4. Java Script

Javascript es un lenguaje interpretado usado para múltiples propósitos pero solo considerado como un complemento hasta ahora. Una de las innovaciones que ayudó a cambiar el modo en que vemos Javascript fue el desarrollo de nuevos motores de interpretación, creados para acelerar el procesamiento de código. La clave de los motores más exitosos fue transformar el código Javascript en código máquina para lograr velocidades de ejecución similares a aquellas encontradas en aplicaciones de escritorio. Esta mejorada capacidad permitió superar viejas limitaciones de rendimiento y confirmar el lenguaje Javascript como la mejor opción para la web.

4.1. Incorporando javascript

Siguiendo los mismos lineamientos que en CSS, existen tres técnicas para incorporar código Javascript dentro de HTML. Sin embargo, al igual que en CSS, solo la inclusión de archivos externos es la recomendada a usar en HTML5.

- **Inline.-** Esta es una técnica simple para insertar Javascript en nuestro documento que se aprovecha de atributos disponibles en elementos HTML. Estos atributos son manejadores de eventos que ejecutan código de acuerdo a la acción del usuario. Los manejadores de eventos más usados son, en general, los relacionados con el ratón, como por ejemplo **onclick**, **onMouseOver**, u **onMouseOut**.

```
<!DOCTYPE html>
<html lang=”es”>
	<head>
		<title>Este texto es el título del documento</title>
	</head>
	<body>
		<div id=”principal”>
			<p onclick=”alert(‘hizo clic!’)”>Hacer Clic</p>
			<p>No puede hacer clic</p>
		</div>
	</body>
</html>
```

- **Internas.-** Para trabajar con códigos extensos y funciones personalizadas debemos agrupar los códigos en un mismo lugar entre etiquetas **<script>**. El elemento **<script>** actúa exactamente igual al elemento **<style>** usado para incorporar estilos CSS. Nos ayuda a organizar el código en un solo lugar, afectando a los elementos HTML por medio de referencias.

```
<!DOCTYPE html>
<html lang="es">
	<head>
	<title>Este texto es el título del documento</title>
		<script>
			function mostraralerta()
			{
				alert('hizo clic!');
			}
			function hacerclic()
			{
				document.getElementsByTagName('p')[0].onclick=mostraralerta;
			}
			window.onload=hacerclic;
		</script>
	</head>
	<body>
		<div id=”principal”>
			<p>Hacer Clic</p>
			<p>No puede hacer Clic</p>
		</div>
	</body>
</html>
```

- **Externas.-** Los códigos Javascript crecen exponencialmente cuando agregamos nuevas funciones y aplicamos algunas de las APIs mencionadas previamente. Códigos embebidos incrementan el tamaño de nuestros documentos y los hacen repetitivos (cada documento debe volver a incluir los mismos códigos). Para reducir los tiempos de descarga, incrementar nuestra productividad y poder distribuir y reusar nuestros códigos en cada documento sin comprometer eficiencia, recomendamos grabar todos los códigos Javascript en uno o más archivos externos y llamarlos usando el atributo **src**.

```
<!DOCTYPE html>
<html lang="es">
	<head>
		<title>Este texto es el título del documento</title>
		<script src=”micodigo.js”></script>
	</head>
	<body>
		<div id=”principal”>
			<p>Hacer Clic</p>
			<p>No puede hacer Clic</p>
		</div>
	</body>
</html>
```

4.2. Funciones.

Las funciones son uno de los pilares fundamentales en JavaScript. Una función es un procedimiento en JavaScript, un conjunto de sentencias que realizan una tarea o calculan un valor. Para usar una función, debe definirla en algún lugar del ámbito desde el cual desea llamarla.

La **definición de una función** (también llamada **declaración de función** o **sentencia de función**) consiste de la palabra clave (reservada) function, seguida por:

- El nombre de la función (opcional).
- Una lista de argumentos para la función, encerrados entre paréntesis y separados por comas (,).
- Las sentencias JavaScript que definen la función, encerradas por llaves, { }.

```
function square(number)
{
  return number * number;
}
```

La función square toma un argumento, llamado number. La función consiste de una sentencia que expresa el retorno del argumento de la función (el cual es, number) multiplicado por sí mismo. La sentencia return específica el valor retornado por la función.

Funciones definidas en javascript

- **Windows.prompt(“Cadena”).-** Es una función de entrada de datos, esto nos permitirá ingresar datos a nuestro programa. Detiene el flujo de la ejecución del programa.
- **Windows.alert(“Cadena”).-** Es una función que nos permite mostrar mensajes de alerta en una página web. Detiene el flujo de la ejecución del programa.
- **Document.write(“Cadena”).-** Esta función nos permite insertar texto dentro de código HTML.
- **typeof(valor).-** Retorna el tipo de dato que tiene valor.
- **paseInt(“Cadena”).-** Transforma una cadena en entero siempre que se pueda.
- **isNaN(Valor).-** Retorna true si valor no es número y false si valor es un número

4.3. Tipos de datos.

- **Numeros.-** números enteros o decimales

- **String.-** Cadena de caracteres
- **Boolean.-** Son valores que solo pueden ser true o false

4.4. Variables

Espacio en la memoria del ordenador (RAM) donde se almacenará un valor que podrá cambiar durante la ejecución del programa.

- **Iniciar una variable.** Iniciar una variable es cuando le das un valor a la variable, tener en cuenta para poder iniciar una variable debe ser declarada antes.

```
var puntuación; //Declarar una variable
puntuación=0; //Iniciar una variable
```

```
var puntuación=0; //declarar e iniciar una variable
```

- **Declaración de varias variables en la misma línea.**

```
var puntuación, record, jugador;
var puntuación=500, record=800, jugador="Rubén";
```

- **Características de las variables**
  - Han de comenzar por letra, símbolo $ o _

```
nombreVariable
&nombreVariable
_nombreVariable
```

  - Han de contener solo letras, números, $ y _

```
nombreVariable1
nombreVariable_1
nombre$Varibale_2
```

  - Son case sensitive, es decir que diferencia entre minúsculas y mayúsculas

```
nombreVariable
nombrevariable
NombreVariable
```

  - No deben ser palabras reservadas

```
var alert;          No podemos usar alert como nombre de variable
```

  - Se recomienda que sean descriptivos

4.5. Operadores

- **Básicos.-** Suma (+), resta (-), multiplicación (*) y división (/).
- **Incremento / decremento.-**
  - (+=) Incrementa en x el valor de una variable. nombreVariable+=10; también se puede usar nombreVariable=nombreVariable+10;
  - (++) Incrementa en 1 el valor de una variable. nombreVariable++;
  - (-=) Decrementa en x el valor de una variable. nombreVariable-=10; también se puede usar nombreVariable=nombreVariable-10;
  - (--) Decrementa en 1 el valor de una variable. nombreVariable--;
  - (*=) Multiplica en x el valor de una variable. nombreVariable*=10;

- (/=) Divide en x el valor de una variable. nombreVariable/=10;

4.6. Arrays o arreglos

- **Creación de arreglos.-** Para crear una arreglo en Javascript usamos la siguiente sentencia, var nombreArreglo=[“valor1”,”valor2”,”valor3”,”valor4”]; o también podemos usar, var nombreArreglo=new Array(“valor1”,”valor2”,”valor3”,”valor4”);. Para declarar un arreglo vacío usamos la siguiente sentencia: nombreArreglo=[];
- **Obtener un elemento del arreglo.-** Un arreglo en Javascript es manejado por índices, estos son conocidos como posiciones. El primer elemento tiene la posición “cero”
- **Propiedades de un arreglo**
  - **Propiedad Length.-** Nos dice cuál es la longitud o números de elementos que tiene de un arreglo. Sintaxis: nombreArreglo.length
  - **Método push.-** Nos permite agregar un elemento al final del arreglo. Sintaxis : nombreArreglo.push(“valor”);
  - **Método unshift.-** Nos permite agregar un elemento al inicio o comienzo de un arreglo. Sintaxis: nombreArreglo.unshift(“valor”);
  - **Método pop.-** Nos permite eliminar el último elemento del arreglo. Sintaxis: nombreArreglo.pop();
  - **Método shift.-** Nos permite eliminar el primer elemento del arreglo. Sintaxis: nombreaArreglo.shift();

4.7. Programación Orientado a Objetos (POO).- Programar utilizando objetos significa que siempre podremos usar los atributos y métodos de un objeto. En Javascript al igual que cualquier otro lenguaje que está basado en POO, es posible modificar dichos objetos.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <input type="button" id="boton"/>
    <input type="button" id="boton1"/>
    <input type="text" id="cuadroTexto"/>

    <script>
        var objetoBoton=document.getElementById("boton");
        objetoBoton.style.width="100px";
        objetoBoton.style.height="90px";
        /*objetoBoton.style.backgroundColor="blue";*/
        var objetoBoton1=document.getElementById("boton1");
        objetoBoton1.style.width="100px";
        objetoBoton1.style.height="90px";
        var objetoCuadroText=document.getElementById("cuadroTexto");
        objetoCuadroText.style.background="#fff";
        objetoCuadroText.focus();
    </script>
</body>
</html>
```

4.8. Estructuras de control de flujo “Condicionales”

- **Operadores para realizar condiciones**

Operador	Significado
==	Igual que …
!=	Diferente o distinto que …
>	Mayor que …
<	Menor que …
>=	Mayor o igual que …
<=	Menor o igual que …
===	Estrictamente igual que … (comprueba igual tipos)
!==	Estrictamente diferente que … (comprueba igual tipos)
&&	Y lógico
||	O lógico

- **If.-** La condición if then else, es como crear dos caminos al momento de escribir código, es decir si cumple la condición hacer algo y si no cumple la condición hacer algo.

Estructura de if then else

```
If(condición)
{
        //Código a ejecutar si la condición se cumple
}
Else
{
        //Código a ejecutar si la condición no es verdad
}
```

Estructura usando solo if

```
If(condición)
{
        //Código a ejecutar si la condición se cumple
}
```

4.9. Estructuras de control de flujo “Bucles”

- **While.-** Una condición while es usando de forma que se repita tantas veces que la condición no cambie. Este tipo de bucles pueden ser de tamaño fijo o un tamaño cambiante.

```
While(condición)
{
        //Código a repetir mientras la condición del bucle se cierta.
}
```

- **For.-** Un bucle que tiene un tamaño definido.

```
for(var a=0;a<valor;a++)
{
        //código a repetir mientras cumpla la condición (a<valor).
}
```

- **Do while.-** Funciona igual que un while, a diferencia que se ejecuta el código una vez sin cumplir la condición del while.

```
do
{
        //código a repetir mientras la condición del bucle sea cierta.
}
While(condición);
```