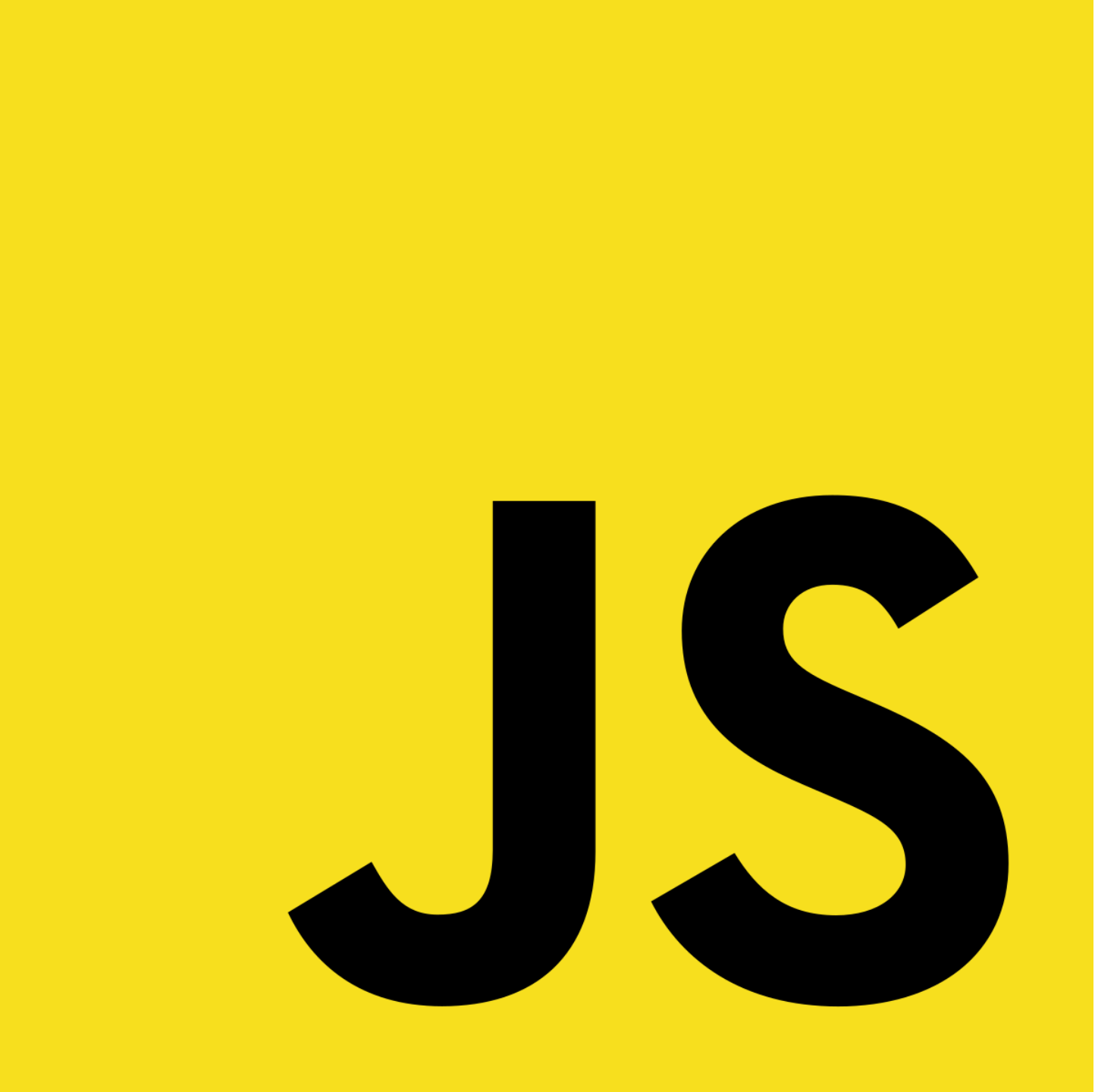
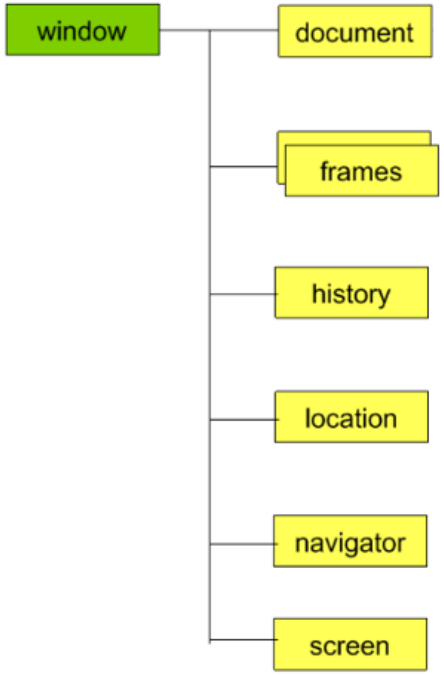


JavaScript

DOM

Modelo de Objetos

El modelo de objetos de navegador (*Browser Object Model*) hace referencia al conjunto de objetos que dependen del navegador del usuario.



Estos objetos permiten obtener y manipular información tanto del navegador como el contenido de la propia web.

Modelo de objetos del navegador (BOM)

- ***window.document***: Hace referencia al documento DOM cargado en la ventana. Es la forma principal de interactuar con el contenido HTML.
- ***window.location***: Proporciona acceso a la URL actual de la ventana, y permite redireccionar a otras URLs.
- ***window.navigator***: Ofrece información sobre el navegador, como el agente de usuario, la plataforma, la conectividad y más.
- ***window.history***: Permite interactuar con el historial de navegación, como retroceder o avanzar en la historia del navegador.
- ***window.screen***: Proporciona información sobre la pantalla física, como las dimensiones y la resolución.

Modelo de objetos del navegador (BOM)

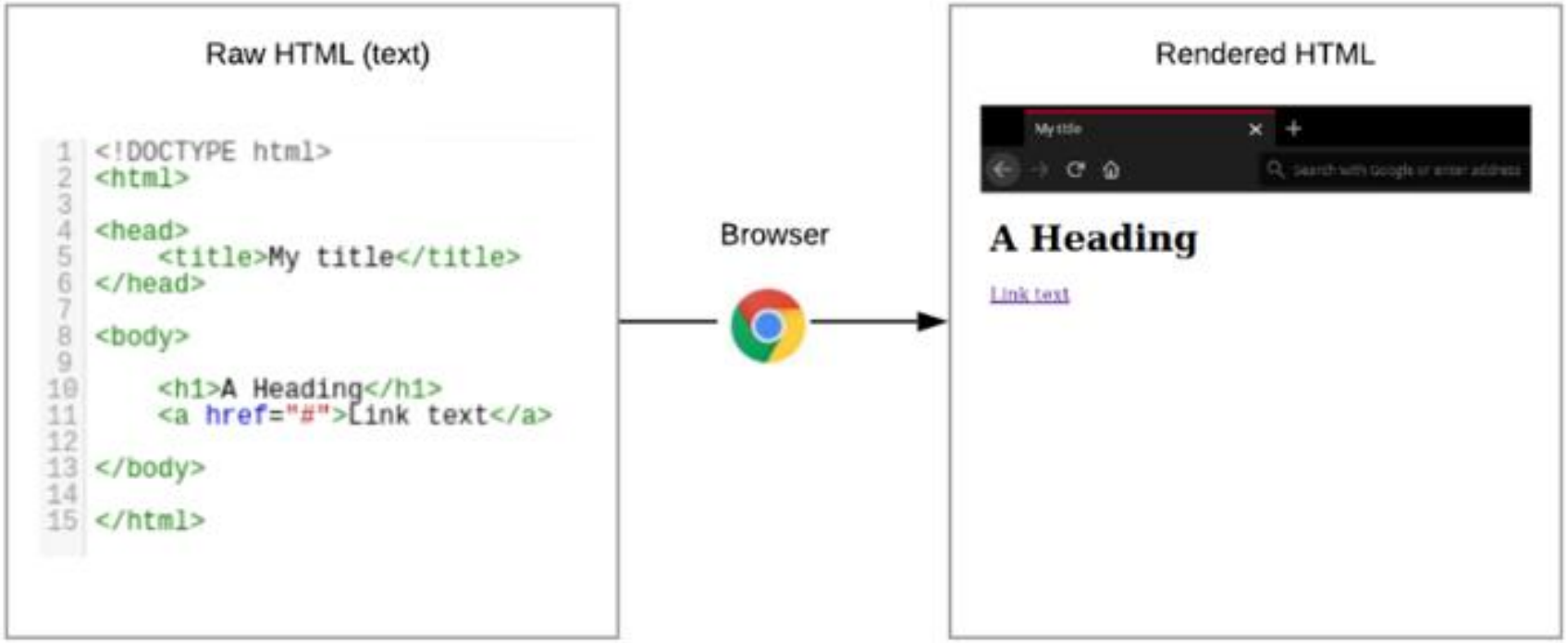
El objeto **location** representa la URL actual del documento cargado en el navegador. Este objeto permite obtener, manipular y cambiar la URL de la página actual.

- *location.href:* Contiene la URL completa de la página actual. También puedes asignarle una nueva URL para redirigir el navegador.
- *location.hostname:* Muestra el nombre del dominio.
- *location.pathname:* Devuelve la ruta después del dominio.
- *location.search:* Devuelve la cadena de consulta (querystring) con los datos concatenados a la URL en caso de existir.

```
document.write("<br>URL actual: " + location.href);
document.write("<br>Dominio: " + location.hostname);
document.write("<br>Ruta: " + location.pathname);
document.write("<br>Cadena de consulta: " + location.search);
```

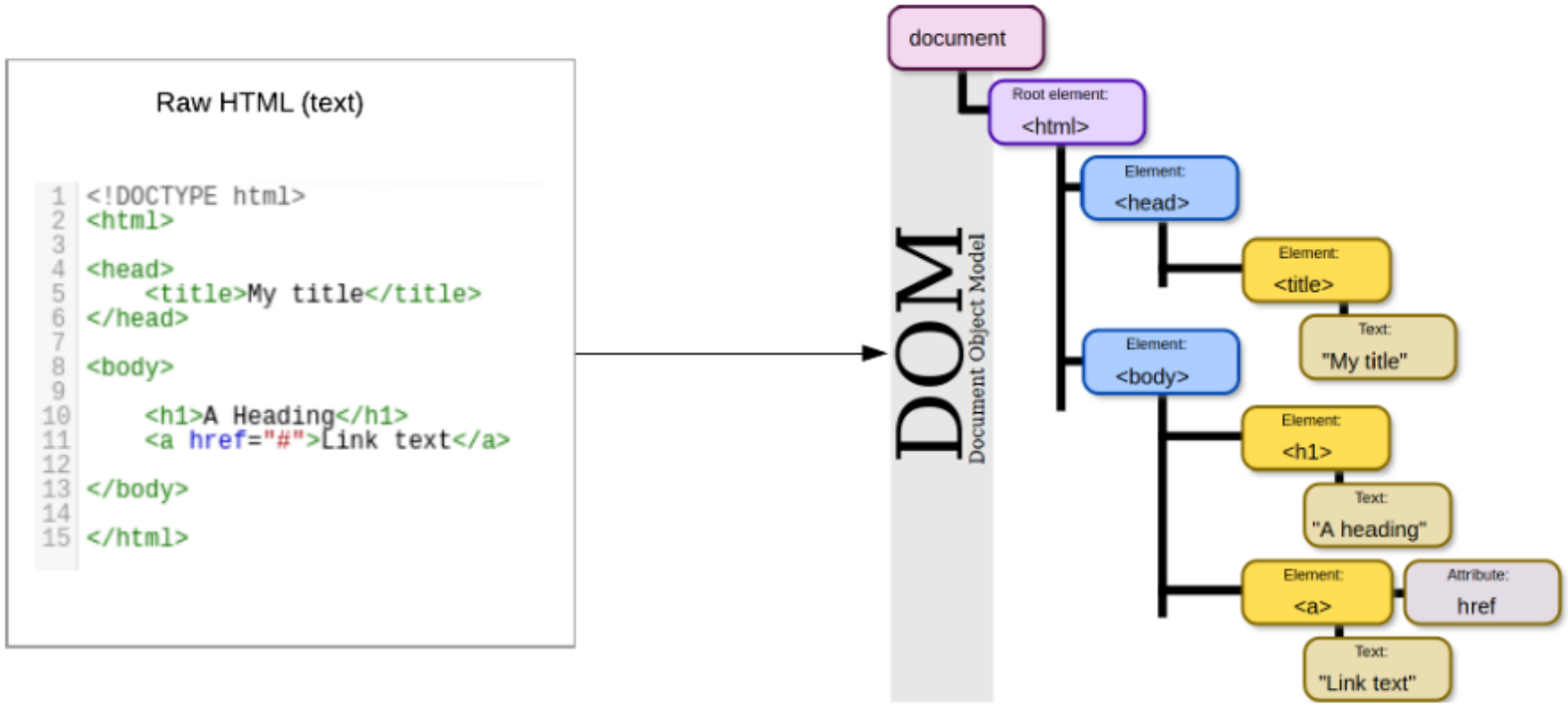
Modelo de objetos del documento (DOM)

El objeto **document** representa el *modelo de objetos de documento* de la página web actual cargada. Permite acceder o modificar el contenido de la página.



Modelo de objetos del documento (DOM)

El modelo de objetos de documento (*Document Object Model*) es un árbol de objetos que representan cada elemento de la web mostrada por el navegador.



Modelo de objetos del documento (DOM)

Cada elemento HTML de la web es un objeto dentro del DOM con sus propiedades, y con Javascript podemos realizar las siguientes operaciones:

- **Navegación/Búsqueda** → Consiste en recorrer y obtener referencias a los objetos de las web sobre los que queremos actuar.

- **Manipulación** → Consiste en modificar el valor de sus atributos y contenido para modificar tanto el aspecto como contenido de la web una vez cargada.

- **Creación/Eliminación** → Crear y eliminar elementos para alterar la página agregando y suprimiendo contenido al margen de su estructura y contenidos originales.

Eventos

Un **evento** es un mecanismo que permite detectar un suceso sobre algún elemento de la página web.

Ej: Pulsar un botón

Una **función manejadora** es aquella que se ejecuta cuando se produce un evento para realizar la tarea asociada al mismo.

Ej: Realizar una validación/cálculo…, etc.

Selección de elementos del DOM

El objeto ***document*** incluye distintos métodos para obtener las referencias de uno o más objetos de la página según sus características.

- ***getElementById()*** → Retorna la referencia al objeto de la página con el valor indicado en su atributo id o *null.*
- ***getElementsByName()*** → Selecciona todos los objetos del cuyo atributo "*name*" tiene el valor indicado.
- ***getElementsByTagName()*** → Selecciona todos los objetos de la página cuya etiqueta HTML es la indicada como argumento tal como "*div*", "*p*", "*img*"…, etc.
- ***getElementsByClassName()*** → Selecciona todos los objetos con la hoja de estilos indicada asociada.

Selección de elementos del DOM

Los siguientes métodos permiten seleccionar los objetos del DOM que cumplen con el selector CSS indicado como argumento:

- Método ***querySelector()*** → Selecciona <u>únicamente el primer objeto</u> de la página que se ajuste al selector CSS indicado como argumento, o *null* si no se encuentra ninguno
- Método ***querySelectorAll()*** → Selecciona <u>todos los objetos que se ajusten al selector CSS</u> indicado retornando una colección de objetos, vacía si no se encuentra ninguno.

```
// Seleccionar todos los elementos con la clase 'miClase'
let elementos = document.querySelectorAll('.miClase');

// Seleccionar todos los <p> dentro de un <div> con la clase 'contenedor'
let parrafos = document.querySelectorAll('.contenedor p');

// Seleccionar todos los campos de entrada de tipo "text"
let inputsTexto = document.querySelectorAll('input[type="text"]');

// Seleccionar todos los <li> de una lista
let items = document.querySelectorAll('ul li');
```

Todos retornan una ***matriz de objetos***

Para acceder a todos los objetos seleccionados es preciso recorrer el *array* mediante la función ***forEach()***

El evento *Load*

La finalización de la carga se detecta con el evento ***load*** del objeto *window*:

```
window.onload = function() {
    // codigo a ejecutarse tras completar
    // carga de la página
}
```

Manejo de eventos sin HTML

La función manejadora puede declararse como una ***función anónima*** asociada al propio evento:

evento.js

```
window.onload = function() {
    // Obtencion de referencia del parrafo
    let parrafo = document.querySelector('#parrafo');
    // Obtencion de la referencia al boton
    let boton = document.getElementById('btn_saludar');
    // Asignacion de función manejadora al evento click del boton.
    boton.onclick = function() {
        parrafo.innerHTML = "HOLA DESDE JAVASCRIPT";
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="evento.js"></script>
</head>
<body>
    <p id="parrafo"></p>
    <button id="btn_saludar">SALUDAME</button>
</body>
</html>
```

Esto ahorra código cuando la función manejadora es única para el evento

Tipos de eventos

Eventos de ratón: Son los eventos que se producen por la acción del ratón:

- ***click***: Ocurre cuando se hace clic en un elemento.
- ***dblclick***: Ocurre cuando se hace doble clic en un elemento.
- ***mousedown***: Ocurre cuando se presiona un botón del mouse sobre un elemento.
- ***mouseup***: Ocurre cuando se suelta un botón del mouse sobre un elemento.
- ***mousemove***: Ocurre cuando se mueve el mouse sobre un elemento.
- ***mouseover***: Ocurre cuando el puntero del mouse se mueve sobre un elemento.
- ***mouseout***: Ocurre cuando el puntero del mouse se mueve fuera de un elemento.
- ***mouseenter***: Similar a ***mouseover***, pero no se propaga hacia los elementos secundarios.
- ***mouseleave***: Similar a ***mouseout***, pero no se propaga hacia los elementos secundarios.

Tipos de eventos

Eventos de formulario: Son eventos propios de los elementos formulario <form> y de los elementos que los conforman como las cajas de texto:

- ***submit***: Ocurre cuando un formulario es enviado.
- ***change***: Ocurre cuando el valor de un elemento de formulario cambia (por ejemplo, un <*input*> o <*select*>).
- ***input***: Ocurre cuando el valor de un campo de entrada es modificado (por ejemplo, en un <*input*> o <*textarea*>).
- ***focus***: Ocurre cuando un elemento gana el foco.
- ***blur***: Ocurre cuando un elemento pierde el foco.
- ***reset***: Ocurre cuando se reinicia un formulario.

Tipos de eventos

Eventos de teclado: Son los eventos se producen por la acción del teclado. Son detectados por elementos que tiene el foco, por ejemplo; una caja de texto:

- ***keydown***: Ocurre cuando una tecla es presionada.
- ***keypress***: Ocurre cuando una tecla es presionada y soltada (ahora se desaconseja usarlo en favor de keydown).
- ***keyup***: Ocurre cuando una tecla es soltada.

Manipulación de elementos

Una vez seleccionado uno o varios objetos podemos obtener y manipular su contenido y atributos mediante las siguientes *propiedades*.

Propiedad	Descripción
innerHTML	Obtiene o establece el contenido HTML de un elemento.
textContent	Obtiene o establece solo el contenido de texto de un elemento, sin incluir etiquetas HTML.
innerText	Similar a textContent, pero solo muestra el texto visible (considerando estilos de visibilidad).
value	Obtiene o establece el valor en elementos de formulario (<input>, <textarea>, <select>, etc.).
style	Accede y modifica estilos en línea del elemento (propiedades CSS).
className	Obtiene o establece las clases de un elemento como una sola cadena.
attributes	Devuelve una colección de todos los atributos del elemento.
src	Obtiene y modifica la referencia al archivo asociado a un elemento <src>, <script>, <iframe>…, etc.
href	Obtiene y modifica la URL de salto de un hipervínculo <a>
checked/disabled	Obtiene y establece un valor lógico para indicar si un elemento de formulario esta marcado o deshabilitado

Manipulación de propiedades

Las propiedades se invocan indicando el nombre de la variable que referencia al elemento seguido de un punto y la propiedad:

objeto.propiedad = valor

Las propiedades actúan como variables de las que podemos obtener/modificar su valor mediante una asignación

```
const parrafo = document.getElementById('parrafo');
parrafo.innerText = "HOLA";
```

Control de formularios (*submit*)

Los elementos ***<form>*** disponen del evento ***submit***.

- Este evento se dispara cuando el usuario pulsa el botón de envío y se validan todas las comprobaciones de los elementos HTML5.
- En la función manejadora pueden realizarse comprobaciones adicionales.

Si hay errores:

- Se cancela el envío de datos con el método ***preventDefault()***.

Para validar un control

- Se invoca al método ***reportValidity()*** → Devuelve un valor lógico cierto/falso mostrando mensaje de error identificativo en caso de error.