PROGRAMACIÓN CON JAVASCRIPT

BOM y DOM

Contenido

1. Introducción

Normalmente, cuando programamos en JavaScript, nos encontramos dentro del contexto de un navegador (esto no ocurre si desarrollamos con NodeJS por ejemplo).

Dentro de este contexto, tenemos algunos objetos predefinidos y funciones que podemos utilizar para interactuar con este navegador y con el documento HTML.

2. Browser Object Model (BOM)

2.1. Objeto window

El objeto window representa al navegador y es el objeto principal. Todo está contenido dentro de window (variables globales, el objeto documento, el objeto location, el objeto navigation, etc.).

El objeto window puede ser omitido accediendo a sus propiedades directamente.

```
'use strict';
// Tamaño 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);
});

//  Podemos omitir el objeto window (está implícito)
console.log(history.length); // Número de páginas del history. Lo mismo que window.history.length
```

1 Ejemplo de código JS. Objeto window

2.2. Timers

Hay dos tipos de “timeros” que podemos crear en JavaScript para ejecutar algún pedazo de código en el futuro (especificado en milisegundos), timeouts e intervalos. El primero se ejecuta sólo una vez (debemos volver a crearlo manualmente si queremos que se repita algo en el tiempo), y el segundo se repite cada X milisegundos sin cesar (o hasta que sea cancelado) .

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

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

2 Ejemplo código JS. Función setTimeout

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

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

3 Ejemplo código JS. Función clearTimeout

- **setInterval(funcion, 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;
setInterval(() => console.log(num++), 1000); // Imprime un número y lo incrementa cada segundo
```

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

```
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);
```

- **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);
}

setTimeout(multiply, 3000, 5, 7); // Después de 3 segundos imprimirá 35 (5*7)
```

4 Ejemplo código JS. setInterval, clearInterval

2.3. Objeto location

Contiene información sobre la url actual del navegador

```
console.log(location.href); // Imprime la URL actual
console.log(location.host); // Imprime el nombre del servidor (o la IP) como “localhost” 192.168.0.34
console.log(location.port); // Imprime el número del puerto (normalmente 80)
console.log(location.protocol); // Imprime el protocolo usado (http ó https)

location.reload(); // Recarga la página actual
location.assign("https://google.com"); // Carga una nueva página. El parámetro es la nueva URL
location.replace("https://google.com"); // Carga una nueva página sin guardar la actual en el objeto history
```

5 Ejemplo código JS. Objeto location

2.4. Objeto history

Para navegar a través de las páginas que hemos visitado en la pestaña actual, podemos usar el objeto **history**. Este objeto tiene métodos bastante útiles:

```
console.log(history.length); // Imprime el número de páginas almacenadas

history.back(); // Vuelve a la página anterior
history.forward(); // Va hacia la siguiente página
history.go(2); // Va dos páginas adelante (-2 iría dos páginas hacia atrás)
```

6 Ejemplo código JS. Acceso al objeto History

2.5. Diálogos

En cada navegador tenemos un conjunto de diálogos para interactuar con el usuario.

Sin embargo, éstos no son personalizables y por tanto cada navegador implementa el suyo a su modo. Por eso no es recomendable usarlos en una aplicación en producción (uniformidad).

En cambio, son una buena opción para realizar pruebas (en producción deberíamos usar diálogos construidos con HTML y CSS).

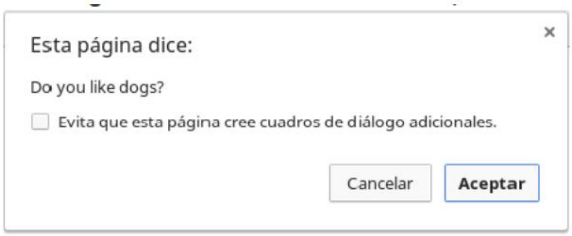
El diálogo alerta muestra un mensaje (con un botón de Aceptar) dentro de una ventana. Bloquea la ejecución de la aplicación hasta que se cierra.

```
alert("Hello everyone!");
```

7 Ejemplo código JS. Diálogo "alerto"

El diálogo confirmado es similar, pero devuelve un valor de tipo boolean. Tiene dos botones (Cancelar -> false, Aceptar -> true). El usuario elegirá entre alguna de estas dos opciones.

```
if(confirm("Do you like dogs?")) {
   console.log("You are a good person");
} else {
   console.log("You have no soul");
}
```



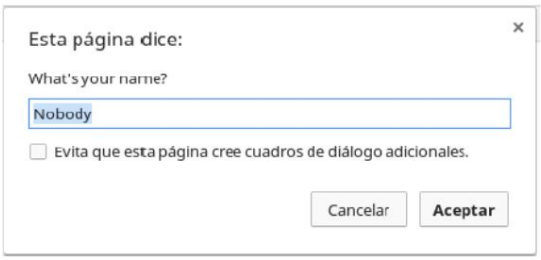
8 Ejemplo código JS. Diálogo "confirm"

El diálogo prompt muestra un input después del mensaje. Se utiliza para dar la opción al usuario de introducir algún valor, devolviendo un string con el valor introducido.

Si el usuario pulsa el botón de Cancelar o cierra el diálogo devolverá null. Se puede establecer un valor predeterminado (segundo parámetro).

```
let name = prompt("What's your name?", "Nobody");

if(name !== "null") {
   console.log("Your name is: " + name);
} else {
   console.log("You didn't answer!");
}
```



9 Ejemplo código JS. Diálogo "prompt"

3. Documento Object Model (DOM)

El Document Object Model es una estructura en árbol que contiene una representación de todos los nodos del HTML incluyendo sus atributos. En este árbol, todo se representa como nodo y podemos añadir, eliminar o modificarlos.

El objeto principal de DOM es documento. Éste es un objeto global del lenguaje.

Cada nodo HTML contenido dentro del documento es un objeto elemento, y estos elementos contienen otros nodos, atributos y estilo.

Manipular el DOM usando sólo JavaScript es algo más complicado que con la ayuda de librerías como JQuery o frameworks como Angular.

A continuación, veremos algunos métodos básicos y propiedades de los elementos del DOM en JavaScript.

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 definido el id especificado, o null si no existe

document.getElementByClassName("clase") -> Devuelve una lista (array) de elementos que tengan la clase especificada. Si la llamada a esta función se realiza desde un nodo en lugar de desde documento, la búsqueda de elementos se realizará a partir de este nodo.

document.getElementByTagName("etiqueta HTML") -> Devuelve una lista (array) con los elementos con la etiqueta HTML especificada. Por ejemplo, podemos buscar párrafos (p), títulos (h1), etc.

element.childNodes -> Vuelve un listado con los elementos descendentes (hijos) del nodo. Los nodos de tipo texto y comentarios también se obtienen en el listado.

element.children -> Vuelve un listado con los elementos descendentes (hijos) del nodo. (Excluye comentarios y etiquetas de texto)

element.parentNode -> Devuelve el elemento padre del nodo

element.nextSibling -> Vuelve el siguiente nodo del mismo nivel. Podemos utilizar nextElementSibling para obtener sólo los elementos de tipo HTML.

3.1. Selector Query

Una de las principales características que JQuery introdujo cuando se lanzó (en 2006) fue la posibilidad de acceder a los elementos HTML en base a 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.

document.querySelector("selector") -> Vuelve el primer elemento coincidente

document.querySelectorAll("selector") -> Devuelve una lista (array) con todos los elementos que coinciden con el selector.

```
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"
```

10 Ejemplos de selectores CSS que podemos utilizar para encontrar elementos

File: example1.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="./example1.js"></script>
  </body>
</html>
```

File: example1.js

```
console.log(document.querySelector("#div1 a").title); // 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 > p > a").title); // Imprime "hello world"
console.log(document.querySelector(".normalLink[title^='bye']").title); // Imprime "bye world"
console.log(document.querySelector(".normalLink[title^='bye'] + a").title); // Imprime "hello again"

let elems = document.querySelectorAll(".normalLink");
elems.forEach(function(elem) { // Imprime "hello world" y "bye world"
  console.log(elem.title);
});

let elems2 = document.querySelectorAll("a[title^='hello']"); // Atributo title empieza por "hello..."
elems2.forEach(function(elem) { // Imprime "hello world" y "hello again"
  console.log(elem.title);
});


let elems2 = document.querySelectorAll("a[title='hello world'] ~ a"); // Hermanos de <a title="hello world">
elems2.forEach(function(elem) { // Imprime "bye world" y "hello again"
  console.log(elem.title);
});
```

11 Ejemplos utilizando querySelector y querySelectorAll

3.2. Manipulando el DOM

document.createElement("etiqueta") → Crea un elemento HTML. No estará en el DOM, hasta que lo insertamos (utilizando appendChild, por ejemplo) dentro de otro elemento del DOM.

document.createTextNode("texto") → 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.

element.insertBefore(newChildElement, childElem) → Inserta un nuevo elemento hijo antes del elemento hijo recibido como segundo parámetro.

element.removeChild(childElement) → Elimina el nodo hijo que recibe por parámetro.

element.replaceChild(newChildElem, oldChildElem) → Sustituye un nodo hijo con un nuevo nodo.

3.3. Atributos

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

element.attributes → Devuelve el array con los atributos de un elemento

element.className → Se utiliza 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.classList → Array de clases CSS del elemento. A diferencia de className, que es una cadena con las clases separadas por espacio, este atributo te las devuelve en forma de array o lista. Tiene métodos muy útiles para consultar y modificar clases como:

- classList.contains(“clase”): true si tiene la clase.
- classList.replace(“clase1”,”clase2”): Quita la clase “clase1” y la sustituye por la clase "clase2".
- classList.add(“clase1”): Añade la clase “clase1” al elemento.
- classList.remove(“clase1”): Le quita la clase “clase1”.
- classList.toggle(“clase1”): Si no tiene “clase1”, lo añade. De lo contrario, la leva.

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

File: example1.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="./example1.js"></script>
  </body>
</html>
```

File: example1.js

```
let link = document.getElementById("toGoogle");
link.className = "specialLink"; // Equivale a: link.setAttribute("class", "specialLink");
link.setAttribute("href", "https://twitter.com");
link.textContent = "Twitter";
if(!link.hasAttribute("title")) { // Si no tenía el atributo title, establecemos uno
  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);
```

12 Ejemplo código JS manipulación de atributos

4. Enlaces de interés

Objeto window: https://www.w3schools.com/jsref/obj_window.asp

Objeto location: https://www.w3schools.com/jsref/obj_location.asp

Objeto documento: https://www.w3schools.com/jsref/dom_obj_document.asp

Objeto history: https://www.w3schools.com/jsref/obj_history.asp

window.navigator: https://www.w3schools.com/jsref/obj_navigator.asp

Objeto Screen: https://www.w3schools.com/jsref/obj_screen.asp

5. Bibliografía

Bernal Mayordomo, Arturo. Curso Programación con JavaScript. Cefire 2020

W3Schools - JavaScript Tutorial