos para las páginas web y la posibilidad de enviar más datos, más rápidamente que con otros formatos de datos

### 5.1.2 Estructura de un archivo JSON

JSON es un subconjunto de JavaScript. Probablemente se esté utilizando ya Javascript – ya que es el mundo del lenguaje de programación más utilizado – y usted no tendrá que aprender otro idioma o formato estándar. Los datos codificados en JSON se integran muy fácilmente como variables Javascript en estructuras o arreglos

La mejor manera de describirlo es quizá mediante un ejemplo:

```
{
  "Nombre" : "Juan",
  "Edad": 28,
  "Aficiones": ["Música", "Cine", "Tenis"],
  "Residencia": "Madrid"
}
```

Como vemos, en JSON se pueden representar dos tipos de estructuras:

Un conjunto de pares (clave,valor) encerrado entre los caracteres “{” y “}”, separando la clave del valor por el símbolo “:”, y separando cada par del siguiente con el carácter “,”.

Un conjunto ordenado de valores encerrado entre los caracteres “[” y “]”, y separando cada valor del siguiente con el carácter “,”.

Por otra parte, el valor de un par (clave, valor) puede ser un elemento simple (cadena de caracteres o número) o bien una estructura de datos. Esto permite representar estructuras de una complejidad arbitraria.

Por ejemplo, la estructura del ejemplo de arriba podría ser parte de un documento JSON más complejo, como valor asociado a una clave en una estructura de nivel superior:

```
{
  "responsable":
  {
    "Nombre" : "Juan",
    "Edad": 28,
    "Aficiones": ["Música", "Cine", "Tenis"],
    "Residencia": "Madrid"
  },
  "empleados":
  [
    {
      "Nombre" : "Elena",
      "Edad": 26,
      "Aficiones": ["Música", "Cine"],
      "Residencia": "Madrid"
    },
    {
      "Nombre" : "Luis",
      "Edad": 31,
      "Aficiones": ["Teatro", "Cine", "Fútbol"],
      "Residencia": "Madrid"
    }
  ]
}
```

En este caso, la estructura principal es un conjunto de pares (clave, valor) con claves “responsable” y “empleados”.

El valor de la clave “responsable” es un conjunto de pares (clave, valor), mientras que el valor de la clave “empleados” es un array que contiene dos elementos, cada uno de los cuales, a su vez, es una estructura de pares (clave, valor).

Aquí puede ver otro ejemplo de codificación simple:

```
var contacto = {
    "nombre": "Juan",
    "apellidos": "Perez Perez",
    "edad": 30
};
```

### 5.1.3 Creación de un archivo JSON

Vamos a crear un archivo json a partir de un ejemplo muy básico de una cadena JSON:

```
[
  {"Titulo": "El señor de los anillos", "Autor": "J.R.R. Tolkien"},
  {"Titulo": "Cancion de hielo y fuego", "Autor": "George RR Martin"},
  {"Titulo": "Los Pilares de la Tierra", "Autor": "Ken Follett"}
]
```

Como podemos ver tenemos una colección de tres libros y cada uno de ellos contiene su título y el nombre de su autor. La forma de trabajar con estos datos desde JavaScript sería la siguiente:

```
var cadenaLibros = '[{"Titulo": "El señor de los anillos", "Autor": "J.R.R. Tolkien"},
{"Titulo": "Cancion de hielo y fuego", "Autor": "George RR Martin"},
{"Titulo": "Los Pilares de la Tierra", "Autor": "Ken Follett"}]';
```

```
var libros = JSON.parse(cadenaLibros);
```

```
for(var i = 0; i < libros.length; i++)
    alert('El libro: ' + libros[i].Titulo + ' es del autor: ' + libros[i].Autor);
```

Con la función `JSON.parse()` convertimos la cadena en un array de objetos. Después con el bucle recorremos el array y mostramos el título y el autor de cada libro accediendo a ellos como propiedades. Vamos ahora a complicar un poco más el asunto. Supongamos que tenemos la siguiente cadena JSON:

```
[{"Titulo": "El señor de los anillos", "Autor": "J.R.R. Tolkien",
  "Partes": [{"Tomo": 1, "Titulo": "La comunidad del anillo"},
              {"Tomo": 2, "Titulo": "Las dos torres"},
              {"Tomo": 3, "Titulo": "El retorno del rey"}]},
 {"Titulo": "Cancion de hielo y fuego", "Autor": "George RR Martin",
  "Partes": [{"Tomo": 1, "Titulo": "Juego de tronos"},
              {"Tomo": 2, "Titulo": "Choque de reyes"},
              {"Tomo": 3, "Titulo": "Tormenta de espadas"},
              {"Tomo": 4, "Titulo": "Festín de cuervos"}]},
 {"Titulo": "Los Pilares de la Tierra", "Autor": "Ken Follett"}]
```

Ahora vemos que algunos libros contienen "partes" dentro de ellos, separando los distintos tomos con los que cuenta cada uno. Lo que vamos a realizar ahora es mostrar estos datos creando una lista con la ayuda de JQuery.

```
var cadena = '[{ ... }]'; // Usamos la cadena anterior, no la copio para ahorrar espacio
```



```
var libros = JSON.parse(cadena);

var ul = $('<ul></ul>'); // Creamos un elemento ul

for( var i = 0; i < libros.length; i++ ) {
    var li = $('<li></li>') // Creamos un elemento li
    // Añadimos el titulo y el autor al elemento li
    li.append(libros[i].Titulo + ' (' + libros[i].Autor + ')');

    // Comprobamos si el libro tiene partes
    if( libros[i].Partes != undefined && libros[i].Partes.length > 0 ) {
        var ulInterno = $('<ul></ul>'); // Creamos otro elemento ul

        for( var j = 0; j < libros[i].Partes.length; j++ ) {
            // Por cada parte crearemos un elemento li que añadiremos al ul que acabamos de
            // crear
            ulInterno.append('<li>' + libros[i].Partes[j].Tomo + ' - ' + libros[i].Partes[j].Titulo +
            '</li>');
        }
        li.append(ulInterno); // Añadimos el último elemento ul al li creado primero
    }
    ul.append(li); // Añadimos el li inicial al ul inicial.
}
$('body').append(ul); // Añadimos la lista al body de la pagina
```

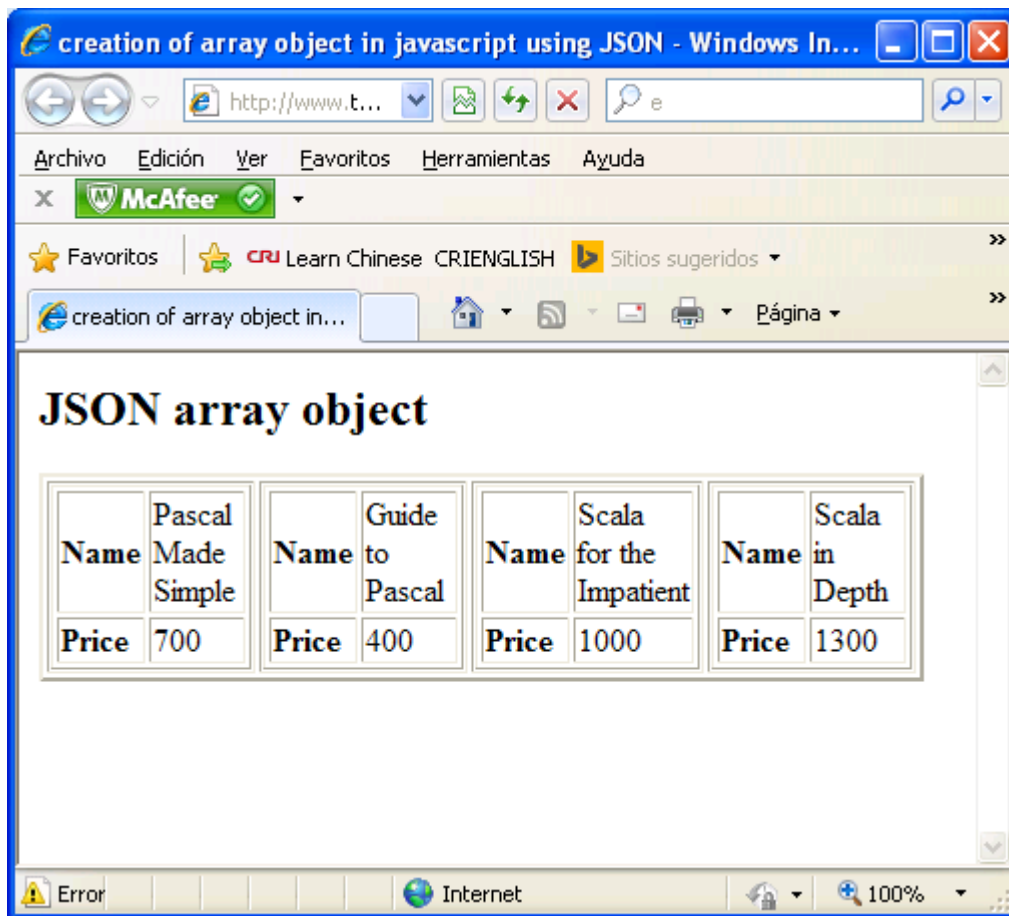
Con estos sencillos pasos hemos conseguido crearnos una lista con todos los libros de nuestra colección. El código creo que es bastante sencillo y con los comentarios no requiere ninguna explicación más.

A parte de la función `JSON.parse()` que es la fundamental para realizar la transformación entre la cadena JSON y un array de objetos hay otra función que no se ha usado, pero que cabe mencionar. Se trata de la función `JSON.stringify()` esta función realiza la tarea inversa a la anterior. Es decir convierte un objeto en una cadena. Un ejemplo se puede ver si ejecutamos lo siguiente:

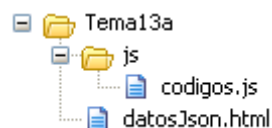
```
var nuevaCadena = JSON.stringify(libros);
alert(nuevaCadena);
```

## Laboratorio 13.1

En este laboratorio vamos a aplicar la creación de un código JSON para almacenar datos y luego imprimirlo en una página web con códigos javascript.



En este ejercicio, va a crear la siguiente estructura de carpetas y almacenar los archivos en las carpetas correspondientes.



1. Cree un archivo en la carpeta js e inserte el siguiente código:

```

document.writeln("<h2>JSON array object</h2>");

var books = { "Pascal" : [
    { "Name" : "Pascal Made Simple", "price" : 700 },
    { "Name" : "Guide to Pascal", "price" : 400 }
],
  "Scala" : [
    { "Name" : "Scala for the Impatient", "price" : 1000 },
    { "Name" : "Scala in Depth", "price" : 1300 }
  ]
}
  
```

```

}

var i = 0
document.writeln("<table border='2'><tr>");
for(i=0;i<books.Pascal.length;i++)
{
    document.writeln("<td>");
    document.writeln("<table border='1' width=100 >");
    document.writeln("<tr><td><b>Name</b></td><td width=50>"
    + books.Pascal[i].Name+"</td></tr>");
    document.writeln("<tr><td><b>Price</b></td><td width=50>"
    + books.Pascal[i].price +"</td></tr>");
    document.writeln("</table>");
    document.writeln("</td>");
}

for(i=0;i<books.Scala.length;i++)
{
    document.writeln("<td>");
    document.writeln("<table border='1' width=100 >");
    document.writeln("<tr><td><b>Name</b></td><td width=50>"
    + books.Scala[i].Name+"</td></tr>");
    document.writeln("<tr><td><b>Price</b></td><td width=50>"
    + books.Scala[i].price+"</td></tr>");
    document.writeln("</table>");
    document.writeln("</td>");
}
document.writeln("</tr></table>");

```

2. Guarde su archivo en la carpeta js con el nombre codigos.js.
3. Cree una archivo html e inserte el siguiente código:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title></title>
    <script src="js/codigos.js"></script>

  </head>
  <body>

  </body>
</html>

```

4. Grabe su archivo html con el nombres datosJson.html y luego ejecute su pagina.

# Resumen

1. Se describe JSON como “un intercambio de datos en formato ligero” y se basa en dos ideas clave: El uso de pares de nombre / valor y de una lista ordenada de valores.
2. Dado que estas características existen en casi cualquier lenguaje de programación importante (lo que conocemos como arreglos), esto hace que JSON sea una buena solución para muchas plataformas de desarrollo..
3. Con JSON se pueden representar dos tipos de estructuras:

Un conjunto de pares (clave,valor) encerrado entre los caracteres “{” y “}”, separando la clave del valor por el símbolo “:”, y separando cada par del siguiente con el carácter “,”

Un conjunto ordenado de valores encerrado entre los caracteres “[” y “]”, y separando cada valor del siguiente con el carácter “,”.

Se puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://es.wikipedia.org/wiki/JSON>
- <https://geekytheory.com/json-i-que-es-y-para-que-sirve-json/>
- <http://blog.openalfa.com/introduccion-al-formato-json>
- <http://www.adictosaltrabajo.com/tutoriales/prototypejs-ajax-json/>