# Tema 5

## Los objetos de lenguaje

MEDAC
Instituto Oficial de Formación Profesional

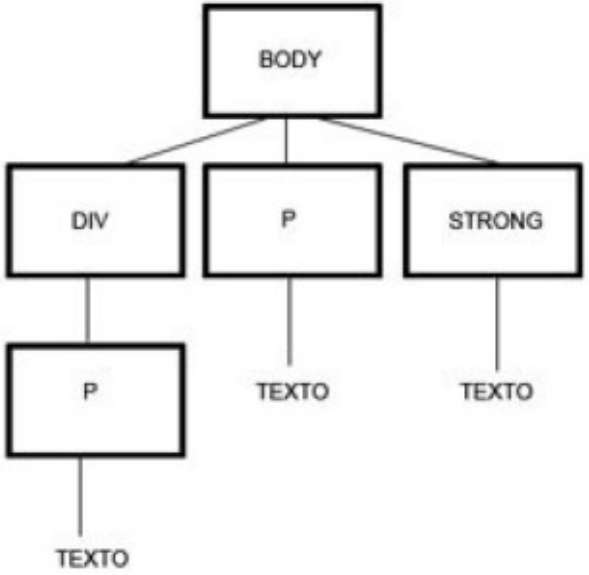
# DOM

- Cualquier página web que se precie está llena de diferentes elementos: párrafos, enlaces, contenedores... Todos estos elementos presentes en una web forman una estructura informática denominada 'árbol' llamada DOM: Document Object Model.

- Un árbol es una estructura informática usada para representar datos que está compuesta por nodos (los diferentes elementos web) y que se interconectan entre sí imitando la forma de un árbol o matorral invertido (arriba la raíz y el árbol se expande hacia abajo).

- Con Javascript vamos a poder acceder de diferentes formas a nodos de ese árbol (etiquetas de mi página) con objeto de alterar sus atributos HTML o sus propiedades CSS. Haciendo esto vamos a poder crear páginas con elementos interactivos y dinámicos que van a cambiar o alterar su comportamiento cuando el visitante lo decida.

# DOM




```
<body>
  <div>
    <p>
      texto
    </p>
  </div>
  <p>
    texto
  </p>
  <strong>
     texto
  </strong>
</div
```

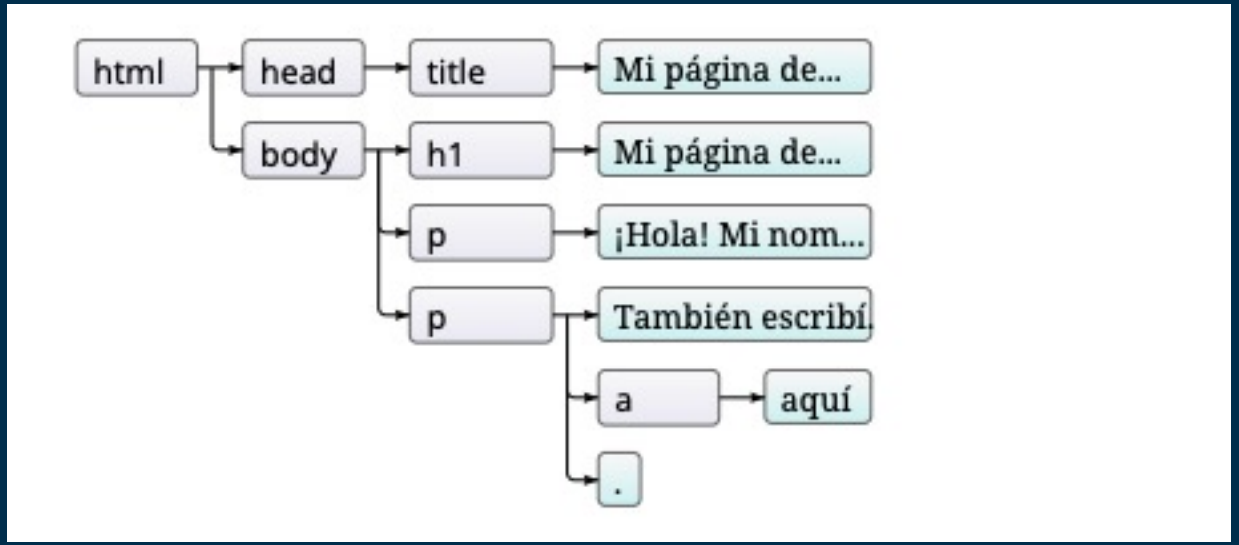
MEDAC

# Crear el árbol DOM de este documento

```
<!doctype html>
<html>
  <head>
    <title>Mi página de inicio</title>
  </head>
  <body>
    <h1>Mi página de inicio</h1>
    <p>Hola, mi nombre es Marijn y esta es mi página de inicio.</p>
    <p>También escribí un libro! Léelo
      <a href="http://eloquentjavascript.net">aquí</a>.</p>
  </body>
</html>
```

MEDAC

# Crear el árbol DOM de este documento




Para cada caja, hay un objeto, con el que podemos interactuar para descubrir cosas como que etiqueta HTML representa y que cajas y texto contiene

MEDAC

# Acceso correcto al DOM

- Antes de aprender cómo modificar los diferentes elementos del árbol DOM de una página es fundamental saber dónde colocar nuestro código, dado que, de no hacerlo correctamente, lo más normal es que no se obtengan los resultados esperados.

- Lo primero es saber que el árbol DOM se crea una vez que se han terminado de cargar todos los elementos de la página. Sin embargo, lo normal es que el Javascript se cargue antes de que lo haga toda la página (hay elementos que tardan más en cargarse como fotos, archivos multimedia...). ¿Qué ocurre si sucede esto? Pues que el Javascript va a intentar acceder a un árbol DOM que aún no se ha creado y, por tanto, no podrá hacer su trabajo correctamente y dará un error.

# Acceso correcto al DOM

Para solucionar esto debemos forzar al Javascript a que espere a que el árbol DOM esté creado. Para ello tenemos varias formas:

1. Colocar el código Javascript al final de la página.
2. Haciendo uso de los eventos load o DOMContentLoaded.
3. Haciendo uso de herramientas de librerías externas (pej, JQuery) (se ve más adelante)

# 1.- Colocando el código Javascript al final de la página

Consiste en colocar todo el código usado para alterar el árbol DOM al final de la página.

Problema: en proyectos de gran complejidad, la mayoría de las veces esto no es posible y, aparte, nos hace colocar código Javascript en otra zona que no es la cabecera y esto, hoy día, no es elegante.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mi pagina</title>
    <script>
        //codigo Javascript que no toca el DOM
    </script>
</head>
<body>
   elementos HTML de mi pagina
</body>
</html>
<script>
        //código que accede al arbol DOM
</script>
```

MEDAC

# 2.- Usando los eventos load o DOMContentLoaded

- Esta forma es mucho más recomendable. Consiste en usar algún evento que nos indique cuando se ha cargado el árbol DOM. Veremos los eventos más adelante, pero se puede adelantar un evento que nos será muy útil en este punto: El evento load

- Este evento espera a que todos (incluidas imágenes y elementos multimedia) los elementos de la página se carguen y posteriormente lanza el Javascript que tiene asociado y lo ejecuta.

- La forma de actuar es muy sencilla. Basta con colocar el manejador de dicho evento (en el próximo punto veremos qué es eso) y, dentro, el código que se encargará de tocar los nodos del DOM

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mi pagina</title>

    <script>
        //codigo Javascript que no toca el DOM

        window.onload = function(){
            //codigo que accede al DOM
        }
    </script>
</head>
<body>
   elementos HTML de la pagina
</body>
</html>
```

MEDAC

# 2.- Usando los eventos load o DOMContentLoaded

Nota: Aunque todavía se suele usar este evento, para estos casos, es mucho mejor usar el evento DOMContentLoaded que, como su nombre indica, espera a que se cargue el árbol DOM para lanzar su código Javascript asociado. Esto es mucho más eficiente a esperar que se cargue toda la página (imágenes y contenido multimedia incluidos) como hace el evento load.

Sin embargo, DOMContentLoaded solo se puede usar con funciones asociadoras de eventos y eso es algo que se explica más adelante , por lo que, de momento, debemos usar el evento load.

# Acceso a nodos

- Cuando creamos una página o aplicación web usando HTML, automáticamente se va a crear el árbol DOM asociado. Como se ha indicado antes, un nodo del árbol DOM no es más que un elemento HTML de la estructura de la página/aplicación.

- Con Javascript vamos a poder acceder directamente a cualquier elemento o un grupo de elementos del árbol DOM de nuestra pagina/aplicación y guardar esa referencia en una variable. De esta forma vamos a poder editar/añadir/quitar propiedades HTML y CSS a nuestro gusto e incluso vamos a poder crear y/o eliminar nodos del árbol a nuestro gusto.

# Objeto: document

- En JavaScript, document es un objeto que representa toda la estructura de la página HTML cargada en el navegador. Es la puerta de entrada para interactuar con el árbol DOM, que organiza todos los elementos de la página de forma jerárquica.

- Al cargar una página web, el navegador crea una versión en memoria de la estructura HTML. document es el objeto JavaScript que permite acceder a esta estructura.

- Lo primero que debemos hacer cuando queremos “tocar” nodos del árbol DOM es conseguir una referencia al nodo o grupo de nodos que queremos. Para esto, Javascript nos proporciona las llamadas funciones especializadas que utilizamos con el objeto document

# 1- getElementsByTagName

- La función document.getElementsByTagName(“nombre_de_una_etiqueta”) obtiene todos los elementos de nuestra página cuya etiqueta sea la misma que la que indicamos como parámetro. La función devuelve un array con los elementos encontrados. En cada celda de ese array tenemos una referencia al elemento correspondiente (el orden será el orden de aparición en el código).

```
var parrafos = document.getElementsByTagName("p")
```

Nota: en realidad estas funciones devuelven una estructura HTMLCollection muy similares a los arrays

MEDAC

```
var parrafos = document.getElementsByTagName("p")
```

# 1- getElementsByTagName

- Con la línea de código anterior vamos a obtener un array con todos los párrafos de la página y dicho array lo almacenamos en la variable párrafos. En la posición 0 del array tendremos una referencia al primer párrafo que aparece en el código HTML de la página, en la posición 1, al siguiente párrafo...

- ¿Y qué es eso de “una referencia”? En realidad, una referencia no es más que un objeto (no un número, ni una cadena, ni un booleano). Es un 'indicador' (un 'puntero') que apunta al elemento correspondiente de nuestra página.

- Así pues, el array almacenado en párrafos va a contener punteros que apuntan a los párrafos de la página.

MEDAC

# 1- getElementsByTagName(*)

- Ejemplo: Crear un script en JavaScript que seleccione todos los elementos <p> de una página y modifique su contenido al cargarse la página.
- Instrucciones:
  - Crea una página HTML con varios elementos <p> que contengan texto.
  - En un archivo JavaScript separado, escribe un script que use window.onload para asegurarse de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Usa getElementsByTagName("p") para seleccionar todos los elementos <p> de la página.
  - Recorre cada uno de los elementos <p> y modifica su contenido para que muestre: "Este es el párrafo número X", donde X representa el número del párrafo en orden.

MEDAC

# 2- getElementsByName

- Sintaxis: document.getElementsByName(“nombre”)

- Esta función nos selecciona todos aquellos elementos cuyo atributo name sea el mismo que el indicado como parámetro. El atributo name normalmente es único para cada elemento de nuestra página (salvo para los botones radio del mismo conjunto). Por tanto, la mayoría de las veces vamos a obtener un 'puntero' al elemento con el nombre indicado.

# 2- getElementsByName

- Sintaxis: document.getElementsByName(“nombre”)

```
var elemento = document.getElementsByName("especial");


. . .
<input type='text' name='especial' … >
```

- Con el código anterior vamos a tener en elemento una referencia a la caja de texto con nombre especial.

MEDAC

# 2- getElementsByName

(*)

- Ejemplo: Crear un script en JavaScript que seleccione un campo de texto por su atributo name y muestre su contenido al hacer clic en un botón**.** Al escribir un nombre en la caja de texto y hacer clic en el botón, se debe mostrar el nombre ingresado en un alert. Si el campo de texto no existe, el script debe mostrar un mensaje indicando que no se encontró el elemento.
- Instrucciones:
  - Crea una página HTML con una caja de texto de tipo <input> y un botón.
  - Asigna el atributo name="usuario" a la caja de texto y configura el botón para llamar a una función JavaScript al hacer clic.
  - En un archivo JavaScript separado, escribe el script utilizando window.onload para asegurar que el código se ejecute después de que la página haya terminado de cargarse.
  - Usa getElementsByName("usuario") para seleccionar el campo de texto y, al hacer clic en el botón, muestra el valor ingresado en un mensaje de alerta.

# 3- getElementsByClassName

- Sintaxis: document.getElementsByClassName(“nombre_de_clase”)

- Esta función nos selecciona todos aquellos elementos que pertenezcan a la clase CSS especificada entre paréntesis. Es decir, cuyo atributo class sea el mismo que el indicado como parámetro. Como las clases se usan para crear agrupaciones, esta función siempre va a devolver un array (incluso cuando la clase solo tenga un elemento).

# 3- getElementsByClassName

```
var elementos = document.getElementsByClassName("Emenu");
. . .
<nav>
  <a href='#' class='Emenu'> Inicio </a>
  <a href='#' class='Emenu'> Nosotros </a>
  <a href='#' class='Emenu'> Proyectos </a>
  <a href='#' class='Emenu'> Productos </a>
  <a href='#' class='Emenu'> Contacto </a>
</nav>
```

Con el código anterior vamos a tener en elementos un array de referencias a todos los elementos de la clase Emenu.

MEDAC

# 3- getElementsClassName

(*)

- Ejemplo: Crear un script en JavaScript que seleccione todos los enlaces de navegación con una clase específica y modifique su apariencia al cargarse la página**.** Queremos que al cargar la página, el color de fondo de todos los enlaces de navegación cambia automáticamente a un color específico (en este ejemplo, lightblue).
- Instrucciones:
  - Crea una página HTML con un menú de navegación que incluya varios enlaces dentro de la etiqueta <nav>.
  - Asigna a cada enlace la clase "nav-link".
  - En un archivo JavaScript separado, utiliza window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Usa getElementsByClassName("nav-link") para seleccionar los enlaces de navegación y cambia el color de fondo de cada enlace cuando se cargue la página.

# 4- getElementById

- Sintaxis: document.getElementById(“id”)

- Posiblemente, esta es la forma más común de acceder a un nodo del árbol DOM. Esta función nos selecciona aquel elemento cuyo atributo id coincide con el indicado como parámetro.

- El valor del atributo id debe ser único para cada elemento de mi página, por tanto, esta función siempre devolverá una referencia a un elemento.

# 4- getElemenById

```
var elemento = document.getElementById("yo");
. . .
<div id='pepe'>
  <p name='trozo'> ... </p>
  <a href='#' id='yo'> Pulsa aquí </a>
</div>
```

Con el código anterior vamos a obtener una referencia al enlace dado que es el elemento con id = ‘yo’

# 4- getElementById

(*)

- Ejemplo: Crear un script en JavaScript que seleccione un enlace específico en la barra de navegación y modifique su texto y enlace de destino al cargarse la página**.** Al cargar la página, el enlace "Inicio" debe mostrar el texto "Página Principal" y apuntar al archivo home.html en lugar de index.html.

- Instrucciones:
  - Crea una página HTML con una barra de navegación que contenga varios enlaces dentro de la etiqueta <nav>.
  - Asigna el atributo id="home-link" al enlace "Inicio".
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Utiliza getElementById("home-link") para seleccionar el enlace "Inicio" y cambia su texto a "Página Principal" y su href a "home.html".

# Funciones base

- Aparte de las funciones especializadas vistas en los apartados anteriores, a partir de la versión 6, Javascript proporciona otras dos herramientas para acceder a nodos del árbol DOM. A estas herramientas se las conoce como funciones base. Estas funciones nos permiten seleccionar nodos del árbol usando cualquiera de los selectores de CSS actuales, siendo, por tanto, mucho más versátiles que las funciones especializadas.

# 1- querySelector

- Sintaxis: document.querySelector(“selector_css”)

- Devuelve una referencia al primer elemento que cumpla con el criterio dado por el selector que va entre paréntesis

MEDAC

# 1- querySelector

```
var elemento = document.querySelector("div p");
. . .
<div id='pepe'>
  <p> primero </p>
  <p> segundo </p>
  <p> tercero </p>
</div>
```

Con el código anterior, en la variable elemento obtenemos una referencia al primer párrafo dentro de un div (en el código será primero).

# 1- querySelector. Repaso de selectores CSS

Selectores de tipo

```
p {
    color: blue; /* Aplica el color azul a todos los elementos <p> */
}
```

Selectores de clase

```
.mi-clase {
    font-size: 20px; /* Aplica el tamaño de fuente a todos los elementos con clase "mi-clase" */
}
```

Selectores de id

```
#mi-id {
    background-color: yellow; /* Aplica un fondo amarillo al elemento con ID "mi-id" */
}
```

MEDAC

# 1- querySelector. Repaso de selectores CSS

Selectores de atributos

```
a[href] {
    text-decoration: none; /* Aplica estilos a todos los enlaces que tienen un atributo "href" */
}
```

Selectores de descendientes

```
div p {
    margin: 10px; /* Aplica un margen a todos los <p> que están dentro de un <div> */
}
```

Selectores de hijos

```
ul > li {
    list-style-type: square; /* Aplica un estilo de lista cuadrado a los <li> que son hijos directos de un <ul> */
}
```

MEDAC

# 1- querySelector

(*)

- Ejemplo: Crear un script en JavaScript que cambie el contenido de un párrafo a “¡Hola Mundo!” y que aplique un estilo de color rojo al texto
- Instrucciones:
  - Crea una página HTML que contenta un div que a su vez contenta un párrafo con un texto
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Utiliza querySelector para cambiar el contenido del párrafo a “¡Hola Mundo!” y aplica un color rojo al texto.

# 2- querySelectorAll

- Sintaxis: document.querySelectorAll(“selector_css”)

- Devuelve un array con referencias a todos los elementos que cumplas con el criterio dado por el selector que va entre paréntesis.

Nota: En realidad estas funciones devuelven una estructura llamada NodeList, que son similares a los arrays (pero no son arrays)

# 2- querySelectorAll

```
var elemento = document.querySelectorAll ("div p");
. . .
<div id='pepe'>
  <p> primero </p>
  <p> segundo </p>
  <p> tercero </p>
</div>
```

Con el código anterior, en la variable elemento obtenemos un array con referencias a todos los párrafos dentro de un div. Siempre devuelve un array de referencias, incluso si usamos un selector que afecta a un solo elemento: elementos = querySelectorAll("#pepe");

MEDAC

# Importante: HTMLColletion y NodeList

Aunque las estructuras HTMLCollection y NodeList se comportan de forma “similar” a un array, no son estructuras idénticas y las funciones para array no funcionarán en ellas.
Si alguna vez necesitamos convertir alguna de esas dos estructuras en array forzosamente, podemos hacerlo así:

```
//obtenemos una HTMLCollection
var lista = document.getElementsByTagName(…)
//Se transforma en array
var nuevoArray = Array.from(lista);
```

# Acceso a atributos HTML

- Ya hemos aprendido varias formas de acceder a distintos elementos de nuestra página. Como se ha explicado, con estas funciones obtenemos referencias que apuntan a los elementos deseados. La pregunta lógica que nos podemos hacer se es ¿y ahora qué? ¿qué hago con esas referencias?

- Obtener esas referencias es el primer paso para poder alterar los elementos de mi página. A continuación, vamos a ver como cambiar/crear/eliminar propiedades de los elementos seleccionados.

- Una vez nos hemos posicionado sobre un elemento, podemos consultar o cambiar los valores de todos los atributos HTML de ese elemento. Para hacer esto existen dos formas: la forma clásica (que ya está obsoleta pero aún funciona sin problemas) y la forma actual.

# Acceso a atributos HTML. Forma clásica. Notación punto

- Una vez que tenemos la referencia al objeto, podemos tocar sus atributos usando la notación punto:

```
<table id="tabla" width='230px' height='300px'
border='1px'>

      . . .

</table>
```

MEDAC

# Acceso a atributos HTML. Forma clásica. Notación punto

```
//Obtengo la referencia al elemento
var elemento = document.getElementById("tabla");

//Muestro el valor de su atributo altura
alert(elemento.height);

//cambio valor del atributo border
elemento.border = '5px';
```

- Para el acceso a atributos HTML se coloca la variable con la referencia a ese elemento seguido de punto y el nombre del atributo a consultar.:

- Si lo que queremos es cambiar el valor de un atributo, indicamos ese atributo (usando el punto) y le asignamos un nuevo valor.

MEDAC

# Ejemplo de acceso a atributos

(*)

- Ejemplo: Crear un script en JavaScript que al pulsar un botón muestre en consola los atributos de anchura y borde de una tabla (con dos filas, una de encabezado y otra de datos y dos columnas) y que luego los modifique.
- Instrucciones:
  - Crea una página HTML que contenga una tabla con atributos: width=300px, border=2px y un botón que llame a una función que modifique la tabla.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Dentro de la función de modificación lee los atributos width y border de la tabla y muéstralos por consola
  - Modifica los atributos de la tabla para que tenga esto nuevos valores: width=500px, border=4px

# Acceso a atributos HTML

- Destacar que si a la hora de asignar un nuevo valor al atributo colocamos un valor incorrecto, este atributo quedará sin valor (y por tanto el elemento quedará afectado de alguna manera) hasta que se vuelva a cargar la página

- Si asignamos valor a un atributo que no tiene el elemento, a partir de esa asignación, al elemento se le añadirá el atributo asignado

# Acceso a atributos HTML (*)

```
var elemento = document.getElementById("parra");


elemento.align= "center"
//A partir de este punto, el párrafo quedará alineado al
centro


. . .


<p id= "parra">
  Soy un párrafo con alineación a la izquierda por
defecto.
</p>
```

MEDAC

# Acceso a atributos HTML. Forma actual. Métodos get y set

- A partir de de la versión 6 de JavaScript tenemos disponible una serie de métodos propios de Javascript para el manejo de atributos de elementos HTML

- Nosotros nos vamos a centrar en dos: getAttribute y setAttribute

# Consulta de atributos con getAttribute

```
var elemento = document.getElementById("tabla");

alert(elemento.getAttribute("height"))
```

- El método necesita el nombre del atributo a consultar como parámetro y devuelve el valor que tiene ese atributo
- Si el atributo no existe getAttribute devolverá null o empty string

# Modificar atributos con setAttribute

```
var elemento = document.getElementById("tabla");

elemento.setAttribute("height","200px")
```

- En este caso son necesarios dos argumentos: el atributo a modificar (una cadena) y el nuevo valor del atributo (cadena o número)
- Al igual que pasaba en la forma clásica, si a la hora de asignar un nuevo valor al atributo, colocamos un valor incorrecto, ese atributo quedará sin valor (y por tanto el elemento quedará afectado de alguna manera) hasta que se vuelva a cargar la página.
- Del mismo modo, si asignamos valor a un atributo que no tiene el elemento, a partir de esa asignación, al elemento se le añadirá el atributo asignado

# Ejemplo de acceso a atributos (nueva) (*)

- Ejemplo: Crear un script en JavaScript que al pulsar un botón muestre en consola los atributos de altura, anchura y borde de una tabla (con dos filas, una de encabezado y otra de datos y dos columnas) y que luego los modifique.
- Instrucciones:
  - Crea una página HTML que contenga una tabla con atributos: width=300px, height =200px y border=2px y un botón que llame a una función que modifique la tabla.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Dentro de la función de modificación lee los atributos width, height y border de la tabla y muéstralos por consola
  - Modifica los atributos de la tabla para que tenga esto nuevos valores: width=500px, height=300px border=4px

# Atributos importantes

- Sabemos que en HTML cada etiqueta tiene una serie de atributos posibles. Además, actualmente, todos los atributos que hacen referencia al diseño del elemento, si bien aún funcionan, están obsoletos y no deben usarse (hay que usar CSS).

- Sin embargo, en Javascript existen unos atributos comunes que tienen todos los elementos HTML a los que hacemos referencia

# Atributo innerHTML

- Todos los elementos de nuestra página van a tener un atributo llamado innerHTML el cual guarda el contenido HTML de ese elemento, es decir, todo lo que hay entre las etiquetas de apertura y cierre de ese elemento (ya sea texto u otras etiquetas).

Ejemplo:

```
<div id='padre'>
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

MEDAC

# Atributo innerHTML

Si hacemos:

```
var elemento = document.getElementById('padre');


//muestra el contenido HTML del párrafo
console.log(elemento.innerHTML)
```

Vamos a obtener todo el código HTML contenido en ese nodo:

```
<p id='contenido'>
     En un lugar de la Málaga profunda.
</p>
```

MEDAC

# Atributo innerHTML

- El contenido de ese atributo se puede cambiar. Así pues, aunque no es la mejor manera de hacerlo, podemos alterar el contenido de un elemento a través de su innerHTML

```
var elemento = document.getElementById('contenido');

//cambia el contenido del párrafo
elemento.innerHTML = "Cambio el <b>texto</b>";
. . .
<p id='contenido'>
   En un lugar de la Málaga profunda.
</p>
```

- Al ejecutarse el código anterior el contenido del párrafo del ejemplo cambiará por el indicado en el atributo innerHTML (hasta que la página se vuelva a cargar).

# Atributo innerHTML

Obviamente, el contenido de innerHTML puede consultarse, mostrarse por pantalla, copiarse a otra variable...

```
var p1 = document.getElementById('contenido');
var p2 = document.getElementById('salida');

//copio el contenido de p1 a p2
p2.innerHTML = p1.innerHTML;
. . .
<p id='contenido'>
   En un lugar de la Málaga profunda.
</p>

<p id='salida'>
   No voy a durar mucho así escrito.
</p>
```

MEDAC

# Ejemplo innerHTML

(*)

- Ejemplo: Crear un script en JavaScript que cambie el contenido de un párrafo cuando el usuario haga clic en un botón. Al pulsar el botón, el texto del párrafo debe actualizarse para mostrar un mensaje nuevo en cursiva.
- Instrucciones:
  - Crea una página HTML con un párrafo que tenga el id="text-paragraph" y un botón.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Al hacer clic en el botón, cambia el contenido del párrafo a cursiva usando innerHTML.

# Otro Ejemplo innerHTML

(*)

- Ejemplo: Crear un script en JavaScript que copie el contenido de un párrafo fuente a otro párrafo destino con su formato y cuando el usuario haga clic en un botón.
- Instrucciones:
  - Crea una página HTML con dos párrafos: uno con el id="source-paragraph" que esté en negrita y contenga el texto a copiar, y otro con el id="destination-paragraph" donde aparecerá el contenido copiado.
  - Añade un botón
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Al hacer clic en el botón, copia el contenido del párrafo fuente al párrafo destino usando innerHTML.

# Atributo textContent

Todos los elementos de nuestra página van a tener un atributo llamado textContent. Este atributo se usa para consultar y modificar solo el texto de ese elemento y el de todos sus hijos e ignora las etiquetas HTML que encuentre. Sin embargo, respeta los saltos de líneas y los espacios entre palabras.

Ejemplo

```
<div id='padre'>
   Soy texto fuera del parrafo.
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

MEDAC

# Atributo textContent

Si hacemos:

```
var elemento = document.getElementById('padre');

//muestra todo el texto del párrafo
console.log(elemento.textContent)
```

Vamos a obtener todo el texto que posee ese elemento y sus hijos sin etiquetas HTML

```
Soy texto fuera del parrafo.

   En un lugar de la Málaga profunda
```

MEDAC

# Atributo textContent

- El contenido de ese atributo se puede cambiar. De esta forma podemos alterar el texto de un elemento

```
var elemento = document.getElementById('padre');
//cambia el texto del DIV
elemento.textContent= "Cambio solo el texto";
. . .
<div id='padre'>
   Soy texto fuera del parrafo.
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

- Es importante destacar que, si cambio el texto de un elemento de esta forma, se eliminan todos los hijos que tiene ese elemento

MEDAC

# Atributo textContent

- El DIV del ejemplo anterior quedaría;

```
<div id='padre'>
    Cambio solo el texto
</div>
```

MEDAC

# Ejemplo textContent

(*)

- Ejemplo: Escribe un código HTML que contenga un título <h2> y un botón. Al hacer clic en el botón, el contenido del título debe cambiar a un nuevo mensaje.
- Instrucciones:
  - Crea un archivo HTML con un título <h1> que explique el propósito del ejercicio, un título <h2> con un texto inicial y un botón para realizar la acción.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Añade un evento click al botón para que, cuando se pulse, el contenido del título <h2> cambie a un nuevo texto usando textContent.

# Atributo class (para forma de acceso clásica)

- El atributo class sirve para indicar a que clase CSS pertenece ese elemento. Al ser un atributo HTML podemos mostrarlo o alterarlo con Javascript sin problemas.

- Tan sólo hay que tener en cuenta que, cuando se use la forma clásica de acceso a atributos en Javascript, ese atributo debe indicarse usando className . Esto es debido a que la palabra class se usa para definir clases en Javascript (como ocurre en muchos lenguajes).

# Atributo class (para forma de acceso clásica)

- Sabiendo esto podemos cambiar fácilmente la clase a la que pertenece un elemento usando la forma clásica (notación de punto)

```
var elemento = document.getElementById("tabla");

//mostramos la clase a la que pertenece
alert(elemento.className);

//cambiamos la clase
elemento.className = 'listaUsuarios';
. . .
<table id="tabla" class='usuarios'>
  . . .
</table>
```

- Si cambiamos la clase a un elemento, este perderá todas las propiedades de la antigua clase y obtendrá las propiedades de la clase nueva

MEDAC

# Atributo class (para forma de acceso actual)

- Destacar que si se usa la forma actual de acceso a atributos (getAttribute y/o setAttribute), el atributo class no pierde su nombre y se escribiendo igual.

```
var elemento = document.getElementById("tabla");

console.log(elemento.getAttribute("class"));

elemento.setAttribute("class","listaUsuarios");
```

MEDAC

# Ejemplo className

(*)

- Ejemplo: Escribe un código HTML que contenga un párrafo y un botón. Al hacer clic en el botón, cambia la clase del párrafo para que tenga un estilo distinto.
- Instrucciones:
  - Crea un archivo HTML con un párrafo que tenga un id llamado "parrafo" y un botón.
  - Define dos clases CSS: una clase "estiloOriginal" para el estilo inicial del párrafo y una clase "nuevoEstilo" para el estilo modificado.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Al hacer clic en el botón, cambia la clase del párrafo de "estiloOriginal" a "nuevoEstilo" usando className

# Ejercicios

Ejercicios 1 Árbol DOM. Aula Virtual Tema 5

MEDAC

# this en JavaScript

- this es una palabra especial que se refiere al contexto en el que se está ejecutando el código. En otras palabras, this representa el "dueño" de la función que está siendo ejecutada

- En nuestro caso this puede representar el elemento HTML que genera el evento. Por ejemplo, el botón sobre el que se ha hecho click.

- Dentro de la función asociada al evento, this nos permite acceder a las propiedades y atributos del elemento sin tener que usar id o clases específicas.

- Usar this facilita escribir funciones reutilizables que no dependen de elementos específicos, sino que pueden funcionar con cualquier elemento que las llame.

# Ejemplo usando this

(*)

- Ejemplo: Tenemos un html con varios párrafos. Cada vez que el usuario hace clic en un párrafo cualquiera, queremos cambiar su contenido a un texto “¡Has hecho click en mi!
- Instrucciones:
  - Crea un archivo HTML con 3 párrafos.
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Al hacer clic sobre cualquiera de los tres párrafos, cambia el texto a ¡Has hecho click en mi!

# Eventos

- Javascript es un lenguaje basado en eventos. Es decir, cuando sucede 'algo' en nuestra página, el Javascript se activa y se ejecuta. Ese 'algo' puede ser cuando el usuario pase por encima de una imagen, haga click en un enlace, pulse una tecla, presione un botón... A esas acciones que suceden en nuestras páginas y que provocan la ejecución del Javascript se les denomina eventos.

- Cada elemento HTML de nuestra página tiene asociado una serie de eventos que pueden realizarse sobre él. Algunos elementos tendrán un tipo de evento que no tengan otros y viceversa. O incluso varios elementos pueden tener eventos comunes.

# Eventos

- El ejemplo de uso más claro lo tenemos en nuestra forma de trabajar. Hasta ahora asociamos el evento onclick a un botón, es decir, 'cuando se pulse el botón', se va a ejecutar el código Javascript indicado:

```
<input type="button" value="pulsa" onclick="miFuncion()" >
```

- Sin embargo, un elemento no tiene por qué tener asociado sólo un evento. Vamos a a poder asociar tantos eventos como queramos a un elemento o a varios

# Eventos

```
<input type="button" value="pulsa" onclick="miFuncion()"
                                   onmouseover = "miOtraFuncion()" >
```

- En este ejemplo cuando hagamos click sobre el botón se ejecutará la función miFuncion() y cuando pasemos el ratón por encima del botón, se ejecutará miOtraFuncion()

# Eventos más importantes en Javascript

| Evento | Descripción | Elementos para los que está definido |
|---|---|---|
| onblur | Deseleccionar el elemento (perder el foco) | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento (ganar el foco) | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |

# Eventos más importantes en Javascript

| | | |
|---|---|---|
| `onkeypress` | Pulsar una tecla | Elementos de formulario y `<body>` |
| `onkeyup` | Soltar una tecla pulsada | Elementos de formulario y `<body>` |
| `onload` | La página se ha cargado completamente | `<body>` |
| `onmousedown` | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| `onmousemove` | Mover el ratón | Todos los elementos |
| `onmouseout` | El ratón *"sale"* del elemento (pasa por encima de otro elemento) | Todos los elementos |
| `onmouseover` | El ratón *"entra"* en el elemento (pasa por encima del elemento) | Todos los elementos |
| `onmouseup` | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |

# Eventos más importantes en Javascript

| Evento | Descripción | Elementos para los que está definido |
|---|---|---|
| `oncontextmenu` | Cuando aparece el menú contextual al pulsar el botón derecho del ratón o la tecla correspondiente. | `<body>` y el documento. |
| `onreset` | Inicializar el formulario (borrar todos sus datos) | `<form>` |
| `onresize` | Se ha modificado el tamaño de la ventana del navegador | `<body>` |
| `onselect` | Seleccionar un texto | `<input>`, `<textarea>` |
| `onsubmit` | Enviar el formulario | `<form>` |
| `onunload` | Se abandona la página (por ejemplo al cerrar el navegador) | `<body>` |

# Ejercicios

Ejercicios 2 Árbol DOM. Aula Virtual Tema 5

MEDAC

# Modificar propiedades CSS usando el atributo style

- Al igual que hemos podido acceder a atributos HTML, con Javascript vamos a poder acceder a las propiedades CSS que tenga un objeto seleccionado.

- El modo de proceder es el mismo:
  - Acceder al elemento deseado
  - Cambio el valor de una propiedad colocando la variable donde tengo la referencia seguida de punto, la palabra style, otro punto y el nombre de la propiedad CSS a la que queremos alterar su valor

# Modificar propiedades CSS

```
#tabla{
    border: 1px solid black;
    width: 300px;
    height: 300px;
    padding: 10px;
    margin-left: 20px;
    background-color: aqua;
}
```

```
var elemento = document.getElementById("tabla");
elemento.style.border = '2px solid red';
elemento.style.padding = '0.75em';
elemento.style.width = '500px'
```

- En el código anterior estamos cambiando los valores de las propiedades border, padding y width, Es importante colocar valores entre comillas (son cadenas) y además deben ser correctos para cada propiedad. Si no hacemos eso, estaremos asignando un valor incorrecto a la propiedad y el navegador no la tendrá en cuenta.

# Modificar propiedades CSS usando el atributo style

```
#tabla{
    border: 1px solid black;
    width: 300px;
    height: 300px;
    padding: 10px;
    margin-left: 20px;
    background-color: aqua;
}
```

```
var elemento = document.getElementById("tabla");
elemento.style.marginLeft = 0;
elemento.style.backgroundColor = 'blue';
```

- Si lo que queremos es alterar el valor de una propiedad con nombre compuesto (pej: margin-left) debemos actuar de la siguiente forma:
  - Se elimina el guión (-) del nombre y se juntan las palabras.
  - La primera letra de la segunda palabra irá en mayúsculas.

MEDAC

# Modificar propiedades CSS usando el atributo style

- Hay otra forma alternativa de hacer lo anterior, usando la función setProperty

```
referencia.style.setProperty(propiedad css, valor)
```

```
var elemento = document.getElementById("tabla");

elemento.style.setProperty('margin-left','2px');
elemento.style.setProperty('background-color','blue');
elemento.style.setProperty('width','500px');
```

- Destacar que, usando esta forma, las propiedades compuestas se escriben igual que en código CSS.

MEDAC

# Modificar propiedades CSS

- Tanto usando la forma .style.propiedad como .style.setProperty, lo que estoy haciendo en realidad es añadir valores al atributo style de esa etiqueta (y si no lo tiene, lo añade). No estoy tocando realmente la hoja de estilos.

# Ejemplo modificación CSS

(*)

- Ejemplo: Tenemos un párrafo con un estilo asignado: letra en negro, tamaño 16px, ancho de la fuente normal, padding 10px, boder: 1px solid black. Queremos que al pulsar un botón cambien el estilo del párrafo a: letra en color azul, tamaño 20 px, ancho de la fuente bold, color de fondo amarillo, color del borde naranja
- Instrucciones:
  - Crea un archivo HTML con párrafo y el estilo indicado
  - En un archivo JavaScript separado, usa window.onload para asegurarte de que el código se ejecute solo después de que la página haya terminado de cargarse.
  - Al hacer clic sobre el botón modificar los elementos del estilo indicado.

# Consultar valores CSS de la hoja de estilos

- Lo normal es pensar que podríamos consultar/mostrar/copiar valores de propiedades CSS de la misma forma:

```
console.log(elemento.style.padding);
```

- Sin embargo, esto no es correcto si lo que queremos es mostrar valores de propiedades de la hoja de estilos. Si usamos esta forma, toda aquella propiedad que no esté en el atributo style, va a devolver una cadena vacía, aunque la tengamos declarada en la hoja de estilos.

- Nos serviría para consultar valores del atributo style de un elemento, pero no para hoja de estilos

# Consultar valores CSS de la hoja de estilos

```
p{
    border:  2px solid black;
    width:  500px;
    padding: 10px;
}
```

```
 <p id="parra">
     Texto muy largo para que se vea bien la anchura y el
paddind del elemento.
</p>
```

```
function consulta(){
    var elemento = document.getElementById('parra');
    console.log(elemento.style.padding);
}
```

- Al ejecutar la función, vamos a obtener por consola: <empty string> Esto es debido a que esa propiedad no está definida en el atributo style (en este caso, ni siquiera está definido ese atributo) independientemente de que la propiedad esté definida en la hoja de estilos, como ocurre en este ejemplo.

MEDAC

# Consultar valores CSS de la hoja de estilos

- Es por eso que, para consultar cualquiera de los valores de propiedades CSS de una hoja de estilos, debemos hacer uso de un par de herramientas de Javascript que, por desgracia, complican un poco algo que se podría hacer de forma sencilla tal y como hemos mostrado.

- Para consultar valores de propiedades CSS debemos hacer los siguiente:

  - Obtener una referencia al elemento.
  - Obtener los estilos del elemento con la función window.getComputedStyle(referencia)
  - Usar la función getPropertyValue para obtener el valor de la propiedad deseada.

# Consultar valores CSS

```
p{
   border:  2px solid black;
   width:  500px;
   padding: 10px;
}
```

```
 <p id="parra">
     Texto muy largo para que se vea bien la anchura y el
paddind del elemento.
</p>
```

Obtendríamos el valor de `padding`:

```
function consulta(){
    //paso 1
    var elemento = document.getElementById('parra');
    //paso 2
    var estilos = window.getComputedStyle(elemento);
    //paso 3
    var valor = estilos.getPropertyValue('padding')

    console.log(valor);
}
```

MEDAC

# Consultar valores CSS

- Si deseamos consultar el valor de una propiedad compuesta, la escribiríamos tal y como se escribe en código CSS:

```
let letra = estilo.getPropertyValue('font-size');
let fondo = estilo.getPropertyValue('background-color');
```

# Consultar valores CSS

- Aún usando esta forma para obtener valores CSS, hay que tener presente que, en la mayoría de los casos, no se nos va a mostrar las unidades como aparecen escritas en la hoja de estilos o en el atributo style:

  - Los valores de colores se van a devolver siempre con notación RGB aunque se usen nombres de colores o valores hexadecimales.
  - Para las medidas, siempre va a devolvernos valores en pixeles o porcentajes, independientemente de que usemos puntos, em, auto …

- Obviamente, para todos estos cambios de unidades, Javascript nos hace los cálculos adecuados.

# Consultar valores CSS

```css
p{
   margin: 0 auto;
   background-color: lightgreen;
   width:  500px;
   padding: .75rem;
   font-size: 2em;
}
```

```html
 <p id="parra">
     Texto muy largo para que se vea bien la anchura y el paddind del elemento.
</p>
```

- .75 em equivale al 75% del tamaño de la fuente base definida en el documento HTML ó del navegador si no se define una fuente base
- 2em: doble del tamaño de la fuente del elemento padre

MEDAC

```
var elemento = document.getElementById('parra');
var estilos = window.getComputedStyle(elemento);

//obtenemos valores
let margen = estilos.getPropertyValue('margin')
let fondo = estilos.getPropertyValue('background-color')
let anchura = estilos.getPropertyValue('width')
let padding = estilos.getPropertyValue('padding')
let letra = estilos.getPropertyValue('font-size')

//mostramos los valores
console.log(margen);
console.log(fondo);
console.log(anchura);
console.log(padding);
console.log(letra);
```

**En Firefox**

```
0px auto
rgb(144, 238, 144)
500px
32px
22.4px
```

**En Chrome y Edge**

```
0px 392.5px
rgb(144, 238, 144)
500px
32px
22.4px
```

MEDAC

# Consultar valores CSS

- Destacar que los valores devueltos siempre serán del tipo cadena, ya que,como se ve en el ejemplo, también devuelve las unidades. Nunca va a devolver números, incluso si el valor no tiene unidad definida en la hoja de estilos.

- Si intentamos acceder a una propiedad que existe tanto en el atributo style como en la hoja de estilos, devolverá el valor de la propiedad en el atributo style porque es el más prioritario y es el que se está aplicando.

- Si intentamos acceder a una propiedad que no existe ni en la hoja de estilos ni en el atributo style, Javascript nos devolverá los valores adecuados que tendría esa propiedad por defecto si existiera, pero no realizará cambios

# Ejemplo Consulta CSS

(*)

- Ejemplo: Crea una página web que contenga un cuadro de texto (div) con algún contenido dentro. Junto al cuadro, agrega varios botones que permitan cambiar dinámicamente el estilo del cuadro, incluyendo el color de fondo, el color del texto y el tamaño de la fuente
  Estilo inicial: width: 500px; height: 100px; background-color: lightgray; color: black; font-size: 16px; text-align: center; padding: 20px; border: 2px solid #333
- Instrucciones:
  - Al hacer clic en el botón Cambiar Color de Fondo, el color de fondo del cuadro debe cambiar de gris claro a azul claro y de nuevo a gris claro si se hace clic otra vez.
  - Al hacer clic en Cambiar Color del Texto, el color del texto debe cambiar de negro a verde y viceversa.
  - Al hacer clic en Cambiar Tamaño de la Fuente, el tamaño de la fuente debe alternar entre 16px y 20px

# Obteniendo el tamaño de un elemento

- El tamaño que ocupa un elemento en la página no es el que indica la propiedad width/height de su CSS. A ese valor hay que sumarle los valores (si los hubiera) del padding, del borde y, en algunos casos, del margin.

- Cuando se alteran propiedades de nodos del árbol DOM, el programador no tiene por qué saber los valores de las propiedades CSS. Sin embargo, puede ser necesario conocer cuanto ocupa ese elemento dentro de la estructura de la página

# Obteniendo el tamaño de un elemento

- Para obtener ese valor podemos hacer uso de las siguientes funciones de JavaScript

| Función | Descripción |
|---|---|
| `clientWidth` | Obtiene la *anchura* del elemento incluyendo el padding. |
| `clientHeight` | Obtiene la *altura* del elemento incluyendo el padding. |
| `offsetWidth` | Obtiene la *anchura* del elemento incluyendo el padding y el borde. |
| `offsetHeight` | Obtiene la *altura* del elemento incluyendo el padding y el borde. |

- Si usamos las funciones explicadas hacemos que sea el propio Javascript el que calcule todos los valores de forma correcta

# Obteniendo el tamaño de un elemento

```
<div id='padre'>
  Qué tamaño tengo...
</div>
```

```
#padre{
    width: 400px;
    padding: 10px;
    border: 5px solid black;
    margin: 20px;
}
```

```
var elemento = document.getElementById('padre');

console.log("Anchuras: ")
console.log(elemento.clientWidth)
console.log(elemento.offsetWidth)

console.log("Alturas: ")
console.log(elemento.clientHeight)
console.log(elemento.offsetHeight)
```

```
Anchuras:
420
430
Alturas:
38
48
```

MEDAC

# Ejercicios

Ejercicios 3 Árbol DOM. Aula Virtual Tema 5

MEDAC

MEDAC

Instituto Oficial de Formación Profesional