

# Introducción a Javascript

Clase 6

# Repaso

## Consideraciones de lo aprendido sobre objetos:

- Como vimos anteriormente, Javascript es un objeto.
- Se sabe que los objetos tienen 2 cosas distintivas:
  - **Propiedades**      =>      Características / Variables comunes
  - **Métodos**           =>      Acciones / Variables de tipo función
- ¿Cómo se accede a las propiedades o ejecutamos los métodos?
  - objeto.**propiedad**
  - objeto.**metodo()**

Objeto window

# Objeto window

## ¿Qué es window?

- **window** representa al browser (navegador)
- **window** es un objeto **global**, que por tener esta condición no hace falta escribirlo
- Todo inicia en **window**
- Por ejemplo:
  - document.write('string')   => **window**.document.write('string')   // Método
  - alert('Mensaje')           => **window**.alert('mensaje')       // Método
  - var variable = 'string'      => **window**.variable = 'string'       // Propiedad

# Objeto window

## ¿Qué se puede hacer con el objeto window?

- Se pueden obtener y manipular los elementos HTML que están en el documento.
- Se pueden generar nuevos elementos que no están en el documento.
- Se pueden eliminar elementos existentes en el documento.
- Se puede interactuar con el dispositivo (eventos de teclado, mouse).
- Se puede obtener información del browser (nombre, versión, etc.).
- Las acciones que se pueden realizar con window se dividen en dos modelos:

BROWSER OBJECT MODEL (BOM)	DOCUMENT OBJECT MODEL (DOM)
<ul style="list-style-type: none"><li>• Modelo de objetos del <b><i>browser</i></b></li><li>• Es propietario</li><li>• No es estándar</li><li>• Casi está en desuso</li></ul>	<ul style="list-style-type: none"><li>• Modelo de objetos del <b><i>documento</i></b></li><li>• Se base en las etiquetas HTML</li><li>• Está casi que estandarizado</li><li>• Es el que vamos a ver a continuación :D</li></ul>

# DOM I

**Interviewer : How does DOM work?**

**ME:**

**FAMILY**

# DOM: Document Object Model

## ¿Qué es DOM?

- Es el modelo de objetos que ofrece el documento HTML
- Se encuentra estandarizado entre los diferentes browser
- Permite encontrar elementos dentro del HTML y manipular **toda** la estructura nodal del documento, por lo que permite: encontrar / crear / modificar / eliminar elementos HTML, así como atributos y eventos

## Requisitos para poder trabajar con DOM

- Al tener que manipular elementos HTML, es necesario que estén cargados a la hora de interactuar con ellos, de lo contrario, Javascript no tendrá las etiquetas con las cuales trabajar
- Para asegurarse que los elementos ya están cargados, existen dos posibilidades
  - Colocar el **script en el body** después de haber creado las etiquetas a manipular... o para mayor seguridad, **antes de cerrar la etiqueta body**.
  - Colocar el **script en el head** y meter todas las funciones de DOM dentro del **evento onload del browser**, que al ejecutarse indica que la página y sus etiquetas ya están cargadas.



# DOM: Scope o ámbito

## Ejemplos de carga del script de DOM

```
<!doctype>
<html>
  <head>...</head>
  <body>
    <!-- Etiquetas del sitio -->
    <script>
      /*
      Aquí va todo el código que
      maneje el DOM.
      */
    </script>
  </body>
</html>
```

```
<!doctype>
<html>
  <head>
    <script>
      window.addEventListener('load', () => {
        /*
        Aquí va todo el código que
        maneje el DOM.
        */
      });
    </script>
  </head>
  <body>
    <!-- Etiquetas del sitio -->
  </body>
</html>
```

# Métodos de búsqueda

# DOM: Métodos de búsqueda

## ¿Cómo buscar elementos HTML?

- Para buscar elementos en el HTML existen varios métodos que "inician" en el document:
  - document.**getElementById**
    - Busca por el atributo **id**. Devuelve un **HTML Element** (un objeto)
  - document.**getElementsByTagName**
    - Busca por **nombre de etiqueta**. Devuelve una **HTML Collection** (array de objetos)
  - document.**getElementsByName**
    - Busca por el atributo **name**. Devuelve una **HTML Collection**
  - document.**getElementsByClassName**
    - Busca por el atributo **class**. Devuelve una **HTML Collection**
  - document.**querySelector**
    - Busca utilizando sintaxis **de CSS**. Devuelve el **primer HTML Element**
  - **document**.querySelectorAll
    - Busca utilizando sintaxis de **CSS**. Devuelve una **HTML Collection**

# DOM: Métodos de búsqueda

`document.getElementById(id)`

- Este método tiene un parámetro que recibe el id de la etiqueta a buscar
- Como el id es único, devuelve en forma de objeto el elemento del HTML buscado
- De este elemento se puede acceder por notación de puntos como si de un objeto se tratara
  - A cualquier **atributo** como las **propiedades** del objeto
  - A cualquier **evento** como los **métodos** del objeto.

`document.getElementsByTagName('etiqueta')`

- Se utiliza para buscar elementos HTML por su nombre de etiqueta
- Devuelve un array de elementos HTML correspondientes a dicha etiqueta
- Como el retorno es un array, se lo puede recorrer y además aplicarles métodos de arrays
- Siempre retorna un array (por más que tenga un solo elemento o no encuentre ningún elemento)
- Este método puede ser ejecutado por el **document** o desde cualquier otro elemento HTML
- En cada índice del array hay un elemento HTML, por lo que se podrá acceder a sus atributos (para lectura y escritura) y a eventos.

# DOM: Métodos de búsqueda

`document.getElementsByClassName('clase o clases')`

- Se utiliza para buscar elementos HTML por su nombre de clase o clases.
- En el caso de que sean más de una clase, no importa el orden.
- Devuelve un array de elementos HTML correspondientes a dicha clase o clases.
- Como el retorno es un array, se lo puede recorrer y además aplicarles métodos de arrays.
- Siempre retorna un array (por más que tenga un solo elemento o no encuentre ningún elemento).
- Este método puede ser ejecutado por el document o desde cualquier otro elemento HTML.
- En cada índice del array hay un elemento HTML, por lo que se podrá acceder a sus atributos (para lectura y escritura) y a eventos.

# DOM: Métodos de búsqueda

`document.querySelector('css query')`

- Se utiliza para buscar un elemento HTML por una consulta css.
- Devuelve la primera aparición del nodo HTML correspondiente a dicha consulta.
- Si no encuentra ninguna etiqueta, devolverá null.
- Este método puede ser ejecutado por el document o desde un nodo padre HTML desde donde se iniciará la búsqueda.
- Del elemento HTML encontrado se podrá acceder a sus atributos (para lectura y escritura) y a eventos.
- Se debe tener en cuenta que hace una búsqueda estática (solo elementos que ya estén en el DOM al hacer la búsqueda).

# DOM: Métodos de búsqueda

`document.querySelectorAll('css query')`

- Se utiliza para buscar elementos HTML por una consulta css.
- Devuelve un array de nodos HTML correspondientes a dicha consulta.
- Como el retorno es un array, se lo puede recorrer y además aplicarles métodos de arrays.
- Siempre retorna un array (por más que tenga un solo elemento o no encuentre ningún elemento).
- Este método puede ser ejecutado por el document o desde un nodo padre HTML desde donde se iniciará la búsqueda.
- En cada índice del array hay un elemento HTML, por lo que se podrá acceder a sus atributos (para lectura y escritura) y a eventos.

# DOM: Métodos de búsqueda

## Repaso de búsqueda con CSS

- Por nombre de etiqueta: ***etiqueta***
- Por clase: ***.clase***
- Por id: ***#id***
- Todos los elementos: ***\****
- Conjunto de etiquetas: ***etiqueta1, etiqueta2***
- Hijos
  - ***nth-of-type()***: busca desde su padre según el tipo de etiqueta
  - ***nth-of-child()***: busca desde su padre sin importar el tipo de etiqueta.
- Atributos
  - Solo por atributo: ***elemento[atributo]***
  - Por atributo y su valor: ***elemento[atributo=valor].***
- Herencia
  - Directa: ***>***
  - Elemento siguiente: ***+***
  - Elemento anterior: ***~***



Atributos y contenido interno

# DOM: Atributos y contenido interno

etiqueta.*atributo*

- Obtenido un **HTML Element**, se puede pedir cualquier atributo de la etiqueta que representa.
- Los mismos atributos que se escriben en HTML, en Javascript son las propiedades del objeto.
- El nombre de todos los atributos es igual en Javascript, excepto:
  - El atributo **for** de HTML, en Javascript será la propiedad **htmlFor**
  - El atributo **class** de HTML, en Javascript será la propiedad **className**
  - El atributo **data-algo** de HTML, en Javascript será la propiedad **dataset.algo**.
- Son de lectura y escritura, accediendo por notación de puntos (son objetos)
- Se puede acceder a los atributos de una etiqueta con **etiqueta.attributes**

# DOM: Atributos y contenido interno

etiqueta.*innerHTML*

- Es una propiedad que solamente tienen las etiquetas que abren y cierran: **<tag></tag>**
- No existe en: **<img /> <input /> <area /> <br /> <hr />**
- Es el contenido entre **<>** y **</>** de la etiqueta, ya sea texto solamente u otros elementos html.
- Es de lectura y de escritura. Si el contenido es HTML, se renderiza
- Con **=** se reemplaza el contenido y con **+=** se concatena el nuevo contenido al ya existente

Propiedades de estilo

# DOM: Propiedades de estilo

etiqueta.*style*

- Las propiedades de CSS se acceden desde la propiedad `style` del objeto HTML, que también es un objeto
- Las subpropiedades ***etiqueta.style.subpropiedad*** se pueden acceder con notación de puntos o como posiciones asociativas
  - ***etiqueta.style.propiedadCSS***
    - Las propiedades con guión medio, se deben escribir quitando el guión y colocando la letra que le sigue en mayúscula
  - ***etiqueta.style['propiedad-css']***
    - Se pueden usar guiones o la regla anterior.

Posiciones asociativas	Notación de puntos
<code>etiqueta.style['color'] = 'red';</code> <code>etiqueta.style['background-color'] = 'blue';</code>	<code>etiqueta.style.color = 'red';</code> <code>etiqueta.style.backgroundColor = 'blue';</code>

# Observaciones

# DOM: Observaciones

## Uso de notación de puntos o de posición asociativa

- Con los conocimientos adquiridos de objetos, sabiendo que los objetos no se pueden duplicar y la posibilidad de acceder a una propiedad o método mediante corchetes (arrays asociativos), podríamos reducir la escritura de los métodos de DOM, por ejemplo en `querySelector`

- ```
const d = document; // Se puede guardar el objeto document en una variable
const qS = 'querySelector'; // Se puede guardar el nombre del método en una variable.
```

Se puede entonces usar el método de varias maneras:

```
let resultadoBusqueda1 = document.querySelector('css query');
let resultadoBusqueda2 = d.querySelector('css query');
let resultadoBusqueda3 = d['querySelector']('css query');
let resultadoBusqueda4 = d[qS]('css query');
```

# Eventos



**EVENTS...**

**EVENTS EVERYWHERE**

# DOM: Eventos

## Los eventos HTML pasan a ser los nombres de los métodos Javascript

- Como sucede con los atributos de HTML, que se convierten en propiedades en Javascript, los eventos serían los métodos del objeto HTML en Javascript
- Para que sean métodos se les debe asignar una función, que se ejecutará cuando el evento se dispare
- Las funciones asignadas a los métodos que representan a los eventos, reciben el nombre de **"Escuchadores de Eventos"** o **"Event Listener"**
- Dicho esto, la forma correcta de trabajar es mediante el método `addEventListener`.

## Agregar escuchadores de eventos

- Los objetos pueden ejecutar el método **`addEventListener`**
  - **add** => agregar
  - **event** => evento
  - **listener** => escuchador
- Permite agregar más de un listener a un mismo evento de un mismo elemento

# DOM: Sintaxis de Eventos

`objeto.addEventListener('evento', función );`

evento => el evento a escuchar (string, sin el on).

función => la función a ejecutar al escuchar el evento.

## Ejemplo

```
document.querySelector('button'). addEventListener('click', (e) => {
```

```
    // Código que procesa la función.
```

```
});
```

- El parámetro e recibe información del evento que la disparó. Es el único parámetro

# DOM II

# Métodos de atributo

# DOM: Métodos de atributo

En Javascript existen cuatro métodos para manipular atributos de etiquetas

- Además de manipularlos como propiedades (etiqueta.atributo), existen los métodos
  - etiqueta.**getAttribute('atributo')**
    - Obtiene un atributo.
  - etiqueta.**setAttribute('atributo', 'valor')**
    - Crea o modifica un atributo.
  - etiqueta.**hasAttribute('atributo')**
    - Informa si el atributo existe o no.
  - etiqueta.**removeAttribute('atributo')**
    - Remueve un atributo.

# DOM: Métodos de atributo

etiqueta.*getAttribute('atributo')*

- Devuelve el valor del atributo (string) pasado como argumento
- Si el atributo no existe, devolverá null (o cadena vacía para algunas versiones de navegadores)
- Veamos un ejemplo para mayor comprensión ([ejemplo-1-0.html](#))

# DOM: Métodos de atributo

etiqueta.***setAttribute('atributo', 'valor')***

- Si el atributo recibido en el primer argumento (***string***) no existe, lo crea y asigna el valor recibido en el segundo argumento (***string***). Si ya existe, lo modifica.
- Veamos un ejemplo para mayor comprensión ([\*\*\*ejemplo-1-1.html\*\*\*](#))

## Seteando el atributo style

- Al utilizar este método con el atributo style, se debe tener presente que creará o modificará
- Veamos un ejemplo para mayor comprensión ([\*\*\*ejemplo-1-2.html\*\*\*](#))



# DOM: Métodos de atributo

etiqueta.***hasAttribute('atributo')***

- Si el atributo recibido en el argumento (***string***) existe, devuelve ***true***. Si no existe, ***false***.
- Veamos un ejemplo para mayor comprensión ([\*\*\*ejemplo-1-3.html\*\*\*](#))

etiqueta.***removeAttribute('atributo')***

- Remueve (si existe) el atributo recibido en el argumento (string)
- Veamos un ejemplo para mayor comprensión ([\*\*\*ejemplo-1-4.html\*\*\*](#))

# DOM: Texto de estilo

etiqueta.***style.cssText***

- Además de las sub propiedades de la propiedad style, existe una que permite pasarle una cadena completa de estilos, la sub propiedad cssText.
- Al igual que cualquier propiedad, es de lectura y escritura.
- Devuelve o recibe una cadena de texto que representa los estilos del elemento.
- Veamos un ejemplo para mayor comprensión ([ejemplo-2-0.html](#))

# Nodos

# DOM: Nodos

## Propiedades "familiares" para moverse entre nodos

- Por el momento hemos buscado etiquetas mediante su id.
- Existen propiedades que representan relaciones "familiares" que permitirán partir de una etiqueta en particular y llegar a otra u otras, por más que no tengan id.

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Título</h1>
```

```
    <div>
```

```
      <p>Párrafo</p>
```

```
      <p id="p">Otro párrafo</p>
```

```
      <p>Otro párrafo más</p>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

```
// La etiqueta padre del padre
```

```
// La etiqueta hermana previa del padre
```

```
// La etiqueta padre
```

```
// La etiqueta hermana previa
```

```
// Buscamos el párrafo
```

```
// La etiqueta hermana siguiente
```

# DOM: Nodos

## Propiedades "familiares" para moverse entre nodos

- Por el momento hemos buscado etiquetas mediante su id.
- Existen propiedades que representan relaciones "familiares" que permitirán partir de una etiqueta en particular y llegar a otra u otras, por más que no tengan id.

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Título</h1>
```

```
    <div>
```

```
      <p>Párrafo</p>
```

```
      <p id="p">Otro párrafo</p>
```

```
      <p>Otro párrafo más</p>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

```
let abuelo = p.parentNode.parentNode;
```

```
let tio = p.parentNode.previousElementSibling;
```

```
let padre = p.parentNode;
```

```
let hermanoMayor = p.previousElementSibling;
```

```
let p = document.getElementById('p');
```

```
let hermanoMenor = p.nextElementSibling;
```

# DOM: Nodos

## Propiedades "familiares" para moverse entre nodos

- Las propiedades recién vistas permiten buscar otras etiquetas a partir de una ya obtenida
  - **parentNode**: devuelve el nodo padre.
  - **previousSibling**: devuelve el nodo hermano anterior (no importa si es nodo html, nodo de texto o nodo de comentario).
  - **previousElementSibling**: devuelve el nodo html hermano anterior.
  - **nextSibling**: devuelve el nodo hermano siguiente (no importa si es nodo html, nodo de texto o nodo de comentario).
  - **nextElementSibling**: devuelve el nodo html hermano siguiente
- Si no encuentra "hermano" o "padre", devolverá null.
- Como también se observó, se pueden ir encadenando las búsquedas, mediante los puntos.
- Por ejemplo, ¿Cómo se debería haber buscado a la etiqueta head a partir del párrafo?
  - `let tioAbuelo = p.parentNode.parentNode.previousElementSibling;`
- Aclaración:
  - Los textos dentro de las etiquetas, así como los espacios, saltos de línea o tabulaciones entre ellas, se consideran **nodos de texto** (textNodes).

# DOM: Nodos

## Propiedades "familiares" para moverse entre nodos

- Además, se pueden buscar a los nodos internos de una etiqueta, o "hijos"
  - **firstChild**: devuelve el primer nodo de la etiqueta contenedora (nodo html, nodo de texto o nodo de comentario).
  - **firstElementChild**: devuelve el primer nodo html de la etiqueta contenedora.
  - **lastChild**: devuelve el último nodo de la etiqueta contenedora (nodo html, nodo de texto o nodo de comentario).
  - **lastElementChild**: devuelve el último nodo html de la etiqueta contenedora.
  - **childNodes**: devuelve un array con todos los nodos de la etiqueta contenedora (nodos html, nodos de texto o nodos de comentario).
  - **children**: devuelve un array con todos los nodos html de la etiqueta contenedora.
- Veamos un ejemplo para mayor comprensión ([ejemplo-2-1.html](#))

DOM III





# Manipulación de etiquetas

# DOM: Métodos de atributo

## Crear, agregar, reemplazar, remover y clonar nodos

- Además de los métodos de búsqueda, existen otros que permiten su manipulación
  - `document.createElement('etiqueta')`: crea una etiqueta.
  - `document.createTextNode('texto')`: crea un nodo de texto.
  - `document.createComment('comentario')`: crea un nodo de comentario.
  - `nodoPadre.appendChild(nodoHijo)`: agrega un nodo hijo a su padre.
  - `padre.append(...nodoHijo|cadena DOM)`: agrega un nodo hijo o cadena DOM al final del padre.
  - `padre.prepend(...nodoHijo|cadena DOM)`: agrega un nodo hijo o cadena DOM al inicio del padre.
  - `etiqueta.before(...nodo|cadena DOM)`: agrega un nodo o cadena DOM antes de la etiqueta.
  - `etiqueta.after(...nodo|cadena DOM)`: agrega un nodo o cadena DOM después de la etiqueta.
  - `nodoPadre.replaceChild(nuevoNodo, nodoHijo)`: reemplaza un nodo hijo a su padre.
  - `etiqueta.replaceChildren(...nuevosNodos)`: reemplaza los nodos de la etiqueta por nuevos nodos.
  - `nodoPadre.removeChild(nodoHijo)`: elimina un nodo hijo a su padre.
  - `nodo.remove()`: elimina un nodo.
  - `nodo.cloneNode()`: clona un nodo.

# DOM: Manipulación de etiquetas

`document.createElement('etiqueta')`

- Se utiliza para crear un elemento HTML por su nombre de etiqueta
- Devuelve un HTML element del tipo del nombre etiqueta recibido en su parámetro.
- Este método solamente puede ser ejecutado por el document.
- Del elemento HTML creado se podrá acceder a sus atributos y a eventos.

`document.createTextNode('texto')`

- Se utiliza para crear un nodo de texto.
- Devuelve un Text Node con el contenido recibido en su parámetro.
- Este método solamente puede ser ejecutado por el document.

`document.createComment('comentario')`

- Se utiliza para crear un nodo de comentario.
- Devuelve un Comment Node con el contenido recibido en su parámetro.
- Este método solamente puede ser ejecutado por el document.

# DOM: Manipulación de etiquetas

`nodoPadre.appendChild(nodoHijo)`

- Se utiliza para agregar un nodo hijo al final de un nodo padre.
- Este método solamente puede ser ejecutado por el nodo padre.

`document.append(...nodo/s|cadena/s DOM):`

- Se utiliza para agregar nodo/s o cadena/s DOM al final de la etiqueta contenedora.

`document.prepend(...nodo/s|cadena/s DOM):`

- Se utiliza para agregar nodo/s o cadena/s DOM al inicio de la etiqueta contenedora.

# DOM: Manipulación de etiquetas

`nodoPadre.insertBefore(nuevoNodo, nodoHijo)`

- Se utiliza para agregar un nuevo nodo antes de otro nodo hijo de un nodo padre.
- Este método solamente puede ser ejecutado por el nodo padre.

`nodoPadre.before(...nodo/s|cadena/s DOM)`

- Se utiliza para agregar nodo/s o cadena/s DOM antes de la etiqueta.

`nodoPadre.after(...nodo/s|cadena/s DOM)`

- Se utiliza para agregar nodo/s o cadena/s DOM después de la etiqueta.

# DOM: Manipulación de etiquetas

`nodoPadre.replaceChild(nuevoNodo, nodoHijo)`

- Se utiliza para reemplazar un nuevo nodo por otro nodo hijo de un nodo padre.
- Devuelve el nodo reemplazado.
- Este método solamente puede ser ejecutado por el nodo padre.

`nodo.replaceChildren(...nodo/s|cadena/s DOM)`

- Se utiliza para reemplazar el contenido de una etiqueta por nodo/s o cadena/s DOM.
- Si no se le pasa parámetro, elimina el contenido de la etiqueta.

# DOM: Manipulación de etiquetas

`nodoPadre.removeChild(nodoHijo)`

- Se utiliza para remover un nodo hijo de un nodo padre.
- Devuelve el nodo removido.
- Este método solamente puede ser ejecutado por el nodo padre.

`nodo.remove()`

- Se utiliza para remover un nodo hijo de un nodo padre.
- Este método solamente puede ser ejecutado por el nodo.

`node.cloneNode(conContenido)`

- Se utiliza para clonar un nodo. Tiene como parámetro opcional la opción de :
  - `true`: clonar la etiqueta, su atributos y todo su contenido.
  - `false` (por defecto): clonar solamente la etiqueta y sus atributos.
- Devuelve el nodo clonado.
- Este método solamente puede ser ejecutado por el nodo.



# Fin de la clase

Este es el espacio para dudas y/o preguntas existenciales