



JavaScript

Sesión 3 - JavaScript y DOM



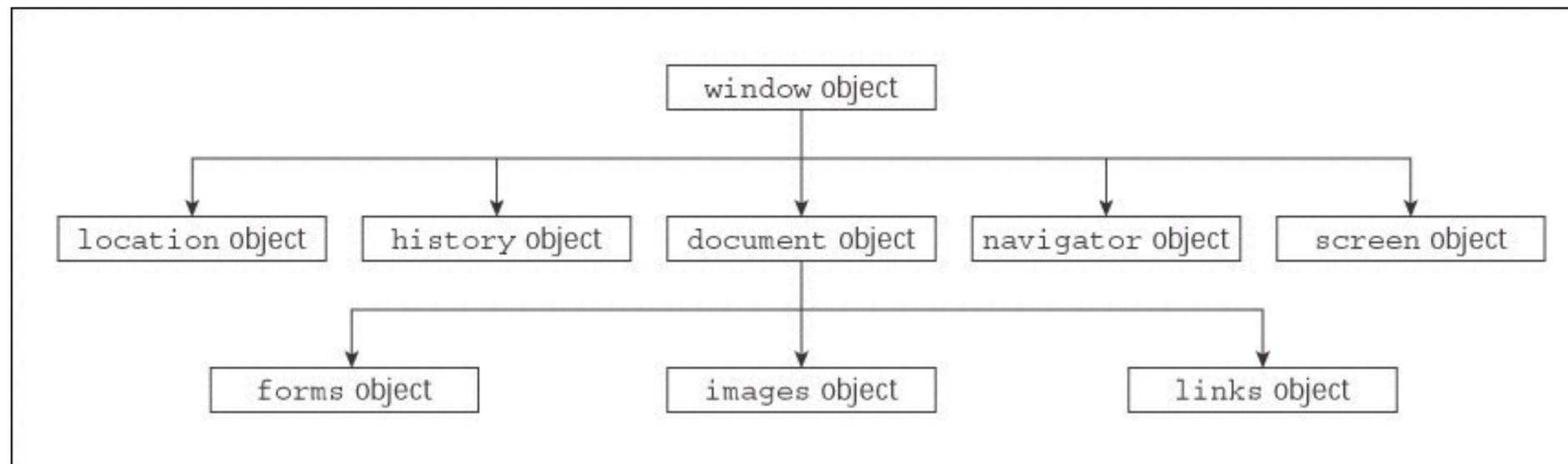
Índice

- BOM
- Trabajando con el DOM
 - Tipos de nodos, enlaces entre nodos
- Trabajando con CSS
 - Estilos, clases
- Animaciones
- Eventos
 - Flujo de eventos, tipos (carga, foco, ratón, teclado)
- Trabajando con Formularios



3.1 BOM

- *Browser Object Model*
- Define una serie de objetos que nos permiten interactuar con el navegador





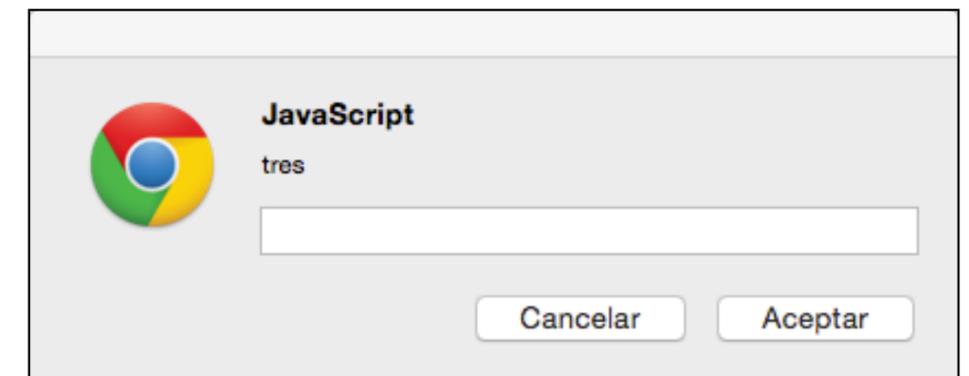
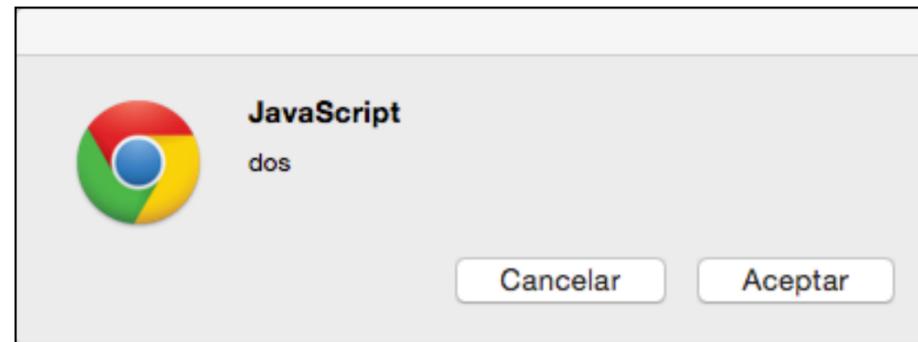
window

- Objeto global
- `window.open()` → abre una nueva ventana del navegador
 - En los navegadores actuales, si intentamos abrir más de una ventana mediante `window.open` el propio navegador bloqueará su apertura (p.ej. ventanas emergentes de publicidad)
 - Devuelve un nuevo objeto `window` → será el objeto global para el script que corre sobre dicha ventana
Conteniendo todas las propiedades comunes a los objetos (p.ej. el constructor `Object`, objeto `Math`)
La mayoría de los navegadores no permiten mirar sus propiedades → modo *sandbox*.
Modo *sandbox* → implica que el navegador sólo nos mostrará la información relativa al mismo dominio, y si abrimos una página de un dominio diferente al nuestro no tendremos control sobre las propiedades privadas del objeto `window`.
- `window.close()` → cierra una venta
- `window.alert(mensaje)`, `window.confirm(mensaje)` y `window.prompt(mensaje [, valorPorDefecto])`. → muestra mensajes de alerta, confirmación y consulta



Ejemplos de mensajes

```
alert("uno");  
confirm("dos");  
var resp = prompt("tres");
```





navigator

- Permite acceder a propiedades de información del navegador, tales como su nombre y versión.

```
console.log(navigator.language); // "es-es"
console.log(navigator.cookieEnabled); // true
console.log(navigator.appName); // "Netscape"
console.log(navigator.appVersion); // "5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/600.2.5 (KHTML, like Gecko) Version/8.0.2 Safari/600.2.5"
console.log(navigator.product); // "Gecko"
console.log(navigator.userAgent); // "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/600.2.5 (KHTML, like Gecko) Version/8.0.2 Safari/600.2.5"
```

<http://jsbin.com/luhaye/1/edit?js>



document, document.location

- Objeto que representa el documento mostrado
- Propiedad `location` → información sobre la URL
 - `href`: cadena que representa la URL completa
 - `protocol`: protocolo de la URL
 - `host`: nombre del host
 - `pathname`: trayectoria del recurso
 - `search`: parte que contiene los parámetros, incluido el símbolo ?

<http://localhost:63342/Pruebas/bom/location.html?alfa=beta&gama=delta>

```
console.log("href:" + location.href); // http://localhost:63342/Pruebas/bom/location.html?
alfa=beta&gama=delta
console.log("protocol:" + location.protocol); // http:
console.log("host:" + location.host); // localhost:63342
console.log("pathname:" + location.pathname); // /Pruebas/bom/location.html
console.log("search:" + location.search); // ?alfa=beta&gama=delta
```

- Si a `location.href` le asignamos una nueva URL, el navegador realizará una petición a dicha URL y el navegador cargará el nuevo documento.



document.write

- `document.write(texto)`
- Permite escribir contenido HTML en el documento.

```
<html>
<head><title>La hora</title></head>
<body>
  <p>Son las
    <script type="text/javascript">
      var time = new Date();
      document.write(time.getHours() + ":" + time.getMinutes());
    </script>
  </p>
</body>
</html>
```



3.2 Trabajando con el DOM

- DOM: *Document Object Model* - Modelo de objetos de documento
- El API DOM permite interactuar con el documento HTML y cambiar el contenido y la estructura del mismo, los estilos CSS y gestionar los eventos mediante listeners.
- El *DOM* es un modelo que representa en forma de árbol un documento HTML, formado por nodos.
 - *DOM Level 0* (Legacy DOM): define las colecciones forms, links e images.
 - *DOM Level 1* (1998): introduce el objeto `Node` y a partir de él, los nodos `Document`, `Element`, `Attr` y `Text`. Además, las operaciones `getElementsByTagName`, `getAttribute`, `removeAttribute` y `setAttribute`
 - *DOM Level 2*: facilita el trabajo con XHTML y añade los métodos `getElementById`, `hasAttributes` y `hasAttribute`
 - *DOM Level 3*: añade atributos al modelo, entre ellos `textContent` y el método `isEqualNode`.
 - *DOM Level 4* (2004): supone el abandono de HTML por XML e introduce los métodos `getElementsByClassName`, `prepend`, `append`, `before`, `after`, `replace` y `remove`.



window

- El nodo raíz y padre de todos los nodos es `window`.
- En *JavaScript*, al declarar una variable tiene una alcance global, y todas las variable globales forman parte del objeto `window`.

```
var batman = "Bruce Wayne";  
console.log(window.batman);
```

- Normalmente no referenciamos al objeto `window` directamente.
- Sólo cuando desde una función queremos acceder a una variable global de manera unívoca.

```
var superheroe = "Batman";  
var mejorSuperheroe = function () {  
    var superheroe = "Superman";  
  
    if (window.superheroe !== superheroe) {  
        superheroe = window.superheroe;  
    }  
}
```



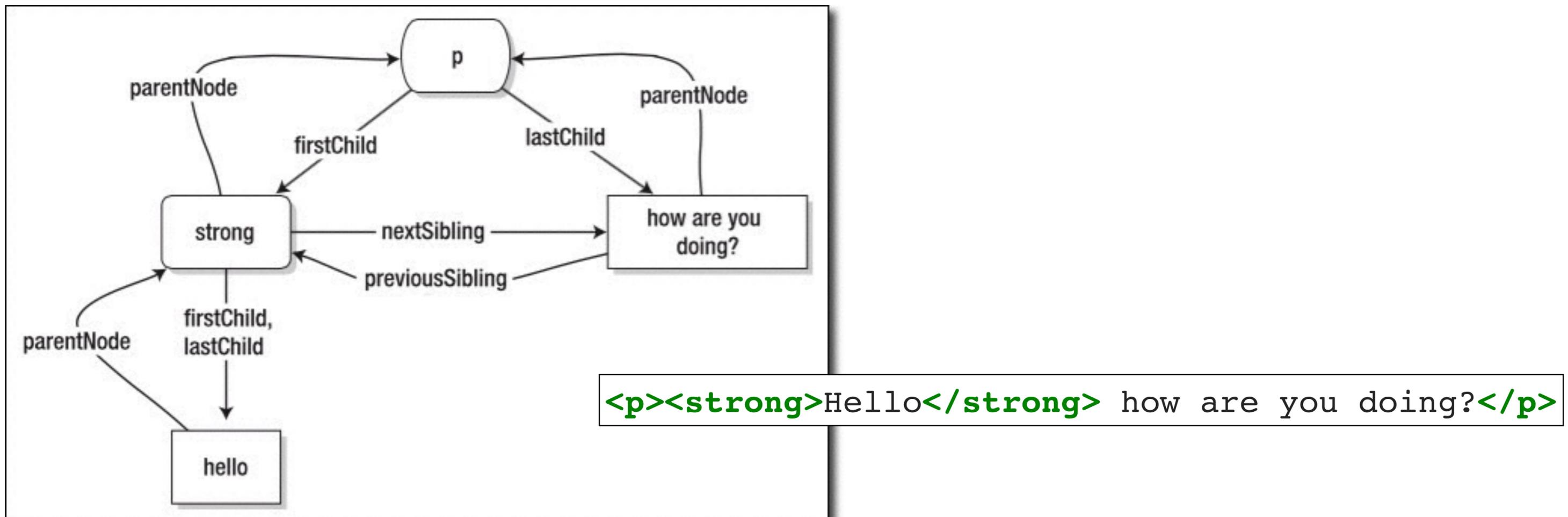
document

- Objeto que da acceso al DOM de una página web
- A partir de `document` podemos acceder a los elementos que forman la página mediante una estructura jerárquica.
- Al objeto que hace de raíz del árbol, el nodo `html`, se puede acceder mediante la propiedad **`document.documentElement`**.
- Si en vez de a `html` necesitamos acceder al elemento `body` → **`document.body`**



Elementos del DOM

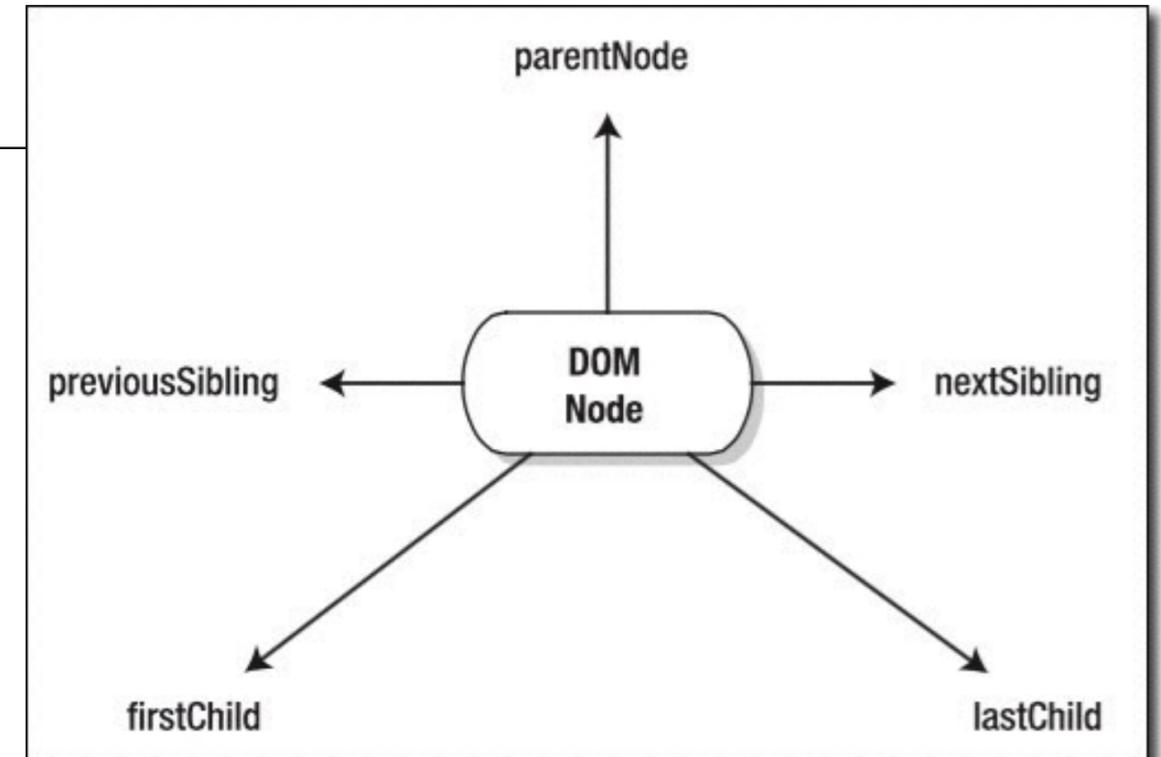
- Cada elemento exceptuando el elemento `<html>` forma parte de otro elemento, que se le conoce como padre (*parent*).
- Un elemento a su vez puede contener elementos hijos (*child*) y/o hermanos (*siblings*)





Enlaces entre Nodos

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo DOM</title>
<meta charset="utf-8" />
</head>
<body>
<h1>Encabezado uno</h1>
<p>Primer párrafo</p>
<p>Segundo párrafo</p>
<div><p id="tres">Tercer párrafo dentro de un div</p></div>
<script src="dom.js" charset="utf-8"></script>
</body>
</html>
```



<http://jsbin.com/bopije/1/edit?html>

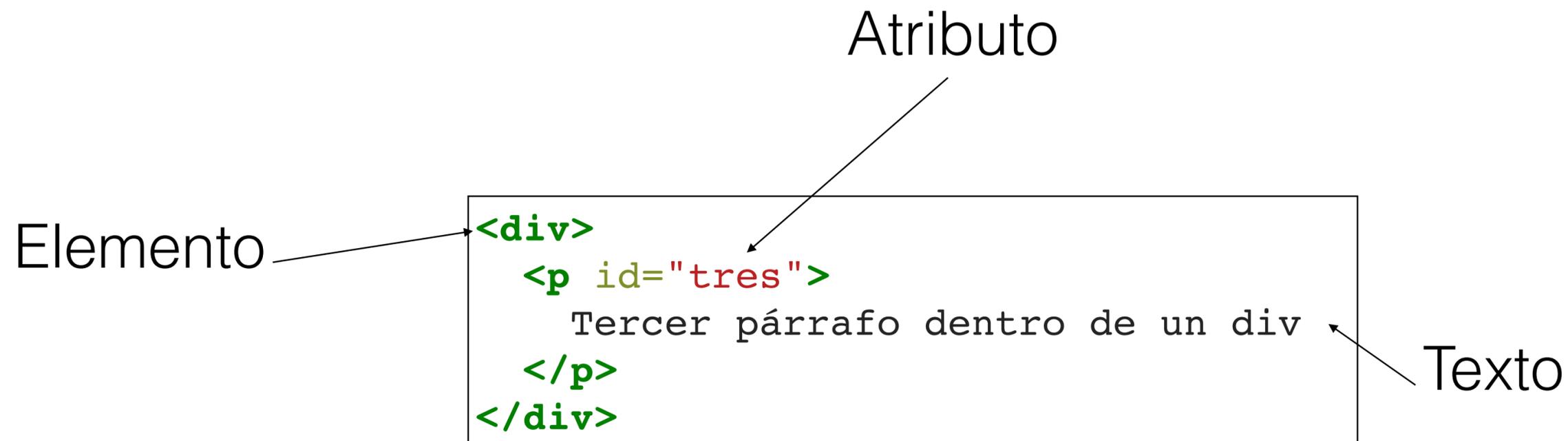
```
var encabezado = document.body.firstChild;
var scriptJS = document.body.lastChild;
var parrafol = encabezado.nextSibling;
var capa = scriptJS.previousSibling;
```

<http://jsbin.com/bopije/1/edit?js>



Tipos de Nodos

- **Element:** nodo que contiene una etiqueta HTML
- **Attr:** nodo que forma parte de un elemento HTML
- **Text:** nodo que contiene texto y que no puede tener hijos





Trabajando con el Tipo de un Nodo

- Para averiguar si un nodo representa un texto o un elemento → propiedad `nodeType`
 - Devuelve un número, por ejemplo: 1 si es un elemento (nodo HTML), 3 si es de texto, 8 comentario.

```
function esNodoTexto(nodo) {  
    return nodo.nodeType == document.TEXT_NODE;  
}  
esNodoTexto(document.body); // false  
esNodoTexto(document.body.firstChild.firstChild); // true
```

- Los elementos contienen la propiedad `nodeName` que indica el tipo de etiqueta HTML que representa (siempre en mayúsculas).
- Los nodos de texto contienen `nodeValue` que obtiene el texto contenido.

```
document.body.firstChild.nodeName; // H1  
document.body.firstChild.firstChild.nodeValue; // Encabezado uno
```



Recorriendo el DOM

- Proceso recursivo

```
function buscarTexto(nodo, cadena) {  
  if (nodo.nodeType == document.ELEMENT_NODE) {  
    for (var i=0, len=nodo.childNodes.length; i<len; i++) {  
      if (buscarTexto(nodo.childNodes[i], cadena)) {  
        return true;  
      }  
    }  
    return false;  
  } else if (nodo.nodeType == document.TEXT_NODE) {  
    return nodo.nodeValue.indexOf(cadena) > -1;  
  }  
}
```

<http://jsbin.com/mufin/9/edit?js>



Seleccionando Elementos

- Grupal: `document.getElementsByTagName(nombreDeTag)` → devuelve un array con los nodos cuya etiqueta sea *nombreDeTag*
- Individual: `document.getElementById(nombreDeId)` → devuelve un nodo cuyo id coincida con *nombreDeId*

```
(function() {  
  var pElements = document.getElementsByTagName("p"); // NodeList  
  console.log(pElements.length); // 3  
  console.log(pElements[0]); // Primer párrafo  
  
  var divpElement = document.getElementById("tres");  
  console.log(divpElement);  
})();
```



querySelector

- Selector API (2013)

- Soportado por todos los navegadores actuales (soporte parcial en IE8)
- Utilizan un selector CSS
- Ofrece mucha flexibilidad
- Empleado por jQuery

```
var pElements = document.querySelectorAll("p");  
var divpElement = document.querySelector("div p");  
var tresElement = document.querySelector("#tres");
```

- **querySelector**(selector) → devuelve el 1^{er} elemento que cumple el selector

- **querySelectorAll**(selector) → devuelve una lista estática con todos los elementos que cumplen el selector

- **getElementById** es **5 veces más rápido** que **querySelector**



Añadiendo Contenido

- Para añadir un párrafo al documento que teníamos previamente, primero tenemos que crear el contenido y luego decidir donde colocarlo.
 - Usar el método `createElement` para crear el elemento
 - Decidir donde colocarlo y añadir el contenido (por ejemplo, mediante `appendChild`).

```
(function() {  
  var elem = document.createElement("p"),  
      texto = "<strong>Nuevo párrafo creado dinámicamente</strong>",  
      contenido = document.createTextNode(texto);  
  
  elem.appendChild(contenido);  
  elem.id = "conAppendChild";  
  document.body.appendChild(elem);  
  // lo añade como el último nodo detrás de script  
})();
```

Encabezado uno

Primer párrafo

Segundo párrafo

Tercer párrafo dentro de un div

Nuevo párrafo creado dinámicamente

```
Q Elements Network Sources Timeline Profiles Resources Audits Console  
<!DOCTYPE html>  
▼ <html lang="es">  
  ▶ <head>...</head>  
  ▼ <body>  
    <h1>Encabezado uno</h1>  
    <p>Primer párrafo</p>  
    <p>Segundo párrafo</p>  
    ▼ <div>  
      <p id="tres">Tercer párrafo dentro de un div</p>  
    </div>  
    <script src="domCreateElement.js" charset="utf-8"></script>  
    <p id="conAppendChild"><strong>Nuevo párrafo creado dinámicamente</strong></p>  
  </body>  
</html>
```



Operaciones de inserción

- **appendChild** (`nuevoElemento`) → el nuevo nodo se incluye inmediatamente después de los hijos ya existentes (si hay alguno) y el nodo padre cuenta con una nueva rama.
- **insertBefore** (`nuevoElemento`, `elementoExistente`) → permiten elegir un nodo existente del documento e incluir otro antes que él.
- **replaceChild** (`nuevoElemento`, `elementoExistente`) → reemplazar un nodo por otro
- **removeChild** (`nodoABorrar`) → elimina un nodo
- **cloneNode** () → permite clonar un nodo, permitiendo tanto el elemento como el elemento con su contenido (parámetro a `true`)
- Propiedad **innerHTML** → permite añadir el contenido de un elemento. Parsea el contenido incluido



Ejemplos de Inserción de Contenido

```
(function() {  
    var doc = document,  
        elem = doc.createElement("p"),  
        contenido = doc.createTextNode("<strong>Nuevo párrafo creado dinámicamente</strong>"),  
        pTres = doc.getElementById("tres");  
  
    elem.appendChild(contenido);  
    elem.id = "conAppendChild";  
  
    pTres.parentNode.appendChild(elem); // o insertBefore, replaceChild  
})();
```

```
(function() {  
    var  
        doc = document,  
        elem = doc.createElement("p"),  
        pTres = doc.getElementById("tres");  
  
    elem.innerHTML = "<strong>Nuevo párrafo reemplazado dinámicamente</strong>";  
    elem.id = "conInner";  
  
    pTres.parentNode.replaceChild(elem, pTres);  
})();
```

Encabezado uno

Primer párrafo

Segundo párrafo

Nuevo párrafo reemplazado dinámicamente

```
Q Elements Network Sources Timeline Profiles Resources Audits Console  
<!DOCTYPE html>  
<html lang="es">  
  <head>...</head>  
  <body>  
    <h1>Encabezado uno</h1>  
    <p>Primer párrafo</p>  
    <p>Segundo párrafo</p>  
    <div>  
      <p id="conInner">  
        <strong>Nuevo párrafo reemplazado dinámicamente</strong>  
      </p>  
    </div>  
    <script src="doReplaceElement.js" charset="utf-8"></script>  
  </body>  
</html>
```



getElementById vs querySelector

- Las referencias con `getElements*` están vivas y siempre contienen el estado actual del documento,
- Mediante `querySelector*` obtenemos las referencias existentes en el momento de ejecución, sin que cambios posteriores en el DOM afecten a las referencias obtenidas.

```
(function() {  
    var getElements = document.getElementsByTagName("p"),  
        queryElements = document.querySelectorAll("p");  
    console.log("Antes con getElements:" + getElements.length); // 3  
    console.log("Antes con querySelector:" + queryElements.length); // 3  
  
    var elem = document.createElement("p");  
    elem.innerHTML = "getElements vs querySelector";  
    document.body.appendChild(elem);  
  
    console.log("Después con getElements:" + getElements.length) // 4  
    console.log("Después con querySelector:" + queryElements.length) // 3  
})();
```



Gestionando Atributos

- Operaciones

- `getAttribute` (nombreAtributo)
- `setAttribute` (nombreAtributo, valorAtributo)

```
var pTres = document.getElementById("tres");  
pTres.setAttribute("align", "right");
```

- Propiedad

- `elemento.atributo`

```
var pTres = document.getElementById("tres");  
pTres.align = "right";
```

- El uso de atributos para definir la apariencia del documento está desaconsejado → CSS



3.3 Trabajando con CSS

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8" />
  <title></title>
  <style>
    #batman { }
    .css-class {
      color: blue;
      border : 1px solid black;
    }
  </style>
</head>
<body>
  <div style="font-size:xx-large" id="batman">Batman siempre gana.</div>
  <script src="css.js"></script>
</body>
</html>
```



Propiedad `style`

- Permite obtener /modificar los estilos

```
var divBatman = document.getElementById( "batman" );  
divBatman.style.color = "blue";  
divBatman.style.border = "1px solid black";
```

- Si la propiedad CSS contiene un guión, para usarla mediante JavaScript, se usa la notación *camelCase*. Así pues, `background-color` pasará a usarse como `backgroundColor`.



Clases CSS

- Agrupan varios estilos
- Propiedad **className** (`class` es una keyword de JavaScript)

```
var divBatman = document.getElementById( "batman" );  
divBatman.className = "css-class";  
// divBatman.className = ""; -> elimina la clase CSS
```

- Para añadir más de una clase → separarlas con espacios o utilizar la propiedad **classList**
 - Permite añadir clases → método `add`
 - Para eliminar una clase → `remove`
 - Para cambiar una clase por otra → `toggle`

```
var divBatman = document.getElementById( "batman" );  
divBatman.classList.remove( "css-class" );  
divBatman.classList.add( "css-class2" );
```



Estilo calculado

- Para averiguar el estilo de una determinada propiedad, podemos acceder a la propiedad de `window.getComputedStyle(elem, null).getPropertyValue(cssProperty)`.
- Si el navegador no la soporta (sólo IE antiguos), hay que usar el array `currentStyle`.

```
var divBatman = document.getElementById("batman");  
var color = window.getComputedStyle(divBatman, null).getPropertyValue("color");  
var colorIE = divBatman.currentStyle["color"];
```



Mostrar y Ocultar Contenido

- Propiedad `style.display`
 - `none` → no se muestra el elemento
 - cadena vacía → se muestra

```
var divBatman = document.getElementById("batman");  
divBatman.style.display = "none"; // oculta  
divBatman.style.display = ""; // visible
```



3.4 Animaciones

- Movimiento y manipulación de contenido
- Animación → llamadas sucesivas a una función, con un límite de ejecuciones mediante Timers

```
var velocidad = 2000,
    i = 0;
miFuncion = function() {
    console.log("Batman vuelve " + i);
    i = i + 1;
    if (i < 10) {
        setTimeout(miFuncion, velocidad);
    }
};
setTimeout(miFuncion, velocidad);
```

```
var velocidad = 2000,
    i = 0;
miFuncion = function() {
    console.log("Batman vuelve " + i);
    i = i + 1;
    if (i > 9) {
        clearInterval(timer);
    }
};
var timer = setInterval(miFuncion, velocidad);
```



Ejemplo Movimiento Caja

<http://jsbin.com/daziha/2/edit?html,css,js,output>

```
(function() {
  var velocidad = 10,
  mueveCaja = function(pasos) {
    var el = document.getElementById("caja"),
        izq = el.offsetLeft;

    if ((pasos > 0 && izq > 399) || (pasos < 0 && izq < 51)) {
      clearTimeout(timer);
      timer = setInterval(function() {
        mueveCaja(pasos * -1);
      }, velocidad);
    }

    el.style.left = izq + pasos + "px";
  };

  var timer = setInterval(function () {
    mueveCaja(3);
  }, velocidad);
})();
```

```
<style>
  #caja {
    position: absolute;
    left: 50px;
    top: 50px;
    background-color: blue;
    height: 100px;
    width: 100px;
  }
</style>

<div id="caja"></div>
```



3.5 Eventos

- Asocian un comportamiento a una acción
 - Pulsar un botón, pasar el ratón por encima de un elemento, cargar la página, etc..
- 3 formas:
 1. `elemento.onEvento = manejado` → sólo un manejador
 2. atributo `on*` (`onclick`, `onmouseover`, etc...) de un elemento
 3. `addEventListener(evento, manejador, flujoEvento)` → múltiples manejadores

```
1 var el = document.getElementById("caja");  
  el.onclick = function() {  
    this.style.backgroundColor = "red";  
  };
```

```
2 <button onclick="this.style.backgroundColor='red';">Incrustado</button>
```



Ejemplo Botones y Click

<http://jsbin.com/qawilu/1/edit?html,css,js,output>

```
<head>
  <style>
    .normal {
      background-color: white;
      color: black;
    }
    .contrario {
      background-color: black;
      color: white;
    }
  </style>
</head>
<body class="normal">
  <h1>Hola Eventos</h1>
  <p><a href="http://es.wikipedia.org/wiki/Batman">Batman</a> Forever</p>

  <button>Normal</button>
  <button>Contrario</button>
</body>
```

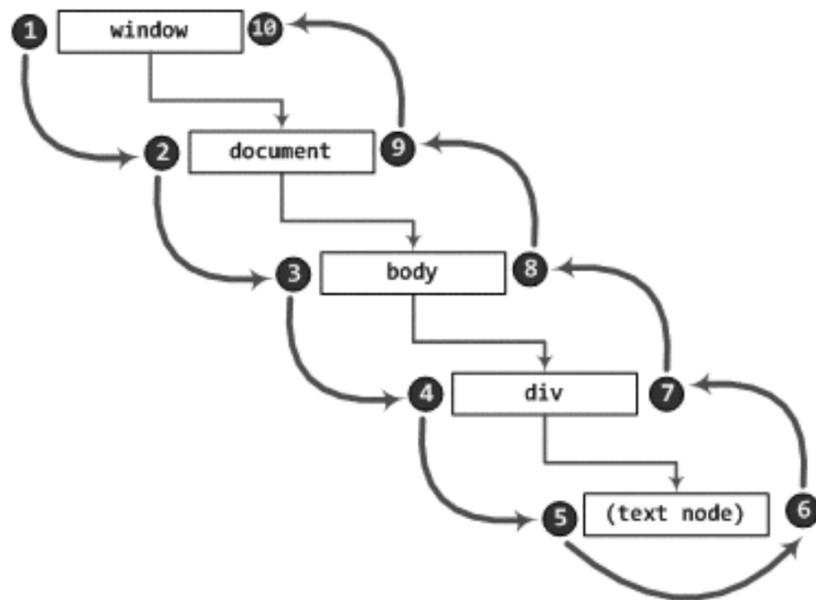
```
(function() {
  var botones = document.getElementsByTagName("button");

  for (var i=0, len=botones.length; i<len; i=i+1) {
    botones[i].onclick = function() {
      var className = this.innerHTML.toLowerCase();
      document.body.className = className;
    };
    // botones[i].onclick = function() {};
  }
})();
```

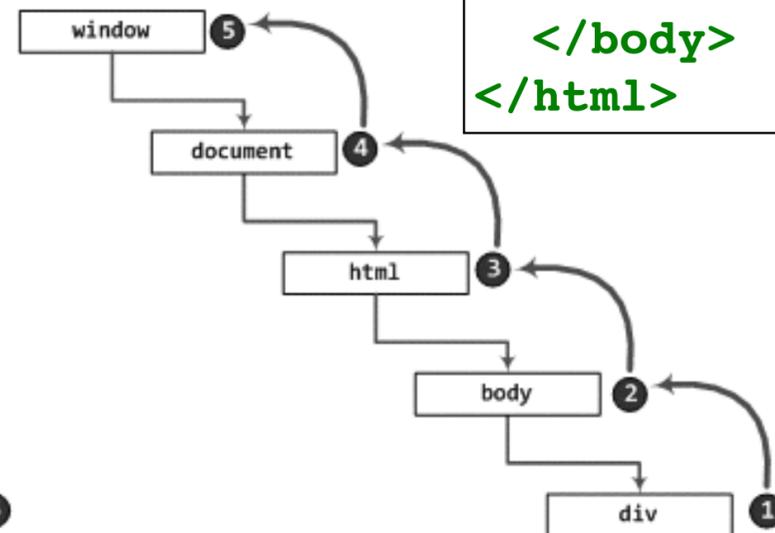


Flujo de Eventos

- **Captura de eventos:** al pulsar sobre un elemento, se produce un evento de arriba a abajo, desde el elemento `window`, pasando por `<body>` hasta llegar al elemento que lo captura.
- **Burbujeo de eventos** (*event bubbling*): el evento se produce en el elemento de más abajo y va subiendo hasta llegar al `window`.



Captura



Burbujeo

```
<html onclick="procesaEvento()">
  <head><title>Ejemplo de flujo de eventos</title></head>
  <body onclick="procesaEvento()">
    <div onclick="procesaEvento()">Pincha aqui</div>
  </body>
</html>
```



Modelo Estándar

- Uso de atributo **on***
- **O addEventListener** (evento, funciónManejador, flujoEvento)
 - flujoEvento: **true** (captura de eventos), **false** (*event bubbling* - recomendado)
- IE8 → **attachEvent**

```
var botones = document.getElementsByTagName("button");
for (var i=0, len=botones.length; i<len; i=i+1) {
  botones[i].onclick = function() {
    var className = this.innerHTML.toLowerCase();
    document.body.className = className;
  };
  // botones[i].onclick = function() {};
}
```

```
var botonClick = function() {
  var className = this.innerHTML.toLowerCase();
  document.body.className = className;
}
var botones = document.getElementsByTagName("button");
for (var i=0, len=botones.length; i<len; i=i+1) {
  botones[i].addEventListener("click", botonClick, false);
  // botones[i].removeEventListener("click", botonClick, false);
}
```



Información del Evento

- Se recibe como parámetro en el manejador
- Propiedades:
 - `type` → evento del que proviene
 - `target` → elemento sobre el cual está registrado

```
var botonClick = function(evt) {  
    var className = this.innerHTML.toLowerCase();  
    document.body.className = className;  
    console.log(evt.type + " - " + evt.target); // click - HTMLButtonElement  
}
```

- Para cancelar un evento → `preventDefault()`

```
var enlaceClick = function(evt) {  
    evt.preventDefault();  
}
```



Delegación de Eventos

- Se basa en el flujo de eventos ofrecidos por *event bubbling*
- Delega un evento desde un elemento inferior en el DOM que va a subir como una burbuja hasta el exterior.

```
(function() {  
  document.addEventListener("click", function(evt) {  
    var tag = evt.target.tagName;  
    console.log("Click en " + tag);  
  
    if ("A" == tag) {  
      evt.preventDefault();  
    }  
  }, false);  
})();
```



Evento de Carga

- Para poder asignar un *listener* a un elemento del *DOM*, éste debe haberse cargado.
- Se recomienda incluir el código *JavaScript* al final de la página *HTML*, justo antes de cerrar el *body*.
- `window.onload` → se lanza cuando el documento ha cargado completamente (incluye las imágenes)

```
function preparandoManejadores() {
    var miLogo = document.getElementById("logo");
    miLogo.onclick() {
        alert("Has venido al sitio adecuado.");
    }
}

window.onload = function() {
    preparandoManejadores();
}
```



Eventos de Foco

- `focus` → al tomar el foco
- `blur` → al perder el foco

```
var campoNombre = document.getElementById("nom");
campoNombre.value = "Escribe tu nombre";

campoNombre.onfocus = function() {
    if ( campoNombre.value == "Escribe tu nombre" ) {
        campoNombre.value = "";
    }
};

campoNombre.onblur = function() {
    if ( campoNombre.value == "" ) {
        campoNombre.value = "Escribe tu nombre";
    }
};
```

```
<form name="miForm">
  Nombre: <input type="text" name="nombre" id="nom" tabindex="10" />
  Apellidos: <input type="text" name="apellidos" id="ape" tabindex="20" />
</form>
```



Eventos de Ratón

- Al hacer click: `mousedown` → `mouseup` → `click`
- Si sucede dos veces de manera consecutiva → `dblclick`
- Coordenadas: `clientX`, `clientY`
- Movimiento: `mousemove`
- Entrar y salir de un elemento: `mouseover`, `mouseout`
 - `target`: referencia el nodo que ha lanzado el evento
 - `relatedTarget`: indica el nodo de donde viene el ratón (para `mouseover`) o adonde va (para `mouseout`)
- Cuidado con el *event bubbling*
 - Al asociar un manejador a un botón, lo normal es que sólo nos interese si ha hecho click.
 - Si asociamos el manejador a un nodo que tiene hijos, al hacer click sobre los hijos el evento "burbujea" hacia arriba, por lo que nos interesará averiguar que hijo ha sido el responsable (propiedad `target`)



Ejemplo Eventos de Ratón

```
var miLogo = document.getElementById("logo");
miLogo.onclick() {
    alert("Has venido al sitio adecuado.");
}
```

```
miParrafo.addEventListener("mouseover", function(event) {
    if (event.target == miParrafo)
        console.log("El ratón ha entrado en mi párrafo");
}, false);
```



Eventos de Teclado

- **keydown**: al pulsar una tecla; también se lanza si se mantiene pulsada
- **keyup**: al soltar una tecla
- **keypress**: tras soltar la tecla, pero sin las teclas de modificación; también se lanza si se mantiene pulsada
- Secuencia de eventos:
 - Carácter alfanumérico: `keydown` → `keypress` → `keyup`.
 - Otro tipo de tecla: `keydown`, `keyup`.
 - Carácter alfanumérico pulsado: se repiten de forma continua los eventos `keydown` y `keypress`
 - Otro tipo de tecla pulsada: se repite el evento `keydown` de forma continua.
- Usaremos `keydown` y `keyup` para averiguar que tecla se ha pulsado, por ejemplo los cursores.
- Si estamos interesado en el carácter pulsado, entonces usaremos `keypress`.



Ejemplo Eventos de Teclado

```
<input type="text" name="cajaTexto" id="cajaTexto" />
```

```
(function() {  
  var caja = document.getElementById("cajaTexto");  
  document.addEventListener("keypress", function(evt) {  
  
    var ascii = evt.charCode;  
  
    if (ascii >= 65 && ascii <=90) {  
      // solo dejamos mayúsculas  
      // las minúsculas van del 97 al 122  
    } else {  
      evt.preventDefault();  
    }  
  }, false);  
})();
```



Teclas especiales

- Al usar los eventos `keydown` y `keyup`, podemos consultar a partir del evento las propiedades:
 - `keyCode`: obtiene el código ASCII del elemento pulsado
 - `altKey`: devuelve true/false si ha pulsado la tecla ALT (option en MAC)
 - `ctrlKey`: devuelve true/false si ha pulsado la tecla CTRL
 - `shiftKey`: devuelve true/false si ha pulsado la tecla SHIFT
 - `metaKey`: devuelve true/false si ha pulsado la tecla command de MAC

```
document.addEventListener("keydown", function(evt) {  
    var code = evt.keyCode;  
    var ctrlKey = evt.ctrlKey;  
  
    if (ctrlKey && code === 66) {  
        console.log("Ctrl+B");  
    }  
  
}, false);
```



Eventos y Closures

- Los *closures* permiten asociar una función a un evento

```
function cambiarTamanyo(tamanyo) {  
  return function() {  
    document.body.style.fontSize = tamanyo + 'px';  
  };  
}
```

```
var tam12 = cambiarTamanyo(12);  
var tam14 = cambiarTamanyo(14);  
var tam16 = cambiarTamanyo(16);
```

```
<button id="tam-12">12</button>  
<button id="tam-14">14</button>  
<button id="tam-16">16</button>
```

```
document.getElementById('size-12').onclick = tam12;  
document.getElementById('size-14').onclick = tam14;  
document.getElementById('size-16').onclick = tam16;
```



3.6 Trabajando con Formularios

- Para interactuar con un formulario, es conveniente asignar un `id`
- Si no tiene `id`, pero si `name` → `document.forms.nombreDelFormulario`
- Para los campos, bien por su `id` o si tienen nombre → `document.forms.nombreDelFormulario.nombreDelCampo`

```
<form name="formCliente" id="frmClnt">
<fieldset id="infoPersonal">
  <legend>Datos Personales</legend>
  <p><label for="nombre">Nombre</label>
    <input type="text" name="nombre" id="nom" /></p>
  <p><label for="correo">Email</label>
    <input type="email" name="correo" id="email" /></p>
</fieldset>

<!-- ...Dirección ... -->

</fieldset>
</form>
```

```
// Mediante el atributo name
var formulario = document.forms.formCliente;
var correo = formulario.correo;
// Mediante el atributo id
var formuId = document.getElementById("frmClnt");
var correoId = document.getElementById("email");
```



Validación Formularios

- Al enviar un formulario, se produce el evento `submit`
 - Dentro del manejador, devolvemos `true` si las validaciones son correctas

```
function preparandoManejadores() {
    document.getElementById("frmClnt").onsubmit = function() {
        var ok = false;
        // validamos el formulario
        if (ok) {
            return true; // se realiza el envío
        } else {
            return false;
        }
    };
}

window.onload = function() {
    preparandoManejadores();
};
```



Campos de Texto

- `type="text"` (o `"password"` o `"hidden"`)
- Contenido mediante propiedad `value`
- Los eventos que puede lanzar son: `focus`, `blur`, `change`, `keypress`, `keydown` y `keyup`.

```
var correoId = document.getElementById("email");  
if (correoId.value == "") {  
    alert("Por favor, introduce el correo");  
}
```



Desplegables

- Etiqueta `select`
- Propiedad `type` → indica si se trata de una lista de selección única (`select-one`) o selección múltiple (`select-multiple`).
- Al seleccionar un elemento, se lanza el evento `change`.
- Para acceder al elemento seleccionado
 - Selección única: propiedad `selectedIndex` (de 0 a n-1).
 - Selección múltiple: recorrer el array de `options` y consultar la propiedad `selected` → `options[i].selected`.
- Una vez tenemos un elemento/opción (objeto `Option`), podemos acceder a la propiedad
 - `value` → obtiene el valor
 - `text` → texto



Ejemplo Desplegables

```
<form name="formCliente" id="frmClnt">
<!-- ... Datos Personales ... -->

<fieldset id="direccion">
  <legend>Dirección</legend>
  <p><label for="tipoVia">Tipo de Via</label>
    <select name="tipoVia" id="tipoViaId">
      <option value="calle">Calle</option>
      <option value="avda">Avenida</option>
      <option value="pza">Plaza</option>
    </select>
  </p>
  <p><label for="domicilio">Domicilio</label>
    <input type="text" name="domicilio" id="domi" /></p>
</fieldset>
</form>
```

```
var tipoViaId = document.getElementById("tipoViaId");

tipoViaId.onchange = function() {
  var indice = tipoViaId.selectedIndex; // 1
  var valor = tipoViaId.options[indice].value; // avda
  var texto = tipoViaId.options[indice].text; // Avenida
};
```



Manipulando Elementos del Desplegable

- Cada elemento es un objeto `Option`
- Para añadir elementos a la lista de manera dinámica → `add(option, lugarAntesDe)`
 - Si no se indica el lugar se insertará al final de la lista
- Para eliminar un elemento → `remove(indice)`

```
var op = new Option("Camino Rural", "rural");  
tipoViaId.add(op, tipoViaId.options[3]);
```



Opciones

- Tanto los *radio* como los *checkboxes* tienen la propiedad `checked` que nos dice si hay algún elemento seleccionado (`true` o `false`).
- Para averiguar cual es el elemento marcado, tenemos que recorrer el array de elementos que lo forman.
- Eventos → `click`, `change`

```
color.checked = true;

var colorElegido = "";

for (var i = 0, l = color.length; i < l; i = i + 1) {
    if (color[i].checked) {
        colorElegido = color[i].value;
    }
}
```



¿Preguntas?