

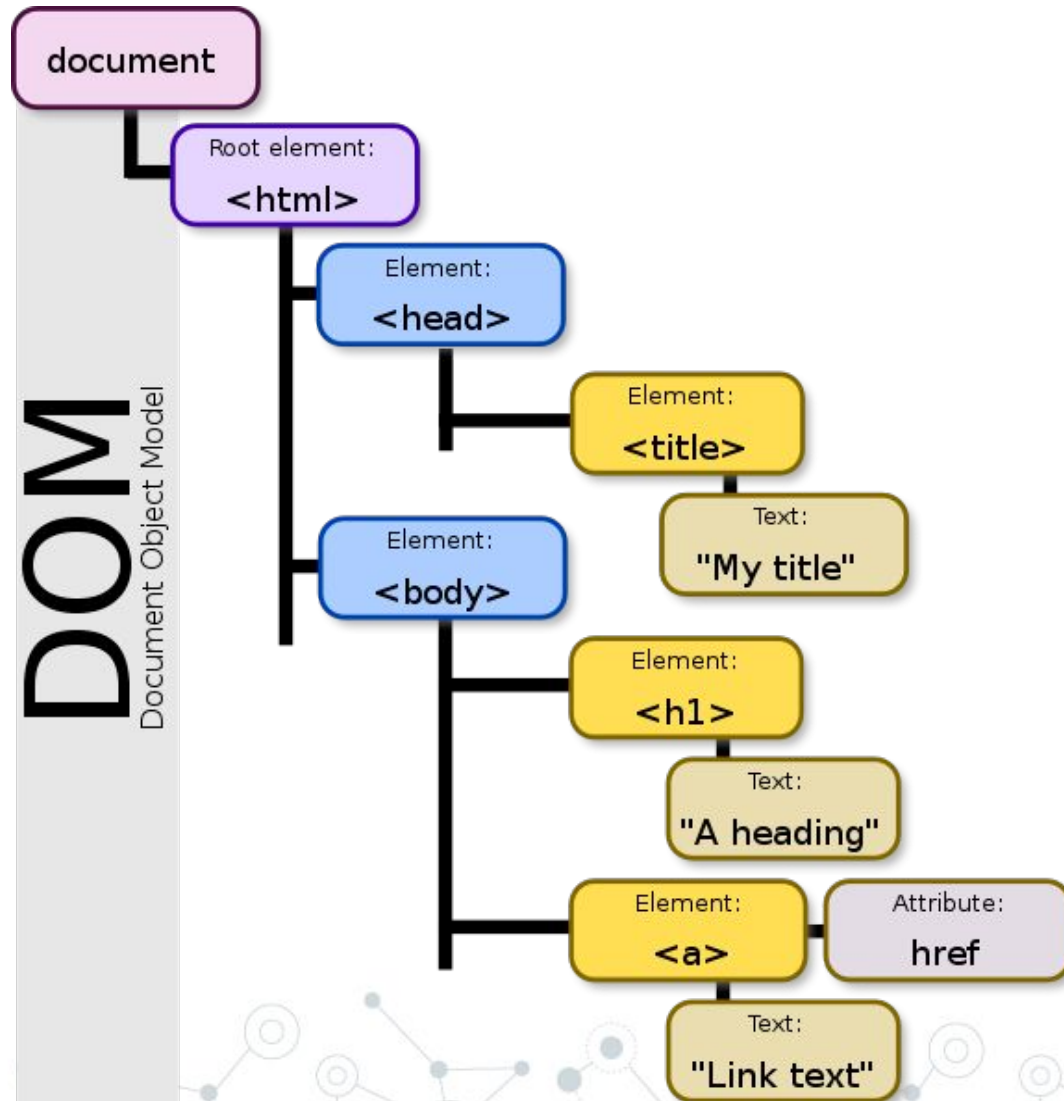


Universidad
Nacional
de Quilmes

DOM Manipulation

CONSTRUCCIÓN DE
INTERFACES DE USUARIO

Document Object Model



Acerca del DOM

- ⊙ Es una interfaz independiente de lenguajes.
- ⊙ Trata a un documento HTML/XML como una estructura de árbol, donde cada nodo es un objeto.
- ⊙ Los métodos que expone permiten el acceso al árbol.
- ⊙ Se puede cambiar la estructura, el estilo o el contenido de un documento.
- ⊙ Los nodos pueden tener *handlers* de eventos para permitir acciones cuando sucede dicho evento.
- ⊙ Se normalizó en 2004 por la World Wide Web Consortium
- ⊙ Previamente cada *browser* tenía su especificación

DOM Manipulation

- ◎ Formas de manipular el DOM
 - Con los *DevTools* de los browsers
 - Ya sea “a mano” o con scripting
 - Muy útil para probar cosas
 - Con Javascript
 - Agregando código en el documento
 - Es la forma que más nos va a interesar

Qué puede hacer JS

- ◎ Con respecto al DOM, JavaScript puede:
 - Crear, modificar y eliminar **elementos** HTML
 - Crear, modificar y eliminar **atributos** de cualquier elemento HTML
 - Agregar, modificar y eliminar **estilos** CSS
 - Reaccionar a cualquier **evento** de los elementos HTML, tanto existentes como agregados dinámicamente

Hello DOM

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Hello DOM</h1>
    <div id="hello"></div>
    <script>
      document
        .getElementById("hello")
        .innerHTML = "Welcome to the Jungle!";
    </script>
  </body>
</html>
```

Hello DOM

Welcome to the Jungle!

DOM document

© El DOM provee la variable global document

- Es el objeto que contiene toda la página web
- Para acceder a cualquier elemento hay que pedirselo al document

© Obteniendo elementos:

- `document.getElementById(id)`
- `document.getElementsByTagName(name)`
- `document.getElementsByClassName(name)`
- `document.querySelector(cssSelector)`
- `document.querySelectorAll(cssSelector)`
 - Ej: `"div.container"`

Manipulando Elementos

- ◎ `element = document.getElementById("foo")`
 - `element.innerHTML = new content`
 - `element.attribute = new value`
 - `element.style.property = new style`
- ◎ `document.createElement(element)`
- ◎ `document.removeChild(element)`
- ◎ `document.appendChild(element)`
- ◎ `document.replaceChild(new, old)`
- ◎ `document.write(text)`

Events » onClick

```
<h3><span id="count">0</span>Likes</h3>
```

```
<div>
```

```
  <button type="button" id="add">👍</button>
```

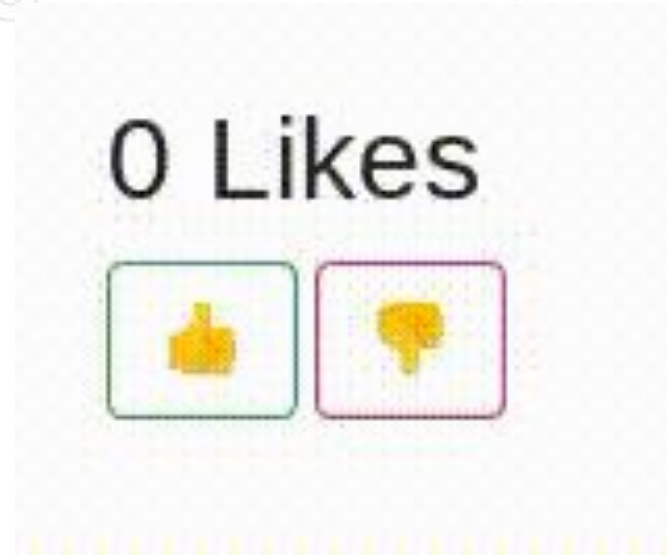
```
  <button type="button" id="sub">👎</button>
```

```
</div>
```

```
<script>
```

```
document.getElementById("add").onclick = () => {  
  let counter = document.getElementById("count");  
  counter.innerHTML = Number(counter.innerHTML) + 1;  
};
```

```
document.getElementById("sub").onclick = () => {  
  let counter = document.getElementById("count");  
  counter.innerHTML = Number(counter.innerHTML) - 1;  
};  
</script>
```



Clocking JavaScript

Existen dos funciones importantes para manejar tiempos de espera e intervalos:

- ⦿ **setTimeout(*function*, *milliseconds*)**
 - Ejecuta la función luego de determinado período de tiempo
 - Solo se ejecuta **1 vez**
- ⦿ **setInterval(*function*, *milliseconds*)**
 - Ejecuta la función en los intervalos de tiempo correspondientes
 - Solo se ejecuta **indefinidamente**
 - Se puede cancelar con **clearInterval()**

Clock » setInterval

```
<h4 id="date"></h4>
```

20/10/2019, 20:59:11

```
<script>
```

```
// Obtengo el elemento
```

```
let dateElem = document.getElementById("date");
```

```
// Inicializo la fecha
```

```
dateElem.innerHTML = new Date().toLocaleString();
```

```
// Refresco cada 1 segundo
```

```
setInterval(() => {
```

```
    dateElem.innerHTML = new Date().toLocaleString();
```

```
}, 1000);
```

```
</script>
```

Clock » setTimeout

```
<h4 id="elem">
  <button type="button" id="btn" onclick="showMessage()">
    Ver mensaje
  </button>
</h4>
```



```
<script>
function showMessage() {
  document.querySelector("#btn").innerHTML = "⌚";
  setTimeout(() => {
    let txt = "Sin televisión y sin cerveza Homero pierde la cabeza";
    document.querySelector("#elem").innerHTML = txt;
  }, 3000);
}
</script>
```

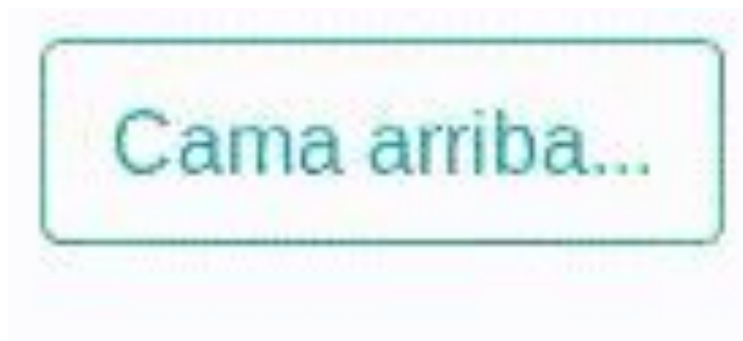
Eventos comunes

- 🎯 **onChange**
 - Cuando se produjo un cambio en un elemento HTML
- 🎯 **onClick**
 - Cuando el usuario hace *click* sobre el elemento
- 🎯 **onMouseOver**
 - Cuando el usuario pasa el *mouse* sobre el elemento
- 🎯 **onMouseOut**
 - Cuando el usuario mueve el *mouse* fuera del elemento
- 🎯 **onKeyDown**
 - Cuando el usuario pulsa una tecla
- 🎯 **onLoad**
 - Cuando el browser termina de cargar el documento

Events » onmouseover » onmouseout

```
<button id="btn" class="btn btn-outline-info" type="button">  
  Cama arriba...  
</button>
```

```
<script>  
  let btn = document.querySelector("#btn");  
  
  btn.onmouseout = (event) => {  
    event.target.innerHTML = "Cama arriba...";  
  };  
  btn.onmouseover = (event) => {  
    event.target.innerHTML = "Cama abajo...";  
  }  
</script>
```



Manipulando elementos

```
<div id="info">  
  <p id="p1">Primer párrafo.</p>  
  <p id="p2">Segundo párrafo.</p>  
</div>
```

Primer párrafo.

Tercer párrafo.

```
<script>  
  let info = document.getElementById("info");  
  let p3 = document.createElement("p");  
  p3.innerHTML = "Tercer párrafo.";  
  
  // Agrego el nuevo elemento  
  info.appendChild(p3);  
  // Elimino el elemento viejo  
  document.getElementById("p2").remove();  
</script>
```

JS++

- © Hay muchísimo más acerca del manejo del DOM desde JavaScript...
- © Entren a https://www.w3schools.com/js/js_htmlDOM.asp
- © Pero es bastante engorroso ser ágil y dinámico porque la interfaz es muy burocrática.
- © Por estas cosas surgen frameworks y librerías como Angular, React, Vue...
- © Para agilizar la manipulación de elementos

