***JAVASCRIPT MID LEVEL***

**CAPÍTULO 7**

**- Introducción a Window**

* **Hereda las propiedades de EventTarget**
* **window.open( ) –** abre una nueva ventana.
* **window.close( ) –** cierra la ventana actual, o la ventana que llamó.
* **window.closed –** indica si la ventana referenciada está cerrada o no.
* **window.stop( ) –** detiene la carga de recursos en el contexto de navegación actual.
* **window.alert( ) –** muestra un cuadro de alerta en el contenido especificado y un botón Aceptar.
* **window.print( ) –** Abre un cuadro de diálogo Imprimir para imprimir en el documento actual.
* **window.prompt( ) –** Abre un cuadro de dialogo con un mensaje que solicita al usuario un dato (String).
* **window.confirm( ) –** Abre un cuadro de diálogo con un mensaje y dos botones: Aceptar y Cancelar.
* **window.screen( ) –** Devuelve una referencia al objeto de pantalla asociado con la ventana.
* **window.screenLeft( ) –** Devuelve la distancia horizontal entre el borde izquierdo del navegador y el borde izquierdo de la pantalla.
* **window.screenTop( ) –** Devuelve la distancia vertical entre el borde superior del navegador y el borde superior de la pantalla.
* **window.scrollX( ) –** Devuelve el número de pixeles que el documento se desplaza actualmente horizontalmente.
* **window.scrollY( ) –** Devuelve el número de pixeles que el documento se desplaza actualmente verticalmente.
* **window.scroll( ) –** Desplaza la ventana a un lugar particular en el documento. (con options y con posiciones).
* **resizeBy( ) –** Cambia el tamaño de la ventana actual en una cantidad específica.
* **resizeTo( ) –** Redimensiona dinámicamente la ventana.
* **moveBy( ) –** Mueve la ventana en un ubicación relativa.
* **moveBy( ) –** Mueve la ventana en un ubicación absoluta

**-- Ojetos barprop**. (Devuelve un valor boleano);

­­\_\_ locationbar.visible

­­\_\_ menubar.visible

\_\_ personalbar.visible

\_\_ scrollbar.visible

\_\_ statusbar.visible

\_\_ toolbar.visible

* **- Location**
* **window.location.href** – Devuelve el hreft (URL) de la página actual.
* **window.location.hostname –** Devuelve el nombre de dominio del servidor web
* **window.location.pathname** ­– Devuelve la ruta y el nombre de archivo de la página actual.
* **window.location.protocol** – Devuelve el protocolo web utilizado (http: o https;)
* **window.location.assign( )** – Carga un nuevo documento.

**CAPÍTULO 9**

**- Eventos**

* **Definición –** Los eventos son cualquier acción producida en la página ya sea por la interacción del usuario (por ejemplo, al pulsar una tecla, al mover el mouse, al hacer clic, etc.) o por la misma página (como cuando carga, cuando deja de cargar, cuando crea lo elementos, cuando se cierra, etc.).
* **Escucha de eventos o “Event Listener”–** Se refiere a un método que como lo indica, escucha el evento que le indiquemos, como primer parámetro se introduce el evento que se va a escuchar y como segundo parámetro el código a ejecutar mediante una función ya sea flecha o básica. No se puede usar parámetros en las funciones que llamemos, excepto la del objeto Event.

**Ej:** const zorrita = document.querySelector(".zorrita");

zorrita.addEventListener("click", saludar);

function saludar(){

alert("Hola");

};

**Ej2:** const zorrita = document.querySelector(".zorrita");

zorrita.addEventListener("click", ()=>{

alert("Hola");

});

* **El objeto Event –** Se refiere al objeto que está ejecutando un evento (this).
* **Flujo de Eventos vs “Event Flow” –** Se refiere a la forma en la que se producirá un evento en un objeto que está dentro de otro y este otro también tiene un evento.
* **Event Bubbling vs Event Capturing –** Bubbling se refiere a la forma en que se producirá un evento en un objeto que está dentro de otro que también tiene un evento, se producirá primero el evento del último hijo que tenga el evento, y como consiguiente los eventos de los padres que tengan un evento en orden hasta el último. Capturing en cambio, al agregar el valor true al evento de un objeto, este se producirá primero y luego se producirán los demás. Si varios tienen el valor true, estos se producirán en el orden que estén escrito el código.
* **event.stopPropagation() –** Detiene la propagación, es decir deja de ejecutar los eventos que sigan después de darle esta instrucción.

**- Eventos de mouse**

* **click** – ocurre con un click.
* **dblclick** – ocurre con un doble click.
* **mouseover** – ocurre cuando el puntero se mueve sobre un elemento o sobre uno de sus hijos.
* **mouseout** – ocurre cuando se mueve el puntero fuera de un elemento o de sus elementos secundario.

----- Otros -----

* **mouseenter** – ocurre cuando el puntero se mueve sobre un elemento.
* **mousemove** – ocurre cuando el puntero se mueve mientras está sobre un elemento.
* **mouseleave** – ocurre cuando el puntero se mueve fuera de un elemento.
* **mousedown** – ocurre cuando el usuario aprieta un botón del mouse sobre un elemento.
* **mouseup** – ocurre cuando un usuario suelta un botón del mouse sobre un elemento.

**- Eventos de teclado**

* **keydown** – ocurre cuando se presiona una tecla.
* **keyup** – ocurre cuando se deja de presionar una tecla.
* **keypress** – ocurre cuando una vez que keydown y keyup se han producido.

**- Eventos de la interfaz**

* **error** – ocurre cuando sucede un error durante la carga de un archivo multimedia.
* **load** – ocurre cuando un objeto se ha cargado.
* **beforeunload –** ocurre antes de que el documento esté a punto de descargarse.
* **unload -**  ocurre una vez que se ha descargado un página.
* **resize –** ocurre cuando se cambia el tamaño de la vista del documento.
* **scroll –** ocurre cuando se desplaza la barra de desplazamiento de un elemento.
* **select –** ocurre después de que el usuario selecciona algún texto de <input> o <textarea>

**- Timers (Temporizadores)**

* **setTimeout() –** Crea un temporizador que recibe como primer parámetro el código a ejecutar en una función, y como segundo parámetro el tiempo que queremos que transcurra para inicializar dicho código.
* **setInterval() –** Crea un temporizador tipo intervalo, que recibe como primer parámetro el código a ejecutar en una función, y como segundo parámetro el tiempo que queremos que transcurra entre cada ejecución del código (intervalo).
* **clearTimeout() –** Elimina un temporizador (almacenado en una variable).
* **clearInterval() –** Elimina un temporizado tipo intervalo (almacenado en un variable)

**Ej:** // ========== TEMPORIZADOR ==========

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (setTimeout)

setTimeout(()=>{

document.write("Hola")

},2000);

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (setInterval)

const intervalo = setInterval(()=>{

document.write("Hola")

},2000);

setTimeout(()=>{

clearInterval(intervalo)

},7000)

**CAPÍTULO 10**

**- Control de flujo y Manejo de errores**

* **Sentencias de bloque –** Se refiere al código escrito en un bloque, los bloques permiten declarar nuevamente una variable que no importa que ya se haya declarado fuera de este, y es que esta variable dentro de este bloque solo existirá dentro del mismo.
* **Sentencias de control de flujo –** Se refiere al código escrito en un bloque que solo se va a ejecutar si se cumple una condición. Con esto básicamente nos referimos a los condicionales que ya hemos visto (if, else, else if, etc).
* **Sentencias de manejo de excepciones –** Se refiere al control que se toma sobre los errores causados por alguna falla o algo inesperado en el código, a través de instrucciones en el código para cuando se produce el error.

**\* Flujo:** Se refiere a las instrucciones que ejecuta el programa una por una.

**- Sentencia Switch**

Es como un if, si se cumple una condición ejecutará la instrucción especificada, solo que esta lo hace a través de casos, en caso de que se cumpla una condición ejecuta la instrucción.

* **Sintaxis y clausula case –** La sintaxis se puede apreciar en el ejemplo, la clausula case se refiere al código que va a ejecutar en caso de que se cumpla la condición especificada en la cláusula case.
* **Break ­–** Detiene la ejecución del case en el que se especifique el break cuando se cumpla una de las condiciones, y es que si no se especifica el break entonces el case se ejecutará.
* **Default ­-** Es como un else, si no se cumple ninguna de las condiciones ejecuta la instrucción que se le especifique.

**Ej:**

**// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (switch);**

let exp = "Manzana";

switch(exp) {

case "Banana":

console.log("Soy de color amarillo");

alert("es banana");

break

case "Pera":

console.log("Tengo forma de cul\*");

alert("es pera");

break

case "Manzana":

console.log("Los primeros humanos pecaron comiendola");

alert("es manzana");

break

default:

console.log("No es ninguna XD");

alert("No es ninguna")

};

**- Excepciones y tipos de Excepciones**

* **Definición y usos –** Se refiere a un error que surgió en el programa, algo salió mal durante la ejecución del código o al no salió como esperábamos. Sirven para que el desarrollador sepa que hubo un error y puedo solucionarlo y/o hacer algo con ellos. El desarrollador también puede crear errores para avisar al usuario o a otros desarrolladores. (excepciones en la documentación oficial, developer).
* **Excepciones ECMAScript –** Son las excepciones de la tecnología que está detrás de JavaScript (ECMAScript).
* **DOMException y DOMError –** Son las excepciones y errores en el DOM.

**- Sentencia de manejo de excepciones Try…Catch.**

En caso de que se produzca un error durante la ejecución del programa, es ahí donde entra try y catch para indicar el error y lo que se hará con ese error. El try es un bloque que viene de la mano con catch, a diferencia del if con el else, si no hay catch o finally, no funciona. No es recomendado usarlo como mucha frecuencia, se usa para casos muy específicos.

* **Sintaxis** – Ejemplo de sintaxis a continuación, los errores de sintaxis no están contemplados para trabajar en estos casos, más bien está contemplado trabajar con errores en código que el programador espera, como errores en una variable, una función etc.
* **catch condicional** - Se refiere a una condición dentro de un catch, es decir, se va a ejecutar el bloque de código especificado en base a una condición. Básicamente es un if dentro de un catch.
* **catch incondicional** – Se refiere a un catch que no tiene condicional, es decir, que no incluirá una condición para que se ejecute un bloque de código.
* **finally ­–** Es una sentencia que se va a ejecutar siempre, se produzca o no el error, en el caso de retornar un valor, se ejecuta igual “sobrescribiendo” el return.
* **throw –** Devuelve una excepción definida en él, puede ser un tipo de dato, una variable, un array, un objeto, etc.

**Ej:**

**// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (try...catch);**

try {

lajdkflsdkl;

}

catch(error) {

console.log(error);

console.log("Lo siento ocurrió un error de referencia");

console.log(typeof error);

}

**- Desventajas de trabajar de manera obsoleta**

- - ¿Cuándo algo se vuelve obsoleto?

* **“Deprecated”**– Significa obsoleto, está en la documentación oficial, es algo que no se usa más o va a ser reemplazado.
* **Inutil** – Se refiere algo que removieron del lenguaje y por ende ya no sirve porque no existe, es inútil.
* **No recomendado** – Pues eso, es obsoleto porque no es recomendado, ya sea porque consume muchos recursos, o por otras razones.
* **Con bugs o fallos** – Tiene fallos.
* **Está por ser reemplazado.**
* **Hay mejores formas de hacerlo.**

- - Los efectos negativos (aplicado a métodos, clases y propiedades)

* **Uso excesivo de recursos.**
* **Código con Bugs o Fallos.**
* **Código innecesariamente largo.**
* **SEO.**

- - ¿Cómo verificar si está obsoleto?

\* Dato: 1 de cada 3 WEBS utiliza librerías de JavaScript obsoletas.

* **Verificar si tienen o usan librerías de JavaScript obsoletas.**
* **Verificar en los sitios basados en estándares oficiales como** developer.mozilla o w3school.com**.**

**- ¿Cómo verificar navegadores obsoletos?**

* **Detección del navegador y de cobertura a múltiples navegadores.** (copiar y pegar en el buscador).

**CAPÍTULO 12**

**- Control de flujo y Manejo de errores**

* **Detección del…**