Contenido

1 MANIPULACIÓN DE ATRIBUTOS

1.1 OBTENER EL VALOR DE UN ATRIBUTO

Los elementos del DOM disponen del método getAttribute para obtener el valor de un atributo concreto. Por ejemplo:

```
let lis=document.querySelectorAll(“li”);

for(let li of lis){

        console.log(li.getAttribute(“class”);

//recorre todos los elementos li y devuelve su clase
```

1.2 MODIFICAR EL VALOR DE UN ATRIBUTO

```
elemento.setAttribute(“atributo”,”nuevo_valor”);
```

1.3 ELIMINAR UN ATRIBUTO

```
elemento.remove(“atributo”,”nuevo_valor”);
```

1.4 AÑADIR Y QUITAR ATRIBUTO DE FORMA RÁPIDA

```
Elemento.toggleAttribute(“atributo”,”nuevo_valor”);
```

Añade el atributo al elemento si este no lo tiene, en caso de tenerlo lo elimina

1.5 SABER SI UN ELEMENTO TIENE ATRIBUTOS

```
Elemento.hasAttribute(“atributo”,”nuevo_valor”);
```

1.6 OBTENER TODOS LOS ATRIBUTOS DE UN ELEMENTO

La propiedad **atributes** de un elemento permite obtener todos los atributos de un elemento. Lo que devuelve es un objeto de tipo **NamedNodeList** que no es una estructura de array por lo que no tenemos acceso a todas las propiedades de la clase **array.**

Cada elemento de la lista que devuelve esta propiedad posee una propiedad llamada **name** que contiene el nombre del atributo y **value** que contiene su valor. Por ejemplo:

```
<form action="">

    <label for="edad">Escriba su edad:</label>

    <input type="number" id="edad" name="edad" min="18" max="65">

</form>

<script>

/*Navegar y mostrar atributos*/



    let listaAtributos=document.getElementById("edad").attributes;

    for (let atributo of listaAtributos) {

        console.log(`Atributo: ${atributo.name} Valor:
${atributo.value}`);

    }

</script>
```

El resultado es:

```
Atributo: type Valor: number
```

```
index.html:74 Atributo: id Valor: edad

index.html:74 Atributo: name Valor: edad

index.html:74 Atributo: min Valor: 18

index.html:74 Atributo: max Valor: 65
```

2 MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

Comenzaremos por la familia de propiedades siguientes, que enmarcamos dentro de la categoría de **reemplazar contenido** de elementos HTML. Se trata de una vía rápida con la cuál podemos añadir (*o más bien, reemplazar*) el contenido de una etiqueta HTML.

Las propiedades son las siguientes:

Propiedades	Descripción
STRING **.nodeName**	Devuelve el nombre del nodo (etiqueta si es un elemento HTML). Sólo lectura.
STRING **.textContent**	Devuelve el contenido de texto del elemento. Se puede asignar para modificar.
STRING **.innerHTML**	Devuelve el contenido HTML del elemento. Se puede usar asignar para modificar.
STRING **.outerHTML**	Idem a **.innerHTML** pero incluyendo el HTML del propio elemento HTML.
STRING **.innerText**	Versión no estándar de **.textContent** de Internet Explorer con diferencias. **Evitar**.
STRING **.outerText**	Versión no estándar de **.textContent**/**.outerHTML** de Internet Explorer. **Evitar**.

La propiedad **nodeName** nos devuelve el nombre del todo, que en elementos HTML es interesante puesto que nos devuelve el nombre de la etiqueta **en mayúsculas**. Se trata de una propiedad de sólo lectura, por lo cuál no podemos modificarla, sólo acceder a ella.

2.1 LA PROPIEDAD TEXTCONTENT

La propiedad **.textContent** nos devuelve el **contenido de texto** de un elemento HTML. Es útil para obtener (*o modificar*) **sólo el texto** dentro de un elemento, obviando el etiquetado HTML:

```
const div = document.querySelector("div");  // <div></div>
```

```
div.textContent = "Hola a todos"; // <div>Hola a todos</div>
div.textContent;  // "Hola a todos"
```

Observa que también podemos utilizarlo para **reemplazar el contenido de texto**, asignándolo como si fuera una variable o constante. En el caso de que el elemento tenga anidadas varias etiquetas HTML una dentro de otra, la propiedad **.textContent** se quedará sólo con el contenido textual completo, como se puede ver en el siguiente ejemplo:

```
// Obtenemos <div class="info">Hola <strong>amigos</strong></div>
const div = document.querySelector(".info");

div.textContent; // "Hola amigos"
```

2.2 La propiedad innerHTML

Por otro lado, la propiedad **.innerHTML** nos permite hacer lo mismo, pero interpretando el código HTML indicado y renderizando sus elementos:

```
const div = document.querySelector(".info"); // <div class="info"></div>

div.innerHTML = "<strong>Importante</strong>"; // Interpreta el HTML
div.innerHTML;   // "<strong>Importante</strong>"
div.textContent;  // "Importante"

div.textContent = "<strong>Importante</strong>"; // No interpreta el HTML
```

Observa que la diferencia principal entre **.innerHTML** y **.textContent** es que el primero renderiza e interpreta el marcado HTML, mientras que el segundo lo inserta como contenido de texto literalmente.

Ten en cuenta que la propiedad **.innerHTML** comprueba y parsea el marcado HTML escrito (*corrigiendo si hay errores*) antes de realizar la asignación. Por ejemplo, si en el ejemplo anterior nos olvidamos de escribir el cierre **** de la etiqueta, **.innerHTML** automáticamente lo cerrará. Esto puede provocar algunas incongruencias si el código es incorrecto o una disminución de rendimiento en textos muy grandes que hay que preprocesar.

Por otro lado, la propiedad **.outerHTML** es muy similar a **.innerHTML.** Mientras que esta última devuelve el código HTML del interior de un elemento HTML, **.outerHTML** devuelve también el código HTML del propio elemento en cuestión. Esto puede ser muy útil para reemplazar un elemento HTML combinándolo con **.innerHTML**:

```
const data = document.querySelector(".data");
data.innerHTML = "<h1>Tema 1</h1>";
```

```
data.textContent;  // "Tema 1"
data.innerHTML;    // "<h1>Tema 1</h1>"
data.outerHTML;    // "<div class="data"><h1>Tema 1</h1></div>"
```

En este ejemplo se pueden observar las diferencias entre las propiedades **.textContent** (*contenido de texto*), **.innerHTML** (*contenido HTML*) y **.outerHTML** (*contenido y contenedor HTML*).

Las propiedades **.innerText** y **.outerText** son propiedades **no estándar** de Internet Explorer. Se recomienda sólo utilizarlas con motivos de fallbacks o para dar soporte a versiones antiguas de Internet Explorer. En su lugar debería utilizarse **.textContent.**

3 MODIFICAR CSS

Es posible modificar el CSS de los elementos a través de los atributos **style** o **class** mediante el método **setAttribute** visto anteriormente. Pero al ser un elemento tan importante, JavaScript proporciona métodos especiales para manipular las clases CSS de un elemento.

3.1 PROPIEDAD STYLE

Los elementos poseen una propiedad llamada **style** que permite acceder a las propiedades CSS de un elemento. Lo que realmente hace es modificar el atributo style del elemento, modifican los valores de la misma. La idea es que style es un objeto que tiene como propiedades todas las propiedades CSS en formato **Camel Case** (mayúsculas estilo camello). Veámoslo con ejemplos.

```
let párrafo=document.getElementsByTagName("p")[0];
```

El código anterior selecciona el primer párrafo (de tipo p) del documento, sea cual sea.

```
parrafo.style.color="red";
```

Esta línea colorea de color rojo el texto de ese párrafo.

Podemos de esa forma modificar todo el CSS que queramos, por ejemplo:

```
parrafo. style. border="1px solid black"
```

Y así con todas las propiedades. El problema viene si hubiéramos querido modificar solo el borde inferior. La propiedad CSS para hacer esa modificación se llama border-bottom, es un nombre problemático para la sintaxis anterior, porque hay un guión en él. En JavaScript los identificadores de propiedades, funciones, métodos, etc, no pueden tener guiones (se confunde con la resta. Lo mismo ocurre la propiedad que modifica el color de fondo: background-color.

Hay que tener en cuenta que el formato de modificación CSS explicado anteriormente, es antiguo y entonces no se disponía de la posibilidad de acceder a una propiedad mediante corche tes. Por eso la propiedad **style** contiene una propiedad por cada propiedad CSS, pero usando el llamado formato **Camel Case**.

En el formato Camel Case, background-color se convierte en backgroundColor, border-bottom en borderBottom, text-align en textAlign y así con todas las propiedades. Por lo que modificar el color de fondo se hace así:

```
parrafo.style.backgroundColor="#CCC";
```

No obstante, la mayoría de navegadores actuales acepta también el formato idéntico a CSS a través de corchetes:

```
parrafo.style ["background-color"] ="+FCC";//Col orea de rojo claro
```

No obstante, hay varios problemas a tener en cuenta a la hora de usar style para modificar el CSS:

- **Manipular la propiedad** de los elementos **style da prioridad a cualquier otro CSS** (ya que se modifica la propiedad style de HTML), pero no siempre es lo deseable.
- Es más fácil de mantener el código si se usan clases para aplicar CSS. Especialmente, Cuando queremos modificar varias propiedades CSS a la vez.
- CSS avanza muy rápido por lo que muchas propiedades CSS ya disponibles, en algunos navegadores no lo están a través de la propiedad style de JavaScript. Aunque la mayoría de navegadores se ponen al día rápido.
- No podemos consultar a través de esta propiedad las propiedades CSS que tiene un elemento al cual se le ha dado formato a través de hojas de estilo externas.

3.2 OBTENER LOS ESTILOS CSS QUE SE APLICAN A UN ELEMENTO

Precisamente para paliar el último problema comentado en el apartado anterior, disponemos de un método en el objeto window llamado getComputedStyle. Este método devuelve un objeto (idéntico al de la propiedad style vista en el apartado anterior) en el que se pueden consultar las propiedades CSS que se están aplicando a un elemento concreto. Este método solo permite ver las propiedades del elemento, pero no modificarlas.

Como hemos comentado, se trata de un método del objeto window y no de los elementos en si. Por ello, el método requiere que se le pase el elemento a evaluar

```
let cssParrafo=window. getComputedStyle(parrafo);

console.log ( cssParrafo. fontFamily);
```

Este código mostrará por pantalla el valor de la propiedad CSS font-family que tiene el párrafo. Se mostrará el valor, independientemente de si esa propiedad se asignó usando la propiedad style o mediante una hoja de estilos externa o mediante JavaScript.

3.3 MANIPULAR LAS CLASES CSS

Los elementos disponen de una propiedad llamada **className**. Mediante esa propiedad pode-mos asignar una clase CSS a un elemento de la página. Por ejemplo:

```
parrafo.className= "remarcado";
```

También podemos simplemente obtener la clase del elemento. Por ejemplo:

```
console, log (parrafo.classNane); //Escribe remarcado
```

Los problemas surgen cuando hay más de una clase asignada al elemento. Si podemos ver las ca ses asignadas. Por ejemplo:

```
<p id="nota 20" class="remarcado anotacion"> A tener en cuenta que el 6 de abril se cierra por reforma</p>
```

Si ahora ejecutamos este código en la consola:

```
console. log( document.getElementByld( "nota20").className);
```

Obtendremos este resultado:

remarcado anotacion

Vemos las dos clases asignadas al párrafo. El problema es que acciones como añadir una clase más a un elemento o quitar una de las clases, se pueden hacer con de esta forma, pero muy incómodo manejar la propiedad class como un string.

Por eso JavaScript incorporó la propiedad **classlist** que veremos en la próxima unidad 7.4.

4 OBTENER ATRIBUTOS DATA

En los documentos HTML existe la posibilidad de crear atributos data. Se trata de un tipo de atributos creados por los propios desarrolladores a voluntad. Se usan para almacenar información que puede ser manipulada desde JavaScript.

Los atributos **data** empiezan por ese mismo término (data) seguidos de un guion y luego el nombre en sí que queramos darle al atributo. Por ejemplo:

```
<p id="p1" data-tipo="Inicio de libro" data-libro="Don Quijote" data-autor-principal="Miguel de Cervantes"> En un lugar de La Mancha. . s/p>
```

En este ejemplo se han diseñado dos atributos de forma personal. No es muy lógico grabar tanta información, pero nos interesa como ejemplo formativo. Ahora la cuestión es cómo se manipulan estos atributos desde JavaScript.

La propiedad **dataset** es la encargada de conseguirlo. El manejo es muy similar a lo que vimos respecto a la propiedad **style.**

dataset es un objeto que contiene como propiedades cada una de los atributos data del elemento actual. Para acceder concretamente a uno se usa una sintaxis de tipo Camel Case (de la que ya hablamos en el apartado de la propiedad style). Concretamente en este ejemplo, podemos ver los valores de los tres atributos de esta forma:

```
console. log(pi.dataset.libro);
console. log(p1 . dataset.autorPrincipal);
console.log(p1.dataset.tipo) ;
```

Muestra

Don Quijote
Miguel de Cervantes
Inicio de libro

Es posible también modificar un atributo data, o crearlo sino existe:

```
p1.dataset.publicacion=”Siglo XVII
```