LENGUAJES DE MARCAS

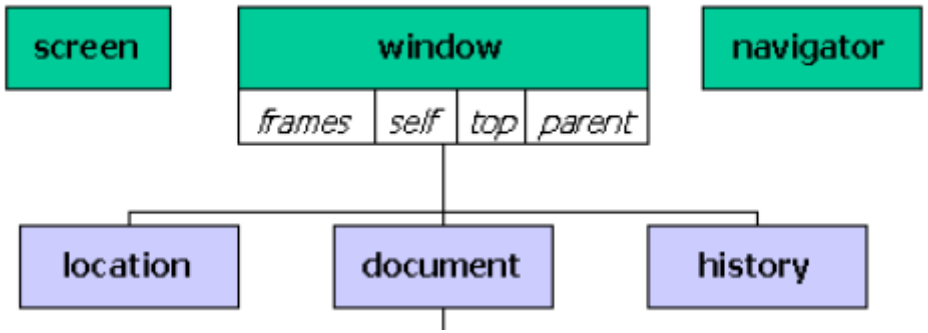
BOM y DOM en JavaScript

Pablo Matías Garramone Ramírez

BOM y DOM

- Normalmente, cuando programamos en JavaScript, nos encontramos dentro del contexto de un navegador (esto no pasa si desarrollamos con NodeJS por ejemplo).
- Dentro de este contexto, tenemos algunos objetos predefinidos y funciones que podemos usar para interactuar con dicho navegador y con el documento HTML, para ello tenemos el BOM y el DOM
- Se suele confundir el BOM y el DOM como que es lo mismo, pero no es así.
- El BOM son todos los objetos referentes al navegador, acciones que podemos realizar fuera del propio contenido de la página, mover una ventana, cambiar su dimensión, volver a una página anterior, etc.
- El DOM sin embargo, es una API para acceder a los elementos internos de un documento, es una estructura jerárquica que nos permite ir de nodos padre a hijos como veremos enseguida.

BOM (Browser Object Model)



- **WINDOW: Objeto de nivel superior del DOM**
  - OPEN, CLOSE
  - ALERT, CONFIRM, PROMPT, STATUS
  - Si usamos la actual no hace falta poner Windows

- **NAVIGATOR: Contiene detalles de la versión actual del Navegador**
  - appName: Nombre del Navegador
  - appVersion
  - plugins
  - platform

- **SCREEN: Es un elemento que identifica propiedades de la pantalla del navegador.**
  - availHeight : El alto de la pantalla en pixeles disponible para el navegador.
  - availWidth : El ancho de la pantalla en pixeles disponible para el navegador.
  - colorDepth : Representa el número de bits usados para representar los colores.
  - height : El alto de la pantalla en pixeles.
  - width : El ancho de la pantalla en pixeles. document.write('width :' + screen.width + '
');
  - Para redimensionar: window.resizeTo(640,480);

- **Ver ejemplos del BOM en los recursos dados**

Pablo Matías Garramone Ramírez

BOM II

- **DOCUMENT: Propiedades que se relacionan con el documento HTML en uso, colores, forma, etc. Casi todas las etiquetas se reflejan aquí.**
  - Document.url
  - Document.location: Localización del documento
  - Document.title
  - Document.lastmodified
  - Document.bgcolor (fondo del documento)
  - Document.fgcolor (color del texto)
  - Document.linkcolor
  - Document.referrer (página anterior visitada por el usuario)
  - Document.anchors
  - Document.links… Forma una array que podemos actuar sobre
    - Length
    - target
    - href

- **LOCATION: Es un reflejo de la ubicación del documento actual con respecto a su URL correspondiente.**
  - Permite que se modifique la URL de un documento.
  - Href: Especifica la URL del documento actual
  - Target

- **History: Es una lista de las URL visitadas**
  - Go
  - Back
  - forward

Pablo Matías Garramone Ramírez

BOM. Objeto Window

- Info: https://www.w3schools.com/jsref/obj_window.asp
- Representa el navegador y es el objeto principal. Todo esta contenido dentro de window (variables globales, el objeto document, el objeto location, el objeto navigation, etc.).
- Puede ser omitido accediendo a las propiedades directamente.

```
'use strict';
// Tamano total de la ventana (excluye la barra superior del navegador)
console.log(window.outerWidth + " - " + window.outerHeight);
window.open("https://www.google.com");
// Propiedades de la pantalla
console.log(window.screen.width + " - " + window.screen.height); // Ancho de pantalla y alto (Resolución)
console.log(window.screen.availWidth + " - " + window.screen.availHeight); // Excluyendo la barra del S.O.
// Propiedades del navegador
console.log(window.navigator.userAgent); // Imprime la información del navegador
window.navigator.geolocation.getCurrentPosition(function(position) {
console.log("Latitude: " + position.coords.latitude + ", Longitude: " + position.coords.longitude);
});
// En las variables globales podemos omitir el objeto window (esta implícito)
console.log(history.length); // Numero de paginas del history. Lo mismo que window.history.length
```

Pablo Matías Garramone Ramírez

Objeto Window. Ventanas de Diálogos

- Ya los vimos en el bloque 1. Alert, confirm y prompt. Pertenecen al objeto window.
- Son una buena opción para hacer pruebas.
- Sin embargo, estos no son personalizables y por tanto cada navegador implementa el suyo a su manera.
- Por ello no es recomendable usarlos en una aplicación en producción. En producción deberíamos usar diálogos construidos con HTML y CSS o programados con JQuery.

Objeto Window. Timers (avisadores)

- El objeto window dispone de una serie de funciones para crear timers y alarmas.
- Hay dos tipos de timers:
  - timeouts. Se ejecutan solo 1 vez.
  - intervals: Se repiten cada X segundos sin parar o hasta que los cancelemos

Pablo Matías Garramone Ramírez

Objeto Window. Timers. Timeouts

- **timeout(funcion, milisegundos)** → Ejecuta una función pasados un número de milisegundos

```
// Imprime inmediatamente la fecha actual
console.log(new Date().toString());
// Se ejecutara en 5 segundos (5000 ms)
setTimeout(() => console.log(new Date().toString()), 5000);
```

- **cleartimeout(timeoutId)** → Cancela un timeout (antes de ser llamado)

```
// setTimeout devuelve un numero con el id, y a partir de ahi, podremos cancelarlo
let idTime = setTimeout(() => console.log(new Date().toString()), 5000);
clearTimeout(idTime); // Cancela el timeout (antes de que se ejecute)
```

Pablo Matías Garramone Ramírez

Objeto Window. Timers. Inervals

- **setInterval(función, milisegundo)** → La diferencia con timeout es que cuando el tiempo acaba y se ejecuta la función, se resetea y se repite cada X milisegundos automáticamente hasta que nosotros lo cancelemos.

```
let num = 1;
// Imprime un numero y lo incrementa cada segundo
setInterval(() => console.log(num++), 1000);
```

- **clearInterval(idIntervalo)** → Cancela un intervalo (no se repetirá mas).

```
let num = 1;
let idInterval = setInterval(() => {
    console.log(num++);
    if(num > 10) { // Cuando imprimimos 10, paramos el timer para que no se repita más
      clearInterval(idInterval);
    }
}, 1000);
```

Pablo Matías Garramone Ramírez

Objeto Window. Timers. Funciones con parámetros

- **setInterval/setTimeOut(nombreFuncion, milisegundos, argumentos…)** → Podemos pasarle un nombre función existente. Si se requieren parámetros podemos establecerlos tras los milisegundos.

```
function multiply(num1, num2) {
    console.log(num1 * num2);
}
// Despues de 3 segundos imprimirá 35 (5*7)
setTimeout(multiply, 3000, 5, 7);
```

Pablo Matías Garramone Ramírez

BOM. Objeto Location

- Info: https://www.w3schools.com/jsref/obj_location.asp
- Contiene información sobre la url actual del navegador. Podemos acceder y modificar dicha url usando este objeto.

```
// Imprime la URL actual
console.log(location.href);
// Imprime el nombre del servidor (o la IP) como “localhost” 192.168.0.34
console.log(location.host);
// Imprime el numero del puerto (normalmente 80)
console.log(location.port);
// Imprime el protocolo usado (http o https)
console.log(location.protocol);
// Recarga la pagina actual
location.reload();
// Carga una nueva pagina. El parámetro es la nueva URL
location.assign("https://google.com");
// Carga una nueva pagina sin guardar la actual en el objeto history
location.replace("https://google.com")
// El más usado. Ir a otra URL desde javaScript, igual que assign
location.href=“https://iessanvicente.com/”
```

Pablo Matías Garramone Ramírez

BOM. Objeto history

- Info: https://www.w3schools.com/jsref/obj_history.asp
- Para navegar a través de las paginas que hemos visitado en la pestaña actual, podemos usar el objeto **history**. Este objeto tiene métodos bastante utiles:

```
// Imprime el numero de paginas almacenadas
console.log(history.length);
// Vuelve a la pagina anterior
history.back();
// Va hacia la siguiente pagina
history.forward();
// Va dos paginas adelante (-2 iria dos paginas hacia atrás)
history.go(2);
```

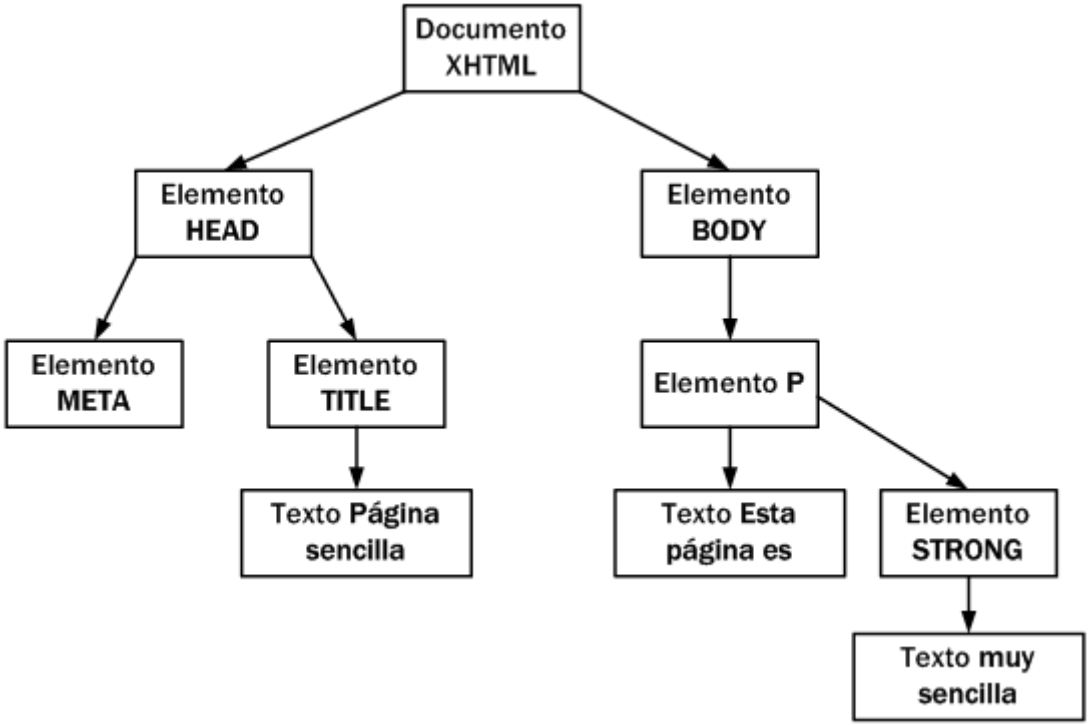
Pablo Matías Garramone Ramírez

DOM. Document Object Model

- La tarea fundamental de JavaScript es la manipulación de las páginas web.
- Una página web no es más que un conjunto de caracteres y etiquetas, lo cuál es sumamente difícil de manejar.
- El DOM trata el XHTML como si fuera XML, y gestiona cada elemento de la página como si fuera un nodo, que forman una estructura en árbol que incluye incluso los atributos.
- Una estructura en árbol es fácil de recorrer y de gestionar.
- Manipular el DOM usando solo JavaScript es algo más complicado que con la ayuda de librerías como JQuery o frameworks como Angular. Pero quedan fuera del ámbito de está asignatura, por lo que lo haremos con JavaScript.

DOM. Ejemplo gráfico

```
<!DOCTYPE html“
<html language=“es”>
  <head>
        <meta charset=“utf-8>
        <title>Página sencilla</title>
  </head>
  <body>
        <p>Esta página es <strong>muy sencilla</strong></p>
  </body>
</html>
```



Pablo Matías Garramone Ramírez

DOM. Estructura

- El objeto principal del DOM es **document**. Este es un objeto global del lenguaje.
- Cada nodo HTML contenido dentro del documento es un **objeto element**, y estos elementos contienen otros nodos, **atributos y estilo**.
- La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:
  - Las etiquetas XHTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en su contenido.
  - Si una etiqueta XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, siendo los nodos generados hijos de su etiqueta padre.

DOM. 12 Tipos de Nodos

- **Document**: Nodo raíz del que derivan todos los demás nodos del árbol.
- **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.
- **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.
- **Text**: Nodo que contiene el texto encerrado por una etiqueta XHTML.
- **Comment**: Representa los comentarios incluidos en la página XHTML.
- **DocumentType**: Representa el nodo DOCTYPE y sus características.
- **DocumentFragment**: Es un trozo de elementos del objeto document sin padre. Se suele usar para crear parte de la página con DOM pero sin que afecte a la página, y cuando ya este finalizado lo incorporamos entero a la página.

- El resto de tipos los ignoramos porque solo afectan al XML:
  - CDataSection: Representa un nodo de una sección CDATA, solo se usa en XML, no en HTML así que lo podemos ignorar.
  - Entity: Representa un nodo Entity.
  - EntityReference: Representa un nodo Entity.
  - ProcessingInstruction: Se trata de instrucciones de aplicación en el XML
  - Notation: Nodo de notación de los DTD.

Pablo Matías Garramone Ramírez

DOM. Funciones

- **Para trabajar con el DOM disponemos de una serie de funciones, que podemos clasificar en 3 grupos:**
  - Funciones de navegación de nodos del DOM
  - Funciones de selección de nodos del DOM
  - Funciones de manipulación de nodos del DOM

Pablo Matías Garramone Ramírez

DOM. Navegación por nodos

- **document.documentElement** → Devuelve el elemento <html>
- **document.head** → Devuelve el elemento <head>
- **document.body** → Devuelve el elemento <body>
- **document.getElementById("id")** → Devuelve el elemento que tiene el id especificado, o **null** si no existe.

```
var cabecera = document.getElementById("cabecera");
<div id="cabecera">
        <a href="/" id="logo">...</a>
</div>
```

- **document.getElementsByClassName("class")** → Devuelve un array de elementos que tengan la clase especificada. Al llamar a este método desde un nodo (en lugar de document), buscará los elementos a partir de dicho nodo.
- **document.getElementsByTagName("HTML tag")** → Devuelve un array con los elementos con la etiqueta HTML especificada. Por ejemplo "p" (párrafos).

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

- **element.childNodes** → Devuelve un array con los descendientes (hijos) del nodo. Esto incluye los nodos de tipo texto y comentarios.
- **element.chidren** → Igual que arriba pero excluye los comentarios y las etiquetas de texto (solo nodos HTML). Normalmente es el recomendado.
- **element.parentNode** → Devuelve el nodo padre de un elemento.
- **element.nextSibling** → Devuelve el siguiente nodo del mismo nivel (el hermano). El metodo **previousSibling** hace justo lo opuesto. Es recomendable usar **nextElementSibling** o **previousElementSibling** si queremos obtener solo los elementos HTML.

Pablo Matías Garramone Ramírez

DOM. Navegación por nodos. Ejemplos

- **Ejemplo.html**

```
<!DOCTYPE>
<html>
  <head>
      <title>JS Example</title>
  </head>
  <body>
      <ul>
        <li id="frstListElement">Element 1</li>
        <li>Element 2</li>
        <li>Element 3</li>
      </ul>
      <script src="./Ejemplo.js"></script>
  </body>
</html>
```

- **Ejemplo.js**

```
// Devuelve <li>
let frstLi = document.getElementById("frstListElement");
console.log(frstLi.nodeName); // Imprime "LI"
// Imprime 1. (elemento -> 1, atributo -> 2, texto -> 3, comentario -> 8)
console.log(frstLi.nodeType);
// Imprime "Element 1". El primer (y único) hijo es un nodo de texto
console.log(frstLi.frstChild.nodeValue);
// Imprime "Element 1". Otra forma de obtener el contenido (texto)
console.log(frstLi.textContent);
// Itera a traves de todos los elementos de la lista
let liElem = frstLi;
while(liElem !== null) {
 // Imprime el texto de dentro del elemento <li>
  console.log(liElem.innerText);
 // Va al siguiente elemento de la lista <li>
  liElem = liElem.nextElementSibling;
}
// Obtiene el elemento <ul>. Similar a parentNode.
let ulElem = frstLi.parentElement;
/* Imprime el codigo HTML de dentro del elemento <ul>:
<li id="frstListElement">Element 1</li>
<li>Element 2</li>
<li>Element 3</li> */
console.log(ulElem.innerHTML);
```

Pablo Matías Garramone Ramírez

DOM. Selector Query

- Una de las principales características que JQuery introdujo cuando se lanzó (en 2006) fue la posibilidad de acceder a los elementos HTML basándose en selectores CSS (clase, id, atributos,….).
- Desde hace años, los navegadores han implementado esta característica de forma nativa (selector query) sin la necesidad de usar jQuery y son:

- **document.querySelector("selector")** → Devuelve el primer elemento que coincide con el selector

- **document.querySelectorAll("selector")** → Devuelve un array con todos los elementos que coinciden con el selector.

Pablo Matías Garramone Ramírez

DOM. Selector Query. Posibles selectores

- a → Elementos con la etiqueta HTML <a>
- .class → Elementos con la clase “class”
- **#id** → Elementos con el id “id”
- .class1.class2 → Elementos que tienen ambas clases, “class1” y “class2”
- .class1,.class2 → Elementos que contienen o la clase “class1”, o “class2”
- .class1 p → Elementos <p> dentro de elementos con la clase “class1”
- .class1 > p → Elementos <p> que son hijos inmediatos con la clase “class1”
- **#id** + p → Elemento <p> que va después (siguiente hermano) de un elemento que tiene el id “id”
- **#id** ~ p → Elementos que son párrafos <p> y hermanos de un elemento con el id “id”
- .class[attrib] → Elementos con la clase “class” y un atributo llamado “attrib”
- .class[attrib="value"] → Elementos con la clase “class” y un atributo “attrib” con el valor “value”
- .class[attrib^="value"] → Elementos con la clase “class” y cuyo atributo “attrib” comienza con “value”
- .class[attrib*="value"] → Elementos con la clase “class” cuyo atributo “attrib” en su valor contiene “value”
- .class[attrib$="value"] → Elementos con la clase “class” y cuyo atributo “attrib” acaba con “value”

Pablo Matías Garramone Ramírez

DOM. Selector Query. Ejemplos

- **Ejemplo.html**

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <div id="div1">
      <p>
        <a class="normalLink" href="hello.html"
        title="hello world">Hello World</a>
        <a class="normalLink" href="bye.html"
        title="bye world">Bye World</a>
        <a class="specialLink" href="helloagain.html"
        title="hello again">Hello Again World</a>
      </p>
    </div>
    <script src="./Ejemplo.js"></script>
  </body>
</html>
```

Ejemplo.js

```
// Imprime "hello world"
console.log(document.querySelector("#div1 a").title);
// ERROR: No hay un hijo inmediato dentro de <div id="div1"> el cual sea un enlace <a>
console.log(document.querySelector("#div1 > a").title);
// Imprime "hello world"
console.log(document.querySelector("#div1 > p > a").title);
// Imprime "bye world"
console.log(document.querySelector(".normalLink[title^='bye']").title);
// Imprime "hello again"
console.log(document.querySelector(".normalLink[title^='bye'] + a").title);
// Imprime "hello world" y "bye world"
let elems = document.querySelectorAll(".normalLink");
elems.forEach(function(elem) {
      console.log(elem.title); });
// Atributo title empieza por “hello...”
let elems2 = document.querySelectorAll("a[title^='hello']");
 // Imprime "hello world" y "hello again"
elems2.forEach(function(elem) {
      console.log(elem.title);
});
// Hermanos de <a title="hello world">
let elems2 = document.querySelectorAll("a[title='hello world'] ~ a");
 // Imprime "bye world" y "hello again"
elems2.forEach(function(elem) {
      console.log(elem.title); });
```

Pablo Matías Garramone Ramírez

DOM. Manipulación del DOM

- **Crear y colocar un NODO**
  - **document.createElement("tag")** → Crea un elemento HTML. Todavía no estará en el DOM, hasta que lo insertemos (usando **appendChild**, por ejemplo) dentro de otro elemento del DOM.
  - **document.createTextNode("text")** → Crea un nodo de texto que podemos introducir dentro de un elemento. Equivale a **element.innerText = "texto"**.
  - **element.appendChild(childElement)** → Añade un **nuevo** elemento hijo al final del elemento padre.

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

- **Colocar un nodo**
  - **element.insertBefore(newChildElement, childElem)** → Inserta un nuevo elemento hijo **antes** del elemento hijo recibido como segundo parámetro.
- **Eliminar un nodo**
  - **element.removeChild(childElement)** → Elimina el nodo hijo que recibe por parámetro.

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
<p id="provisional">...</p>
```

- **Modificar un nodo**
  - **element.replaceChild(newChildElem, oldChildElem)** → Reemplaza un nodo hijo con un nuevo nodo.

Pablo Matías Garramone Ramírez

DOM. Manipulación. Ejemplo

- **Basándonos en el HTML de la lista anterior, vamos a añadir 1 li más y modificar el texto del elemento 3**

```
// Obtiene la primera lista (ul)
let ul = document.getElementsByTagName("ul")[0];
// Tercer elemento de la lista (li)
let li3 = ul.children[2];
// Crea un nuevo elemento de lista
let newLi3 = document.createElement("li");
// Y le asigna un texto
newLi3.innerText = "Now I'm the third element";
// Ahora li3 es el cuarto elemento de la lista (newLi3 se inserta antes)
ul.insertBefore(newLi3, li3);
// Cambiamos el texto para refejar que es el cuarto elemento
li3. innerText = "I'm the fourth element...";
```

- **HTML RESULTANTE**

```
<ul>
  <li id="frstListElement">Element 1</li>
  <li>Element 2</li>
  <li>Now I'm the third element</li>
  <li>I'm the fourth element...</li>
</ul>
```

Pablo Matías Garramone Ramírez

DOM. Atributos

- Dentro de los elementos HTML hay atributos como name, id, href, src, etc. Cada atributo tiene nombre (**name**) y valor (**value**), y este puede ser leído o modificado.

- **element.attributes** → Devuelve el array con los atributos de un elemento
- **element.className** → Se usa para acceder (leer o cambiar) al atributo **class** Otros atributos a los que se puede acceder directamente son: **element.id, element.title**, **element.style** (propiedades CSS)**, …** .
- **element.hasAttribute("attrName")** → Devuelve cierto si el elemento tiene un atributo con el nombre especificado
- **element.getAttribute("attrName")** → Devuelve el valor del atributo
- **element.setAttribute("attrName", "newValue")** → Cambia el valor de atributo.

Pablo Matías Garramone Ramírez

DOM. Atributos. Ejemplos

- **Ejemplo.html**

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <p>
      <a id="toGoogle"
      href="https://google.es"
      class="normalLink">Google
      </a>
    </p>
    <script
      src="./Ejemplo.js"></script>
  </body>
</html>
```

- **Ejemplo.js**

```
let link = document.getElementById("toGoogle");
// Equivale a: link.setAttribute("class", "specialLink");
link.className = "specialLink";
link.setAttribute("href", "https://twitter.com");
link.textContent = "Twitter";
// Si no tenia el atributo title, establecemos uno
if(!link.hasAttribute("title")) {
  link.title = "Ahora voy aTwitter!";
}
/* Imprime: <a id="toGoogle“
  href="https://twitter.com" class="specialLink"
  title="Ahora voy a Twitter!">Twitter</a> */
console.log(link);
```

Pablo Matías Garramone Ramírez

DOM. El atributo Style

- El atributo style permite modificar las propiedades CSS. La propiedad CSS a modificar deben escribirse con el formato **camelCase**, mientras que en CSS se emplea en el formato **snake-case**.
- Por ejemplo, al atributo **background-color** (CSS), se accede a partir de **element.style.backgroundColor**.
- El valor establecido a una propiedad será un string que contendrá un valor CSS valido para el atributo.
- La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio. A continuación se muestran algunos ejemplos:
  - font-weight se transforma en fontWeight
  - line-height se transforma en lineHeight
  - border-top-style se transforma en borderTopStyle
  - list-style-image se transforma en listStyleImage
- El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento XHTML. En su lugar, DOM utiliza el nombre className para acceder al atributo class de XHTML:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"

<p id="parrafo" class="normal">...</p>
```

Pablo Matías Garramone Ramírez

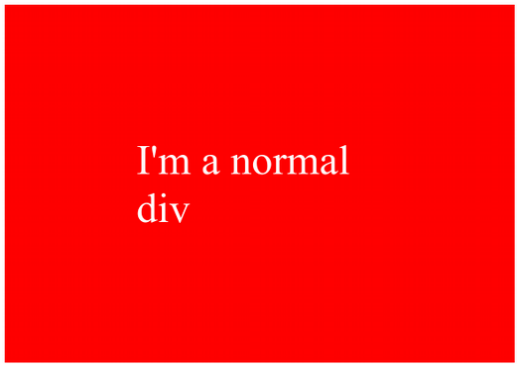
DOM. Atributo Style. Ejemplo

- **Ejemplo.html**

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <div id="normalDiv">I'm a
     normal div</div>
    <script src="./Ejemplo.js">
    </script>
</body>
</html>
```

- **Ejemplo.js**

```
let div = document.getElementById("normalDiv");
div.style.boxSizing = "border-box";
div.style.maxWidth = "200px";
div.style.padding = "50px";
div.style.color = "white";
div.style.backgroundColor = "red";
```



Pablo Matías Garramone Ramírez

DOM. InnerHtml, innerText y textContent

- **innerhtml:** Accede al contenido de un elemento html, justo lo que hay entre las etiquetas de apertura y cierre. Y lo hace literalmente, si pones etiquetas html también saldrán

```
<p id="parrafo">hola que tal</p>

let x=document.getElementById("parrafo").innerhtml;
Console.log(x); // Imprimirá "hola que tal"
```

- Incluso podemos modificarlo:

```
document.getElementById("parrafo").innerhtml="Te cambio el texto";
//Esto cambiará el texto de la etiqueta P.
```

- **Innertext**: Es igual que innerhtml, pero randeriza las etiqueta, por lo que no saldrán. Excepto <script> y <style> y elementos ocultos por CSS. Tampoco aparecerán espacios o tabulaciones.
- **textContent**: Igual que innerText, pero en este caso randeriza todos los elementos incluidas las excepciones de innertext y también espacios y tabulaciones.
- Ejemplo diferenciando los tres: https://www.w3schools.com/jsref/prop_node_innertext.asp

Pablo Matías Garramone Ramírez