1. Introducción

1.1. Introducción a JavaScript

(Archivo)

1.2. Instalaciones y requerimientos

Instalación VSC.

2. DOM

2.1. Incorporando JavaScript a HTML

(Ver Archivo)

Se incorpora la etiqueta **script** con la ruta al archivo **.js**, antes del cierre de la etiqueta **body**.

<script src="ruta"></script>
</body>

2.2. Alert, prompt y confirm

(Ver Archivo)

alert(mensaje);

Ejecuta un diálogo de alerta con un mensaje opcional, y aguardará hasta que el usuario cierre la ventana de diálogo.

confirm(pregunta);

Indica al navegador que muestre un diálogo con un mensaje opcional y que espere hasta que el usuario confirme o cancele el diálogo. Devuelve un booleano.

prompt(mensaje, texto);

Muestra un diálogo con mensaje opcional, que solicita al usuario que introduzca un texto, el texto se puede setear en la segunda propiedad para que aparezca en el input del diálogo como valor por defecto. Devuelve el valor del input, si está vacío devuelve un string vacío, si se cancela devuelve null.

typeof(variable); o **typeof variable;**

Devuelve el tipo de dato de la variable asignada como propiedad del método.

2.3. Repaso Array y Object

(Ver Archivo)

for … of: for (const elemento of array) {
…
}

Ejecuta un bucle que opera sobre una secuencia de valores de un objeto iterable.

for … in: for (const propiedad in object) {
…
}

Itera sobre todas las propiedades de cadena de texto enumerables de un objeto, incluyendo las propiedades enumerables heredadas.

2.4. DOM Document Object Model

(Ver Archivo)

DOM:

Es el HTML leído por el navegador. Por un lado existe el documento html estático, y por el otro, el documento que el navegador interpreta como nodos. Esta lectura de nodos es el DOM que Javascript puede manipular.

El DOM (Document Object Model, en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

document

Es la interfaz que representa cualquier página web cargada en el navegador y sirve como punto de entrada al contenido de la página web, que es DOM. Describe las propiedades y métodos comunes para cualquier tipo de documento.

El árbol DOM incluye elementos como **<body>** y **<table>**, entre muchos otros. Proporciona funcionalidad globalmente al documento, como obtener la URL de la página y crear nuevos elementos en el documento.

document.getElementById(identificador);

Método que devuelve un objeto que representa el elemento de la propiedad **id** especificada. Dado que los identificadores de elementos deben ser únicos si se especifican, son una forma útil de obtener acceso a un elemento específico rápidamente.

Element.innerHTML

Propiedad que devuelve o establece la sintaxis HTML describiendo los descendientes del elemento. Al establecerse se reemplaza la sintaxis HTML del elemento por la nueva.

document.getElementsByClassName(clase);

Método que devuelve un **HTMLCollection** (array o colección) con todos los elementos con el nombre de la clase definida en la propiedad.

document.getElementsByTagName(etiqueta);

Método que devuelve un **HTMLCollection** (array o colección) con todos los elementos con el nombre de etiqueta HTML indicado en la propiedad.

Element.style.propiedad

Propiedad que devuelve los estilos en línea de un elemento en forma de un objeto (CSSStyleDeclaration) que contiene sus propiedades y valores.

Element.setAttribute(nombre, valor);
Método que establece el valor de un atributo en el elemento especificado. Si el atributo ya existe, se actualiza el valor; de lo contrario, se agrega un nuevo atributo con el nombre y el valor especificados. La propiedad **nombre** indica el nombre del atributo, y **valor** su contenido.

2.5. DOM querySelector y form

(Ver Archivo)
Element.querySelectorAll(selector);
Método que devuelve un arreglo que representa una lista de elementos del documento que coinciden con el grupo de selectores indicados en la propiedad. **Element** puede ser tanto **document** como otro grupo de elementos seleccionados previamente denominado también como **NodeList** o **ParentNode**.

El parámetro **selector**, es un string que contiene uno o más selectores para buscar coincidencias y debe ser una cadena de **selectores CSS** válida.

document.forms;
document.forms[índice];
Propiedad que devuelve un arreglo con una lista de todos los elementos **<form>** contenidas en el elemento. El valor del **índice** puede ser su ubicación dentro del array o su **id** específico.

3. Eventos

3.1. Eventos

(Ver Archivo)
Los eventos son todas las acciones que pasan dentro de una página web. Son disparadores que permiten realizar otras acciones, que pueden provenir del DOM o de desarrollos propios en el código, y sirven para lograr que la página o el sistema se comporte como se lo requiere.

onclick=”método”;
Atributo de los elementos que se activa cuando el mouse hace click en él, ejecutando el **método** definido.

onload=”método”
Atributo de ciertos elementos que ejecuta el **método** definido, una vez que el elemento se cargó exitosamente. Los elementos que soportan este evento son:

- **<body>**
- **<embed>**
- **<iframe>**
- **<img>**
- **<link>**
- **<object>**
- **<script>**
- **<style>**
- **<track>**

navigator
Interfaz, tal como lo es **document** que se encuentra dentro de **window**, que representa el estado y la identidad del navegador, también llamado ***user agent***.

onunload=”método(evento)”
Atributo que dispara un evento cuando el documento o un recurso o elemento se descarga. Se está deprecando.

oninput=”método(evento)”
Atributo que dispara un evento que se activa cuando el valor de un elemento **<input>**, **<select>** o **<textarea>** ha cambiado como resultado directo de una acción del usuario.

onmouseover=”método(evento)”
Atributo que activa un evento cuando se utiliza un dispositivo señalador (como un mouse o un panel táctil) para mover el cursor sobre el elemento o uno de sus elementos secundarios.

onmouseout=”método”
Atributo que activa un evento cuando se utiliza un dispositivo señalador (normalmente un ratón) para mover el cursor de modo que ya no se encuentre dentro del elemento o de uno de sus elementos secundarios.

onfocus=”método(evento)”
Atributo que devuelve dispara un evento cuando el usuario establece el foco en el elemento.

3.2. Eventos de teclado

(Ver Archivo)
onkeydown=”método(evento)”
Atributo que activa un evento al presionar una tecla. Proporcionan un código que indica qué tecla se presionó, mientras que keypress indica qué carácter se ingresó. El elemento de un evento de tecla es el elemento en el que se encuentra el foco y que está procesando la actividad del teclado.

onkeyup=”método(evento)”
Atributo que activa un evento al soltar una tecla. Proporcionan un código que indica qué tecla se presionó, mientras que keypress indica qué carácter se ingresó. El elemento de un evento de tecla es el elemento en el que se encuentra el foco y que está procesando la actividad del teclado.

onkeypress=”método(evento)”
Atributo que activa un evento al mantener presionada una tecla con una letra, un número, un signo de puntuación o un símbolo, o cuando se presiona la tecla Intro (incluso cuando se presiona la tecla Intro en combinación con la tecla Shift o Ctrl). De lo contrario, cuando se presiona una tecla modificadora, como Alt, Shift, Ctrl, Meta, Esc u Opción, de forma aislada, el evento de pulsación de tecla no se activa.

.altKey
Es una propiedad del evento dentro de los métodos de teclado, devuelve un true si la tecla que ejecuta el evento es Alt y false si es cualquier otra.

.getModifierState(tecla)
Método que devuelve el estado actual de la tecla modificadora especificada: verdadero si el modificador está activo (es decir, la tecla modificadora está presionada o bloqueada), de lo contrario, falso, es decir, si hay una tecla modificadora activada.

4. Manipulación del DOM
    4.1. Manipular nodos en el DOM

(Ver Archivo)
document.createElemet(nodo, options)
Método que crea el nodo HTML especificado en la propiedad **nodo**. La propiedad opcional **options**, es un objeto con la propiedad **is** definida en el método **define()**.

document.createTextNode(data)
Método que crea un nodo de texto. Puede ser usado para escapar caracteres HTML. El parámetro **data** es un string que contiene el dato que se quiere ingresar al nodo de texto.

Element.appendChild(child)
Método de los elementos que agrega un nuevo nodo al final de la lista de un elemento hijo de un elemento padre especificado. Si el hijo resultante es una referencia hacia un nodo existente en el documento actual, este es quitado del padre para ser puesto en el nuevo nodo padre. Devuelve un nuevo nodo añadido. El parámetro **child** es el nodo a añadir al no padre especificado como **element**.

Element.insertBefore(newNode, referenceNode)
Método de los elementos que inserta el nuevo nodo definido en el argumento **newNode**, antes del nodo de referencia definido en el argumento **referenceNode**.

Element.remove()
Método que elimina el elemento del DOM.

Element.remove(child)
Método que elimina el nodo hijo especificado en el argumento y lo devuelve.

Elemento.replaceChild(newChild, oldChild)
Método que reemplaza el elemento del segundo argumento por el elemento del primer argumento.

    4.2. Manipulando nodos

4.3. Agregar y quitar clases

(Ver Archivo)

Element.classList

Propiedad que retorna una colección (arreglo) de los atributos c*lase* del elemento. Se puede operar sobre la propiedad con los métodos:

- **.classList.add(class, …)**: agrega los valores definidos en los argumentos. Puede incluir una cantidad indefinida de argumentos, en este caso, agrega clases.
- **.classList.remove(class, …)**: remueve los valores definidos en los argumentos. Puede incluir una cantidad indefinida de argumentos, en este caso, remueve clases.
- **.classList.replace(oldClass, newClass)**: reemplaza el valor definido en el primer argumento, por el valor definido en el segundo, en este caso, clases.
- **.classList.toggle(class, force)**: remueve o agrega el valor definido en el primer argumento de la lista. Si no encuentra el elemento, lo agrega, y si lo encuentra, lo elimina. El segundo argumento opcional es un booleano que "fuerza" la acción del toggle.
- **.classList.toggle(class)**: evalúa si el valor del argumento se encuentra en el arreglo, en este caso, de clases.

4.4. Agregar y quitar atributos

(Ver Archivo)

Element.getAttribute(atributo)

Método que devuelve el valor del atributo especificado en su argumento.

Element.setAttribute(atributo, valor)

(Ver más arriba)

Element.removeAttribute(atributo)

Método que remueve el atributo con el nombre especificado en su argumento.

4.5. Agregar y quitar estilos

(Ver Archivo)

Element.style.propiedad

(Ver más arriba)

Las propiedades se escriben en camelCase, cada separador por guión medio (-) lo reemplaza por una mayúscula de la palabra subsiguiente. Cada propiedad, además de setearla, sirve también para obtener su valor:

- **style.color**: propiedad CSS *color*
- **style.backgorundColor**: propiedad CSS *background-color*

Element.querySelector(selector, …);

Método que devuelve el primer elemento del documento que coincida con el atributo especificado de selectores. El atributo **selector** es una cadena de caracteres que contiene uno o más selectores CSS separados por coma.

4.6. BOM Browser Object Model

Es un objeto el cual tiene atributos, métodos y otras propiedades del navegador propio. Se identifica con el objeto **window**, el cual contiene a **document**, es decir, representa la ventana que contiene un documento DOM.

(Ver Archivo)
window
Es la palabra reservada para llamar al objeto que representa al navegador. En navegadores con pestañas, cada una contiene su propio objeto **window**.

window.innerHeight
Propiedad que representa la altura en píxeles del viewport.

window.innerWidth
Propiedad que representa el ancho en píxeles del viewport.

window.open(url, nombre, configuración);
Método que carga un recurso en el contexto de un nuevo navegador (pestaña o ventana nueva). Los argumentos son opcionales, si no los tiene abre una pestaña en blanco (**about: blank**). El argumento **url** especifica qué página se va a cargar, **nombre** especifica el nombre del nuevo recurso y puede ser usado como el destino de enlaces o formularios usando los atributos de elementos **<a>** o **<form>** (no debe tener espacios en blanco), y **configuración** es un string que lista las características del nuevo recurso como **size**, **position**, **scrollbars**, etc.

window.history
Propiedad que retorna un objeto que hace referencia al Historial, el cual provee de una interfaz para operar el historial de sesión del navegador, es decir, las páginas visitadas por la ventana o pestaña actual.

- **.history.back()**: método que hace que el navegador retroceda a una página en el historial de la sesión.
- **.history.forward()**: método que hace que el navegador avance a una página en el hostil de la sesión.

window.screen
Propiedad que retorna una referencia al objeto de la pantalla asociada a la ventana.

- **.screen.height**: devuelve la cantidad de píxeles de alto.
- **.screen.width**: devuelve la cantidad de píxeles de ancho.
- **.screen.colorDepth**: devuelve la profundidad de color en bits.

window.location
Propiedad que retorna un objeto con información acerca de la ubicación actual del documento.

- **.location.href**: retorna un string de la url y permite también actualizarla.

Element.addEventListener(tipo, listener, opciones)
Método que registra un evento a un elemento específico. Argumentos:

- **tipo**: obligatorio, es el tipo de evento (click, keydown, etc.)
- **listener**: obligatorio, es el método que se ejecutaba al ocurrir el evento.
- **opciones**: opcional, objeto de propiedades adicionales para controlar cómo se maneja el evento (capture, once, passive y signal).

navigator.cookiesEnable
Propiedad que retorna un booleano que indica si las cookies están habilitadas o no.

navigator.appCodeName
Propiedad deprecada, devuelve un string con el nombre del navegador (devuelve siempre Mozilla).

navigator.platform
Propiedad deprecada, devuelve un string con el nombre del sistema operativo.

Para ambos casos utilizar la interfaz **userAgent**.

5. Cookies
    5.1. Cookies

(Ver Archivo)
Son archivos o datos pequeños sobre la visita de una página que tiene información que se permite guardar con cierta persistencia en el navegador, con la particularidad que se pueden enviar a los servidores de las páginas web (no así los datos de localStorage y sessionStorage).

Son pequeños archivos de texto que se almacenan en el navegador y se envían con cada solicitud al servidor.

document.cookie
Se obtienen y definen las cookies asociadas con el documento en una cadena que contiene una lista de todas las cookies separadas por punto y coma (en pares *clave=valor*).

document.cookie = string
El valor **string** es una cadena de la forma *clave=valor*.

window.confirm(mensaje)
Indica al navegador que muestre una ventana de diálogo con un **mensaje** opcional, y que espere hasta que el usuario acepte o cancele la ventana de confirmación.

6. Checkpoint de conocimientos

- ¿Cuál es la diferencia entre document.querySelector() y document.querySelectorAll() en JavaScript?

querySelector() devuelve el primer elemento que coincide con el selector, mientras que querySelectorAll() devuelve todos los elementos que coinciden.

- ¿Qué sucede cuando se ejecuta window.alert() en un navegador web

Se muestra un cuadro de diálogo modal que bloquea el resto de la interacción hasta que se cierra.

- ¿Cuál es el propósito principal del BOM (Browser Object Model) en el desarrollo web?

Interactuar con el navegador y realizar tareas como la manipulación de la ventana y el historial del navegador.

- En el contexto de los eventos de teclado en JavaScript, ¿cuál es la diferencia principal entre keydown y keyup?

keydown se activa cuando se presiona una tecla, mientras que keyup se activa cuando se suelta una tecla.

- ¿Qué es y para qué se usa una cookie en el desarrollo web?

Un pequeño archivo de texto que un servidor web puede guardar en el navegador del usuario, utilizado principalmente para mantener la sesión del usuario y recordar información específica.