1.5.- Contenido 5: Construir una página web utilizando scripts prediseñados en lenguaje JavaScript y JQuery para la manipulación del comportamiento de los componentes de la interfaz de usuario

Objetivo de la jornada

1. Describe el rol de los scripts JavaScript y su relación con CSS para otorgar comportamiento a los componentes de una página web.
2. Implementa una página web integrando scripts externos prediseñados para otorgarle el comportamiento requerido.
3. Implementa una página web integrando la biblioteca JQuery para otorgarle el comportamiento requerido.
4. Implementa ventanas modales en una página web utilizando JavaScript y JQuery para otorgarles el comportamiento requerido.

1.5.1.- Integrando JavaScript nuestros proyectos

1.5.1.1. Historia de JavaScript.

La primera versión de HTML, diseñada por Tim Berners-Lee de 1989 a 1991, era de naturaleza bastante estática. Excepto por los elementos enlaces con elementos <a>, las páginas web simplemente mostraban contenido fijo. En 1995, el fabricante dominante de navegadores era Netscape, y uno de sus empleados, Brendan Eich, pensó que sería útil agregar funcionalidad dinámica a las páginas web. De esta forma, diseñó el lenguaje de programación JavaScript, que agrega funcionalidad dinámica a las páginas web cuando se usa junto con HTML. Por ejemplo, JavaScript proporciona la capacidad de actualizar el contenido de una página web cuando ocurre un evento, como cuando un usuario hace clic en un botón[1].

Eich tardó solo 10 días en mayo de 1995 en implementar el lenguaje de programación JavaScript, una hazaña realmente notable. Marc Andreessen, uno de los fundadores de Netscape, originalmente nombró el nuevo lenguaje Mocha y luego LiveScript. Pero para fines de marketing, Andreessen realmente quería el nombre JavaScript. En ese momento, la industria del software estaba entusiasmada con el nuevo lenguaje de programación, Java. Andreessen pensó que todos los devotos del carro de Java se interesarían en su nuevo lenguaje de programación de navegador si este tenía el nombre Java. En diciembre de 1995, Andreessen cumplió su deseo cuando Netscape obtuvo una licencia de marca registrada del fabricante de Java, Sun Microsys-tems, y el nombre de LiveScript se cambió a JavaScript.

En 1996, Netscape envió JavaScript a la organización de estándares Ecma International para promover la influencia de JavaScript en todos los navegadores (no solo en el navegador de Netscape). Ecma International utilizó JavaScript como base para crear el ECMAScript y es actualmente el estándar para los lenguajes de programación interactivos integrados en todos los navegadores web. En el momento de la impresión de este documento, la versión más reciente de ECMAScript es la versión 10, publicada en 2019.

En 1998, Netscape formó la comunidad de software libre Mozilla, que finalmente implementó Firefox, uno de los principales navegadores de hoy. Brendan Eich se mudó a Mozilla, donde él y otros continuaron actualizando JavaScript a lo largo de los años, siguiendo el estándar ECMAScript establecido por Ecma International. Otros fabricantes de navegadores admiten sus propias versiones de JavaScript. Para sus navegadores Internet Explorer y Edge, Microsoft usa JScript. Para su navegador Chrome, Google usa el motor V8 JavaScript. Afortunadamente, todos los fabricantes de navegadores intentan seguir el estándar ECMAScript, por lo que para la mayoría de las tareas, los programadores pueden escribir una versión de su código y funcionará para todos los navegadores diferentes. En general en el mundo del desarrollo web se utiliza el término JavaScript, a pesar de que JavaScript es solo una de varias implementaciones de ECMAScript. A pesar de existir incluso el estándar ECMAScript10, por compatibilidad con los navegadores actuales, es razonable atenerse a la versión ECMAScript6. Sin embargo, los cambios más importantes en nuevas versiones son principalmente adición de nuevas funcionalidades, sin dejar inutilizado el código que es funcional en versiones anteriores. (JavaScript es la implementación utilizada en Firefox de Mozilla).

A JavaScript es el lenguaje "de lado cliente" más popular porque se ejecuta en contexto del navegador (cliente web). JavaScript también es muy fuerte en el lado servidor a través de node.js, sin embargo, para los propósitos de nuestro estudio nuestro foco principal con JavaScript es el lado cliente o Front-End. En este caso es el navegador el que soporta la carga de procesamiento y también es el que nos aporta los recursos con los que contamos para programar las aplicaciones.

Con JavaScript podemos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, con que cuenta este lenguaje es el propio navegador y todos los elementos que hay dentro de una página (que no es poco).

De manera adicional, gracias a la API JavaScript de HTML5, que están disponibles en los navegadores actuales de dispositivos móviles y de escritorio, podemos acceder a todo tipo de recursos adicionales, como la cámara, espacio para almacenamiento de datos, creación de gráficos basados en vectores y mapas de bits, flujos de datos con servidores, etc. Con todo ello se han multiplicado las posibilidades del lenguaje en comparación a sus inicios.

Una de las funcionalidades de mayor importancia para lo que nos compete, es la API del DOM HTML, con la cual tendremos un manejo completo de datos, estructura y despliegue de recursos en el navegador web de los usuarios de nuestras aplicaciones. Las áreas funcionales de la API del DOM HTML incluyen:

- Acceso y control de elementos HTML a través del DOM.
- Acceso y manipulación de datos de formularios web.
- Interacción con contenidos de imágenes 2D y el contexto del **<canvas>** HTML para, por ejemplo, dibujar sobre éste.
- Administración de medios audiovisuales conectados a elementos **<audio>** y **<video>**.
- Drag and drop de contenidos en páginas web.
- Acceso al historial de visitas del navegador.
- Soporte y conexión con otras APIs tales como Web Components, Web Storage, Web Workers, WebSocket, y eventos enviados por el servidor.

1.5.1.2. Agregando un script local en un HTML.

Podemos incorporar código JavaScript en un documento HTML de tres formas:

a. Dentro de atributos de elementos HTML.
b. En un elemento **<script> </script>** que esté dentro de la sección **<body>**.
c. Llamado a un archivo externo al código HTML, como parte de los recursos de la aplicación o desde CDN, el cual contendrá el código .

JAVASCRIPT EN ATRIBUTO DE ELEMENTO HTML

Esta es la forma más simple de incluir código JavaScript en un documento HTML. Sin embargo, esta práctica no se recomienda por el hecho de ensuciar código HTML con lenguaje dinámico de programación. A pesar de esto, los navegadores lo soportan para casos como los siguientes:

- Dentro de atributos de elementos que respondan a eventos (onclick, onmousedown, onmouseup, onmouseover, onmouseout, onchange, onsubmit, onreset, y otros).

- Dentro de atributos que contengan una URI (src o href). En estos casos el código deberá ir precedido de **JavaScript:**, por ejemplo: url(javascript: ...)

```
<body onload="alert('Hello World');">
...
</body>
```

Este ejemplo muestra de forma muy simple como crear una ventana de alerta con el mensaje "Hello World" al momento de cargar el documento HTML. Debido a sus limitaciones de espacio y formato, este método no resulta tan conveniente ni es considerado una buena práctica, como ya se mencionó más arriba.

DENTRO DE ELEMENTO <script>

Otra forma es incluir nuestro código JavaScript al interior de la un elemento HTML del tipo **<script>**. Normalmente la etiqueta **<script>** se incluye antes del cierre de la etiqueta **</body>**. A continuación se presenta un ejemplo de código al interior del documento HTML.

```
<body>
    <script type="text/javascript">
        var i = "Hola Mundo";
        alert(i);
    </script>
</body>
```

EN UN ARCHIVO .js EXTERNO A HTML

La forma más habitual de incluir JavaScript es a través de un archivo externo, el cual contiene el código que será interpretado por el navegador al momento de cargar la página, de forma similar a la carga de un archivo de hoja de estilos CSS.

Para realizar la carga del archivo, se utiliza el elemento **<script>** en el documento HTML. Típicamente se incluye este elemento <script> al final de <body>. De esta forma se asegura dar prioridad a la carga y despliegue de todo el contenido de la página para que sea rápidamente visualizado por el usuario, y luego pasar a los comportamientos programáticos provistos por JavaScript. desplieg

```
<body>
    … otros códigos html …
    <script src="index.js"></script>
</body>
```

En el ejemplo anterior, nuestro documento HTML está llamando a un archivo externo **index.js** donde se encuentra el código JavaScript a ejecutar.

1.5.1.3. Programando en JavaScript.

En el punto anterior, vimos como agregar código JavaScript a un documento HTML. Nuestro objetivo superior es llegar a manipular los elementos del DOM de un documento HTML con el poder que nos da un lenguaje de programación, en este caso JavaScript. Sin embargo, para llegar a explotar las posibilidades de la API del DOM para potenciar nuestros desarrollos con tecnologías web debemos tener un conocimiento, al menos básico, de las posibilidades de este lenguaje de programación.
A continuación hacemos una revisión breve de sus principales particularidades en términos de manejo de variables, tipos de datos, operadores, sentencias condicionales e instrucciones cíclicas.

VARIABLES O IDENTIFICADORES

Todos los lenguajes de programación necesitan en algún momento cargar en memoria los datos que se van a procesar. Por ejemplo, pedir el nombre de un usuario y almacenarlo en la variable **nombreUsuario** o guardar en la variable **resultado** el valor obtenido al multiplicar dos números cualesquiera. Así, las variables son fundamentales al igual que en todos los lenguajes. Debemos mantener presente que **SIEMPRE** los nombres de variables deben ser autoexplicativos en el contexto del programa del que es parte. Por ejemplo si tenemos una variable que almacena un nombre de usuario, preferiremos llamarla **nombre** en lugar de **n**. Esta práctica nos permitirá generar código que pueda ser entendido, analizado, mantenido y mejorado por grupos de desarrollo.

La asignación de nombres para variables en JavaScript tiene algunas consideraciones que debemos respetar:

- El primer carácter debe ser siempre una letra, el guión bajo **_** o el símbolo **$**. Los restantes caracteres pueden ser letras, números o el guión bajo, teniendo como precaución no dejar espacios entre ellos.

- JavaScript diferencia mayúsculas de minúsculas.

- El nombre de la variable no puede coincidir con palabras claves pertenecientes a la propia sintaxis de JavaScript, ni con palabras reservadas de JavaScript. Así, debemos evitar utilizar los siguientes nombres para variables:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

(*) : nuevas en versiones ES5 y ES6.

Fuente: https://www.w3schools.com/js/js_reserved.asp

Para utilizar una variable como parte de nuestro código, primero es necesario crearla asignándole un nombre simbólico, con la siguiente sintaxis:

```
var miEdad;
let miNombre;
```

En el ejemplo anterior, se han creado las variables **miEdad** y **miNombre** sin darles un valor particular, es decir, se ha creado lo que podríamos llamar un contenedor sin un contenido en su interior. Una vez que la variable ha sido creada, podemos asignar un valor o inicializar la variable de la siguiente forma:

```
let miNombre;

miNombre = "Juan";

var miEdad = 55;
```

En el primer caso, se asigna el valor a la variable luego de ser creada. En el segundo caso, el valor se asigna en la misma línea en que la variable es creada.

Por lo tanto, la sentencia **let** y **var** nos premiten definir variables. Dependiendo de dónde estén utilizadas tendrán un alcance distinto[2]:

- **Alcance Global: let** y **var** definen una variable global si se utilizan en la sección principal del código JavaScript.

- **Alcance de Bloque:** Si definimos variables con **let** y **var** dentro de un bloque delimitado por {...}, la primera existirá sólo dentro del bloque y la segunda continuará existiendo fuera de éste.

- **Alcance de Función:** Si definimos variables con **let** y **var** dentro de una función, ambas variables sólo existirán dentro del alcance de esa función.

- **Alcance de Loop:** Las variables definidas con **let** utilizadas como contador de loop existirán sólo en el ámbito de ese loop. Contrariamente, un contador de loop definido con **var** seguirá existiendo fuera de éste y habrá redeclarado una eventual variable del mismo nombre que haya existido fuera del mismo.

TIPOS DE VARIABLES

Una variable en JavaScript puede almacenar distintos tipos de datos, los principales son:

Variable	Explicación	Ejemplo
String	Esto es una secuencia de texto conocida como cadena. Para indicar que la variable es una cadena, debes escribirlo entre comillas.	let miVariable = 'Bob';
Number	Esto es un número. Los números no tienen comillas.	let miVariable = 10;
Boolean	Tienen valor verdadero/falso. true/false son palabras especiales en JS, y no necesitan comillas.	let miVariable = true;
Array	Una estructura que te permite almacenar varios valores en una sola referencia.	let miVariable = [1,'Bob','Steve',10]; Llama a cada miembro del array así: miVariable[0], miVariable[1], etc.

Object	Básicamente cualquier cosa. Todo en JavaScript es un objeto y puede ser almacenado en una variable. Mantén esto en mente mientras aprendes.	let miVariable = document.querySelector('h1');
		let alumno = {
		nombre: “Alberto”,
		edad: 22,
		nivel: B,
		puntaje: 88
		};
		Todos los ejemplos anteriores también.

F*uente: https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/JavaScript_basics*

Nota de interés: El actualmente conocido formato de almacenamiento y transmisión de datos JSON, tuvo su origen en los objetos de JavaScript. **JSON** hace referencia a **JavaScript O**bject **N**otation.

CONSTANTES

Una constante es un tipo especial de variable en el cual su valor, una vez inicializado, no puede modificarse. Este tipo de variable se utiliza para almacenar valores que deben permanecer fijos y no ser alterados dentro del código. La forma de declarar una constante es la siguiente:

```
const horasDia = 24;
```

En relación al alcance de los identificadores declarados con **const**, debemos decir que es del mismo tipo que las variables declaradas con **let**.

ESTRUCTURAS CONDICIONALES

Tal como ocurre en la vida real, en la cual tomamos decisiones en base a condiciones (Por ejemplo: Si llueve, entonces uso mi paraguas), en JavaScript es posible decidir realizar una acción dependiendo de distintas entradas, por ejemplo, comparando dos valores. La forma de realizar la comparación y si realizamos o no cierta acción es a través de la estructura condicional **if-else**.

La estructura **if-else** opera de la siguiente forma

```
if(condición){
    Si se cumple la condición, entonces ejecutar una
    o más instrucciones que estarán ubicadas en este bloque
}
else {
    Si no se cumple la condición, entonces
    ejecutar una o más instrucciones que estarán
    ubicadas en este bloque
}
```

Si queremos tener más alternativas en caso de no cumplir la condición inicial, utilizaremos la siguiente forma:

```
if(se verifica condición 1){
    Si se cumple la condición, entonces ejecutar una
    o más instrucciones que estarán ubicadas en este bloque.
}
else if(se verifica condición 2){
    Si no se cumple la condición 1, pero sí la condición 2,
    entonces ejecutar una o más instrucciones ubicadas
    en este bloque.
}
else if(se verifica condición 3){
    Si no se cumple las condiciónes 1 y 2, pero si la
    condición 3, entonces realizar una o más instrucciones
    ubicadas en este bloque.
}
else {
    En este bloque se incluyen las instrucciones a ejecutar
    si no se cumple alguna de las condiciones anteriores.
}
```

Cuando la condición se cumple, la verificación de condición tendrá valor true (verdadero), por lo que ejecutará el código presente entre el bloque { } correspondiente. En caso de que la condición no se cumpla, la verificación de condición tendrá valor false (falso), por lo que se procederá a evaluar el siguiente else if, y así sucesivamente, hasta verificar que una condición se cumpla. Si ninguna de las condiciones en else if se cumple, entonces serán ejecutadas las instrucciones presentes en el bloque { } correspondiente a else.

OPERADORES

Los operadores permiten combinar diferentes valores en una expresión, manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables.

Ya hemos visto cómo trabajan las variables con los datos, y ahora veremos cómo manipular dichas variables asignando, si es necesario, los resultados a nuevas variables.

Los operadores de JavaScript pueden dividirse en varios grupos. Su clasificación es meramente funcional y se basa en la operación que realizan. Revisamos a continuación algunas de estas categorías:

Operadores matemáticos.

JavaScript permite realizar operaciones matemáticas de forma directa en la consola del navegador o como parte del script. Los operadores matemáticos son:

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
**	Exponenciación
/	División
%	Módulo (Resto de la división)
++	Increment
--	Decrement

Operadores de Comparación y Lógicos

Un operador de comparación determina si una operación entre dos operandos es verdadera o false, por ejemplo en estructuras condicionales if-else como mostramos más arriba. A su vez, un operador lógico entrega el resultado de la operación lógica entre los operandos. Los hemos agrupado por tratarse de operadores que entregan resultados **true/false**:

Operador	Descripción
==	Verifica igualdad de valor
===	Verifica igualdad de valor y tipo
!=	Verifica valor diferente
!==	Verifica valor y/o tipo diferente
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
?	Operador ternario
&&	AND lógico
||	OR Lógico
!	NOT Lógico

Mostramos un ejemplo de la combinación de operadores de este tipo, en base a operadores lógicos:

```
if(!(condición1 && condición2) || condición3){
    …
}
else {
    …
}
```

En este caso al utilizar (!) se está “negando” el resultado de las condiciones 1 y 2 operadas con un && (AND). El resultado ya negado se opera con || (OR) a la condición 3. Con esto, si el resultado final es true, se ejecuta el primer bloque de código, de lo contrario se ejecuta el bloque correspondiente a else.

Operadores de String o Cadena de Caracteres.

+ : Concatenación de dos strings. Ejemplo: “Hola” + “Mundo”.

+= : Concatena variable tipo string con string y lo almacena en la misma variable. Ejemplo: variableTexto += “otro texto”

ITERACIONES (Looping)

Una iteración es un proceso que se repite una o varias veces. En programación es muy útil contar con este tipo de herramientas, ya que nos evita escribir las mismas operaciones una y otra vez. Supongamos que contamos con una lista de cien nombres a quienes queremos enviar el mismo mensaje, si realizamos este procedimiento manualmente, entonces debemos escribir cien veces la misma instrucción y mensaje, lo cual es poco eficiente. Utilizando un proceso de iteración, solo es necesario escribir el mensaje una vez y recorrer la lista de personas para asignarle el destinatario a una sola instrucción de envío de mensaje.

Existen varias formas o expresiones para crear un proceso de iteración, las más comunes son **for**, **while** y **do while**. Todas estas formas se basan en una estructura similar: Contador, condición de salida, iterador.

- **Contador**: variable que inicia el ciclo con determinado valor.
- **Condición de Salida**: será el criterio con el que finaliza la iteración.
- **Iterador o incremento**: acción que actualiza el valor original (actual) del contador una y otra vez hasta alcanzar la condición de salida.

A continuación veremos la estructura de cada uno en pseudocódigo:

```
for(contador; condición salida; incremento){
    Código a ejecutar…
}

-----------------------------------------
```

```
contador
while(condición salida){
    Código a ejecutar…
    incremento
}

-------------------------------------------

contador
do{
    código a ejecutar…
    incremento
} while(condición salida)
```

Un ejemplo simple con **for**:

```
let nombres = ["Andrea", "Juan", "Camila", "Antonio"];
for(let i = 0; i < nombres.length; i++){
    console.log(nombres[i]);
}
```

Primero, se ha creado la variable, que en este caso es un arreglo (array), sobre la cual queremos realizar la iteración. Luego, construimos el iterador **for** donde se define **i** como el contador, el cual tiene un valor inicial igual a **0**. Posteriormente, la operación comprueba si el valor de **i** es menor al largo de la lista. Como el largo de la lista es **4**, se cumple que **0 < 4** por lo que a continuación se ejecuta el código entre **{ }**. Una vez ejecutado el código se realiza el incremento en el contador (**i++**), por lo que ahora i = **1**. El iterador comprueba que **1 < 4**, por lo que se ejecuta el código entre **{ }** nuevamente. Este proceso se repite hasta que **i = 4**.
Nota: Las expresiones **for**, **while**, **do-while** e **if** no necesitan incluir "**;**" luego de cerrar el bloque **{ }**.

FUNCIONES
Una función es un bloque de código que se ejecuta solo cuando es llamada desde el código principal o desde otra función. Una función puede ser reutilizada, es decir, podemos ejecutarla cada vez que ésta es llamada sin alterar su comportamiento.
Para crear una función es necesario declararla de la siguiente forma:

```
function nombreDeLaFuncion(parámetros){
    Código a ejecutar…
}
```

Alternativamente las funciones en JavaScript pueden ser declaradas de la siguiente forma (Para el mismo ejemplo):

```
nombreDeLaFuncion = function(parámetros){
    Código a ejecutar…
}
```

Para declarar la función, es necesario siempre comenzar con la palabra reservada **function**, luego debe escribirse el nombre de la función utilizando caracteres alfanuméricos. A continuación del nombre dado a la función, se debe incluir entre paréntesis **( )** los argumentos que serán utilizados al interior del bloque al interior de la función, entre **{ }**. A continuación un ejemplo simple:

```
function suma(n, m){
    var resultado = n + m;
    return resultado;
}
var valor;
valor = suma(4, 6);
console.log(valor);
```

El script no ejecuta directamente la función **suma**. Primero, es necesario que exista una llamada a la función, en donde se incluyan los argumentos que ésta requiere. En el ejemplo, nuestra función requiere que se incluyan dos números para utilizarlos en la operación, el resultado de la operación se almacena en la variable **resultado** y luego el valor contenido en esta variable es devuelto y almacenado en la variable **valor** del código principal. Finalmente, este **valor** se imprime en la consola.

INTERACCIÓN CON EL DOM

Hasta el momento, se han presentado los conceptos básicos de programación con JavaScript. Sin embargo, es importante conocer como interactúa un programa de JavaScript con un documento HTML, que es finalmente lo que diferencia un sitio web

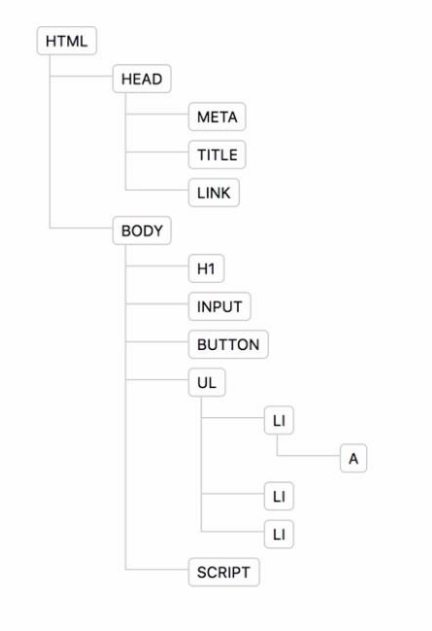
estático de uno dinámico. Utilizar JavaScript nos permite aprovechar los elementos de un sitio web como botones y formularios, e incluso modificar dinámicamente el contenido de un sitio en base a condiciones externas como información del usuario, fecha, hora, etc. o responder a eventos como clicks o movimientos del mouse.

Observemos el siguiente ejemplo de un sitio web muy simple, con algunos elementos que nos servirán de muestra para entender mejor como interactúa JavaScript con el DOM.

Nuestro código HTML es el siguiente:

```
<html lang=”es”>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1 id="titulo">Hola Mundo!</h1>
    <input type="checkbox">
    <button style=":active: color:red;">OK</button>
    <ul>
      <li class="list"><a href="http://www.google.com">Google</a></li>
      <li class="list">Segundo Elemento</li>
      <li class="list">Tercer Elemento</li>
    </ul>
    <script src="index.js" charset="utf-8"></script>
  </body>
</html>
```

Para el ejemplo presentado, el DOM posee la siguiente estructura.



Como muestra la figura, el documento posee un nodo hijo, HTML, el cual a su vez posee dos nodos hijos, HEAD y BODY. Si se quiere acceder al segundo nodo hijo (BODY), es posible acceder a través de la consola del navegador de la siguiente forma:

```
>> document.firstChild.lastChild;
<- <body>
```

Ahora, si queremos acceder al texto del elemento H1, entonces podemos correr la siguiente expresión

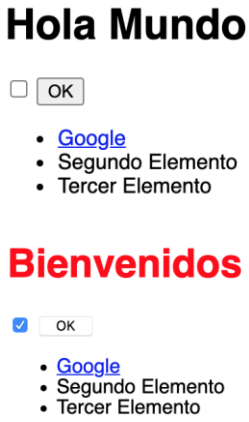
```
>> document.firstChild.lastChild.firstElementChild.innerHTML;
<- Hola Mundo!
```

Si bien la expresión es un tanto engorrosa, sirve para mostrar la estructura del DOM y entender cómo acceder a los elementos según su posición.

El siguiente ejemplo muestra cómo modificar algunas de las propiedades del elemento H1.

```
>> var tituloH1 = document.firstChild.lastChild.firstElementChild;
>> tituloH1.innerHTML = "Bienvenidos";
>> tituloH1.style.color = "red";
>> document.querySelector("input").click();
```

En el navegador observamos el siguiente cambio.



A continuación mostramos algunos de los métodos para interactuar con el DOM más comunes y una breve explicación[3]:

Método	Descripción
document.getElementById('id')	Obtiene el elemento, con un identificador dado, como un objeto.
document.getElementsByTagName('tag-name')	Obtiene todos los elementos con nombre de tag **tagname** y los guarda en un arreglo.
node.getAttribute('attribute')	Obtiene el valor del atributo de nombre **attribute**.
node.setAttribute('attribute', 'value')	Asigna el valor **value** al atributo de nombre **attribute**.
node.nodeType	Lee el tipo de nodo (1=elemento, 3=nodo texto).
node.nodeName	Lee el nombre del nodo, sea éste nombre de elemento o #textNode.
node.nodeValue	Lee o asigna el valor del nodo (El contenido de texto en caso de nodos de texto).
node.previousSibling	Obtiene el nodo hermano previo y lo almacena en un objeto.
node.nextSibling	Obtiene el nodo hermano siguiente y lo almacena en un objeto.
node.childNodes	Obtiene todos los nodos hijos de un objeto y los almacena en una lista. Hay atajos para el primer y último nodo hijo, llamados node.firstChild y node.lastChild, respectivamente.
node.parentNode	Obtiene el nodo que contiene a **node**.
document.createElement(element)	Crea un nuevo elemento con el nombre **element**, que debemos especificar como un string.
document.createTextNode(string)	Crea un nuevo nodo de texto con el valor de nodo **string**.
newNode = node.cloneNode(bool)	Crea un nuevo nodo como copia (clon) de **node**. Si **bool** es **true**, el clon incluye clones de todos los nodos hijos del nodo original.
node.appendChild(newNode)	Agrega **newNode** como un nuevo (al final) nodo hijo de **node**.

node.insertBefore(newNode,oldNode)	Inserta **newNode** como un nuevo nodo hijo de **node**, antes de **oldNode**.
node.removeChild(oldNode)	Remueve el nodo hijo **oldNode** de **node**.
node.replaceChild(newNode, oldNode)	Reemplaza el nodo hijo **oldNode** de **node** con **newNode**.
element.innerHTML	Lee o asigna el contenido HTML de un elemento dado como un string. Incluye todos los nodos hijos con sus atributos y contenidos de texto.

MANEJO DE EVENTOS

El código JavaScript que desarrollamos e incorporamos a nuestros documentos HTML, puede ejecutarse al momento de terminar la carga de la página por el navegador, porque ciertos intervalos de tiempo definidos se hayan cumplido, si se detectan ciertas condiciones en el código HTML, etc. Otra forma es ejecutar código solo cuando un usuario realiza ciertas acciones, como hacer clic, escribir o incluso simplemente mover el mouse. Esta permitirá a los usuarios interactuar con su página web para que su ésta responda de acuerdo con las acciones que ellos realizan.

En un navegador, cada vez que realiza una acción, como hacer clic, escribir o mover el mouse, se gatilla lo que se denomina **Evento**. Un evento es la forma en que el navegador dice: "¡Algo ha sucedido!" Puede escuchar estos eventos agregando un manejador de eventos (Event handler) al elemento donde se presenta ese evento en particular que queremos detectar. Agregar un controlador de eventos es la forma de decir a JavaScript: “Si este evento ocurre un momento determinado, entonces ejecute cierta función." Por ejemplo, si desea que se llame a una función cuando el usuario hace clic en un elemento de la página como un botón, podemos agregar un controlador de eventos de clic al elemento botón.

Eventos comunes son los siguientes:

Evento	Descripción
onchange	Ha cambioado un elemento HTML
onclick	El usuario ha hecho clic sobre un elemento HTML
onmouseover	El usuario posiciona el mouse sobre un elemento HTML
onmouseout	El usuario saca el mouse de un elemento HTML en el que se había posicionado
onkeydown	El usuario presiona una tecla del teclado
onload	El navegador ha terminado de cargar la página

Existen tres formas de agregar un manejador de eventos en JavaScript:

1. Atributo de manejo de evento de HTML.
2. Propiedad de manejo de eventos del elemento.
3. Agregando un monitor de eventos con **addEventListener()**.

El primero de ellos no es aconsejado y se sugiere evitarlo debido a que el código JavaScript estará mezclado dentro del código HTML y complicará su mantenimiento y desarrollo. Además, puede producir errores en casos en que la página haya cargado pero los códigos JavaScript que contienen la función llamada aún no terminan de cargar. Por ejemplo en el siguiente caso, la función showAlert() podría no estar cargada aún y el usuario podría haber presionado el botón con anterioridad:

```
<input type="button" value="Save" onclick="showAlert()">
```

La segunda forma mencionada de agregar un manejador de eventos es asignando la propiedad de manejo de eventos del elemento, por ejemplo **onclick**, a una función como la siguiente:

```
let btn = document.querySelector('#btn');

btn.onclick = function() {
    alert(this.id);
};
```

En este caso, la función anónima se convierte en el método de un elemento **<button>**. Así, **this** corresponde al elemento propiamente tal y se puede acceder a sus propiedades y métodos desde JavaScript.

Para remover el manejador de eventos se utiliza:

```
btn.onclick = null;
```

La tercera forma descrita corresponde a utilizar el método **addEventListener()**, especificando el nombre del evento y la función manejadora del evento. Por ejemplo:

```
let btn = document.querySelector('#boton_test');

// agregando manejador de evento
let mostrarAlertas = function() {
    alert('Clicked!');
    alert(event.type);
};

btn.addEventListener('click', mostrarAlertas);
```

En este caso, gatillamos dos acciones de alerta al momento de efectuarse clic sobre el elemento de identificador **boton_test** en nuestro documento HTML. La eliminación del manejador de eventos la realizamos con:

```
// remueve el manejador de eventos
btn.removeEventListener('click', mostrarAlertas);
```

1.5.1.4. Agregando JQuery a una página web.

LIBRERÍA JQuery

JQuery es una librería de JavaScript que permite manipular elementos, eventos, animaciones y otros, simplificando la sintaxis de JavaScript. Por ejemplo, si se quiere acceder a los elementos de etiqueta **<p>**, las versiones de código JavaScript y JQuery para lograrlos serían las siguientes:

```
>> document.getElementsByTagName("p");
>>$("p"); //JQuery
```

COMO INCLUIR JQuery

Es posible incluir la librería JQuery instalando directamente en el servidor o computador utilizando gestores de librería tales como npm o Yarn, o bien importando la versión alojada en una Red de Distribución de Contenido (CDN) como lo hemos hecho en ejemplos de secciones anteriores.

Si, por ejemplo, queremos trabajar con JQuery desde el CDN de Google, podemos hacerlo de la siguiente forma, agregándola antes de importar el archivo JavaScript donde tengamos nuestro código que usa JQuery.

```
<script
 src="https://ajax.goo-
gleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script src="index.js" charset="utf-8"></script>
```

Esto es válido para la importación de cualquier otro plugin JavaScript a nuestros proyectos.

USO DE JQuery

Existe una infinidad de selectores, métodos, propiedades y eventos para utilizar JQuery. No cubriremos taxativamente todos los detalles de esta librería. Existe una amplia documentación disponible. Reiteramos que JQuery es una herramienta que está confeccionada para simplificar tareas que en JavaScript requieren más elaboración. Mostramos a continuación algunos ejemplos simples de JQuery para modificaciones de estilo, interacción con contenido del documento HTML y detección de eventos.

Para acceder a los parámetros de estilo, jQuery añade el método **css** a continuación del elemento que se quiere modificar. Luego se incluye entre paréntesis el parámetro de estilo y su nuevo valor. Ejemplo:

```
$("p").css("font-family", "Arial");
$("h1").css("color", "red");
```

Podemos acceder al contenido existente en el documento, como también modificarlo. Ejemplo:

```
$("h1").html(); //obtiene texto en etiqueta h1
$("h1").html("Nuevo Texto"); //modifica el texto en etiqueta h1
$var parrafoNuevo = $("#id").html("<p> texto </p>");//crea objeto jQuery
```

Finalmente, podemos acceder a los métodos para controlar eventos, tales como clicks de mouse o submits de formularios.

```
$("#boton").click(function(){
    alert("Presionaste el boton");
});
```

1.5.2.- Referencias

[1] John Dean, WEB PROGRAMMING with HTML5, CSS, and JavaScript, 2019.

[2] JavaScript Tutorial
https://www.w3schools.com/js/

[3] Christian Heilmann, JavaScript DOM Cheatsheet.
https://christianheilmann.com/stuff/JavaScript-DOM-Cheatsheet.pdf

[4] Guía de JavaScript, Mozilla Developer Network.
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide

[5] Nicholas C. Zakas, Understanding ECMASCript 6, 2016.