Resumen de Desarrollo Web en Entorno Cliente (DWEC)

Resumen de PDF

1 Introducción a Javascript

Este tema cubre los conceptos básicos de cómo funciona Javascript en el contexto web y cómo organizar los archivos.

[Image of client-server web architecture]

- **Entorno Cliente vs. Servidor:**
  - **Servidor:** Ejecuta lenguajes como PHP, ASP, JSP, etc. Procesa la petición y envía la respuesta.
  - **Cliente:** Es el navegador web. Ejecuta HTML, CSS y Javascript (JS).
- **Protocolos:** Se mencionan `http://`, `https://` (seguro) y `file:///` (para archivos locales).
- **Inclusión de Javascript en HTML:** Hay tres formas principales:
  1. **En un elemento `<script>`:** Dentro del `<head>` o `<body>`.
  2. **En un archivo externo `.js`:** Es la forma recomendada.
  3. **Como atributo en un elemento (inline):** Por ejemplo, `onclick=""`. No es una práctica recomendada.
- **Organización de archivos:** Se propone una estructura básica para organizar el proyecto:
  - `index.html` (y otros `.html`) en la raíz.
  - Carpeta `css/` para los estilos.
  - Carpeta `js/` para los scripts.
  - Carpeta `images/` para las imágenes.

2 Sintaxis Básica de Javascript

Aquí se definen las reglas fundamentales del lenguaje, cómo declarar variables, los operadores y las estructuras de control.

2.1 Variables y Tipos de Datos

- **Reglas:** Javascript es sensible a mayúsculas y minúsculas ("key sensitive") y no tiene en cuenta los espacios en blanco. El punto y coma (`;`) al final de la instrucción no es estrictamente necesario, pero es una buena práctica.
- **Declaración de variables:**
  - `var`: Declara una variable de ámbito global.
  - `let`: Declara una variable de ámbito de bloque (local).
  - `const`: Declara una constante de ámbito de bloque, solo de lectura.

```
var variableGlobal = 1; //  mbito  global o de funci n
let variableBloque = 2; //  mbito  de bloque (dentro de un if, for, etc.)
const PI = 3.1416;     // Constante, no se puede reasignar
```

- **Hoisting:** Es un comportamiento de Javascript donde las declaraciones (pero no las inicializaciones) se "mueven" al principio de su ámbito. Solo funciona con `var`.

```
console.log(miVar); // Imprime 'undefined' (por Hoisting)
var miVar = 0;

console.log(miLet); // Lanza una Excepci n (ReferenceError)
let miLet = 0;
```

- **Tipos de Datos Primitivos:**
  - `Undefined`
  - `Boolean` (true/false)
  - `Number`
  - `String`
  - `BigInt`
  - `null`
  - `Object`
  - `Function`
- **Valores Falsy:** Valores que se interpretan como `FALSE` en un contexto booleano:
  - `false`
  - `undefined`
  - `null`
  - `0`
  - `NaN` (Not a Number)
  - Cadena vacía `""`

2.2 Operadores

- **Aritméticos:** Suma (+), Resta (-), Multiplicación (*), División (/), Resto (%), Potencia (**), Incremento (++), Decremento (-).
- **Asignación:** Asignación (=), Adición (+=), Sustracción (-=), etc.
- **Comparación:**
  - Igualdad (solo valor): `1 == "1"` (true).
  - Estrictamente iguales (valor y tipo): `1 === "1"` (false).
  - Desigualdad: `!=`, `!==` (estricto).
  - Otros: `>`, `<`, `>=`, `<=`.
- **Lógicos:** `&&` (AND), `||` (OR), `!` (NOT).
- **Ternario:** Es un `if` en una sola línea.

```
// condici n ? SI_se_cumple : NO_se_cumple
let edad = 20;
let mensaje = (edad >= 18) ? "Es mayor de edad" : "Es menor de edad";
```

- **typeof:** Devuelve el tipo de dato de una variable.

2.3 Estructuras de Control y Bucles

- **Condicionales:**
  - `if ... else if ... else`
  - `switch`: Para múltiples comparaciones contra un valor.
- **Bucles (Repetitivas):**
  - `while`: Se ejecuta mientras la condición sea verdadera.
  - `do...while`: Se ejecuta al menos una vez, y luego comprueba la condición.
  - `for`: El bucle estándar con inicialización, condición e incremento.
  - `for...in`: Recorre las **propiedades** de un objeto (índices de un array).
  - `for...of`: Recorre los **valores** de un objeto iterable (elementos de un array).
- **Alteración de Flujo:**
  - `break`: Sale del bucle o `switch`.
  - `continue`: Salta a la siguiente iteración del bucle.
  - `label`: Identificador para usar con `break` o `continue` en bucles anidados.

2.4 Interacción Provisional

Funciones básicas para interactuar con el usuario (no recomendadas para aplicaciones finales):

- `alert()`: Muestra un mensaje.
- `confirm()`: Muestra un mensaje con botones Aceptar/Cancelar y devuelve `true` o `false`.
- `prompt()`: Pide al usuario que introduzca un dato.

3 Tipos de Datos Complejos (Objetos)

Manejo de estructuras de datos más complejas como Arrays, Strings y Objetos predefinidos.

3.1 Arrays (Arreglos)

Colección ordenada de elementos.

- **Declaración:**

```
let arr1 = new Array("a", "b", "c");
let arr2 = ["a", "b", "c"]; // Forma preferida
```

- **Matrices:** Son arrays que contienen otros arrays.
- **Métodos Destructivos** (modifican el array original):
  - `push()`: Añade elementos al **final**.
  - `pop()`: Elimina y devuelve el elemento del **final**.
  - `unshift()`: Añade elementos al **inicio**.
  - `shift()`: Elimina y devuelve el elemento del **inicio**.
  - `splice()`: Elimina, reemplaza o añade elementos en una posición específica.
  - `sort()`: Ordena los elementos.
  - `reverse()`: Invierte el orden de los elementos.
- **Métodos No Destructivos** (devuelven un nuevo array o valor):
  - `slice()`: Devuelve una copia superficial de una porción del array.
  - `concat()`: Une dos o más arrays.
  - `join()`: Une todos los elementos en un string, usando un separador.
  - `indexOf()`: Devuelve el primer índice de un elemento (o -1).
  - `lastIndexOf()`: Devuelve el último índice de un elemento (o -1).
  - `includes()`: Devuelve `true` si el array contiene el elemento.
- **Propiedades:**
  - `length`: Devuelve el número de elementos del array.

3.2 String (Cadenas de texto)

- **Literal vs. Objeto:** Un string literal (`"hola"`) es un tipo primitivo. Un `new String("hola")` es un objeto. Se debe usar la forma literal.
- **Concatenación:**

```
let var1 = "Mundo";
// Forma cl sica
let saludo1 = "Hola " + var1;
// Template Literals (Forma moderna)
let saludo2 = `Hola ${var1}`;
```

- **Métodos Útiles (resumen):** `charAt()`, `indexOf()`, `startsWith()`, `endsWith()`, `includes()`, `split()` (convierte a array), `slice()`, `substring()`, `toLowerCase()`, `toUpperCase()`, `trim()` (elimina espacios).

3.3 Number, Math y Date

- **Number:** Objeto envoltorio para números. Ofrece métodos estáticos como:
  - `Number.parseInt()`: Convierte un string a entero.
  - `Number.parseFloat()`: Convierte a decimal.
  - `Number.isNaN()`: Comprueba si un valor es "No es un Número".
  - `Number.isFinite()`: Comprueba si es un número finito.
- **Math:** Objeto estático con propiedades y métodos matemáticos.
  - Propiedades: `Math.PI`, `Math.E`.
  - Métodos: `Math.abs()` (valor absoluto), `Math.min()`, `Math.max()`, `Math.random()` (aleatorio entre 0 y 1), `Math.round()` (redondear), `Math.sqrt()` (raíz cuadrada).
- **Date:** Objeto para trabajar con fechas.
  - `let hoy = new Date();`
  - `Date.now()`: Milisegundos desde 1970.

4 Funciones, Módulos y Excepciones

Este tema cubre cómo organizar el código en bloques reutilizables (funciones), cómo dividirlo en archivos (módulos) y cómo manejar errores.

4.1 Funciones

Bloques de código que realizan una tarea.

- **Declaración de Función (Clásica):**

```
function nombre(p1, p2) {
  // Instrucciones
  return p1 + p2;
}
```

- **Expresión de Función (Anónima):**

```
let nombre = function(p1, p2) {
  // Instrucciones
  return p1 + p2;
};
```

- **Función Flecha (Arrow Function):** Una sintaxis más corta.

```
// Sintaxis con varias instrucciones
let nombre = (p1, p2) => {
  let resultado = p1 + p2;
  return resultado;
};

// Si solo tiene UNA instrucci n , el 'return' es impl cito
let nombreDirecto = (p1, p2) => p1 + p2;

// Si solo tiene UN par metro , los par ntesis son opcionales
let duplicar = p1 => p1 * 2;
```

4.2 Métodos Avanzados de Array (Funcionales)

Estos métodos usan funciones (a menudo flecha) como argumentos.

- `forEach()`: Ejecuta una función para cada elemento. No devuelve nada.
- `map()`: Crea un **nuevo array** aplicando una función a cada elemento.
- `filter()`: Crea un **nuevo array** solo con los elementos que pasan una condición.
- `reduce()`: Reduce el array a un solo valor (ej. una suma total).
- `some()`: Devuelve `true` si **al menos un** elemento cumple la condición.
- `every()`: Devuelve `true` si **todos** los elementos cumplen la condición.

```
const numeros = [1, 2, 3, 4, 5];

// map(): Duplicar cada n mero
const duplicados = numeros.map(n => n * 2);
// duplicados es [2, 4, 6, 8, 10] (ejemplo similar)

// filter(): Solo los pares
const pares = numeros.filter(n => n % 2 === 0);
// pares es [2, 4] (ejemplo similar)

// reduce(): Sumar todos
const suma = numeros.reduce((acumulador, n) => acumulador + n);
// suma es 15 (ejemplo similar)
```

4.3 Módulos (Importar y Exportar)

Permiten dividir el código en múltiples archivos. Requiere que el script HTML tenga `type="module"`.

- **Exportar (archivo.js):**
  - **Exportación con nombre:**

```
export const PI = 3.1416;
export function miFuncion() { ... }
```

  - **Exportación por defecto:** (Solo una por archivo)

```
export default function funcionDefecto() { ... }
```

- **Importar (archivo2.js):**

```
// Importar exports con nombre
import { PI, miFuncion } from "./archivo.js";

// Importar el export por defecto (puedes ponerle cualquier nombre)
import nombreQueQuiera from "./archivo.js";
```

4.4 Manejo de Excepciones (Errores)

- `try...catch`: Controla código que puede fallar.
- `finally`: Se ejecuta siempre, haya error o no.
- `throw`: Lanza (genera) una excepción personalizada.

```
try {
  // C digo a controlar
  let resultado = getMonthName(myMonth);
} catch (e) {
  // Se ejecuta si hay una excepci n
  logMyErrors(e);
} finally {
  // Se ejecuta siempre
  console.log("fin");
}
```

5 El DOM (Document Object Model)

El DOM es la representación en forma de árbol de un documento HTML, permitiendo a Javascript interactuar con él. Está compuesto por **nodos**.

- **Tipos de Nodo:**
  - `Document`: La raíz del documento.
  - `Element`: Las etiquetas HTML (ej. `<div>`, `<p>`).
  - `Attr`: Los atributos (ej. `id`, `class`).
  - `Text`: El contenido de texto de un elemento.
  - `Comment`: Un comentario HTML.

5.1 Selección de Nodos (Elementos)

Estos métodos se usan (generalmente desde `document`) para encontrar elementos en la página.

```
// Por ID (devuelve un solo elemento)
let div1 = document.getElementById("div1");

// Por nombre de etiqueta (devuelve una colecci n HTML)
let todosLosDivs = document.getElementsByTagName("div");

// Por nombre de clase (devuelve una colecci n HTML)
let movidas = document.getElementsByClassName("movidas");

// Por atributo 'name' (devuelve un NodeList)
let sexoRadios = document.getElementsByName("sexo");

// Por selector CSS (devuelve el PRIMER elemento que coincida)
let primerInput = document.querySelector("input.sexo");

// Por selector CSS (devuelve un NodeList con TODOS los que coincidan)
let todosLosInputs = document.querySelectorAll("input");
```

5.2 Manipulación de Atributos y Clases

Una vez seleccionado un elemento, podemos cambiar sus atributos.

```
let miInput = document.querySelector("#sexo");

// --- Atributos generales ---
// Leer un atributo
let tipo = miInput.type;
// Cambiar un atributo
miInput.name = "apellidos";

// Forma alternativa (funciona para atributos personalizados)
miInput.setAttribute("name", "apellidos");

// --- Estilos (style) ---
// La propiedad CSS 'border-color' se escribe 'borderColor' (camelCase)
miInput.style.borderColor = "red";

// --- Clases (className vs classList) ---
// .className (sobrescribe TODAS las clases)
miInput.className = "sexo";

// .classList (m todo preferido para gestionar clases)
miInput.classList.add("movidas", "sexo");
miInput.classList.remove("movidas");
miInput.classList.toggle("claseActiva"); // A ade si no est , quita si est
miInput.classList.item(0); // Devuelve la clase en el  ndice  0
```

5.3 Creación y Eliminación de Nodos

Podemos añadir o quitar elementos del DOM dinámicamente.

- **Crear y Añadir:**

```
// 1. Creamos el nuevo nodo (a n no est  en la p gina)
let hijo = document.createElement("div");

// 2. Seleccionamos el elemento que ser  su padre
let padre = document.getElementById("padre");

// 3. A adimos el nuevo nodo (hijo) dentro del padre
padre.appendChild(hijo);
```

- **Eliminar:**

```
// 1. Seleccionamos el nodo a eliminar
let hijo = document.querySelector("#hijo");

// Opci n A: A trav s del padre (forma cl sica)
let padre = hijo.parentNode;
padre.removeChild(hijo);

// Opci n B: Directamente (forma moderna)
hijo.remove();
```

6 Eventos

Un evento es una acción (como un clic, pasar el ratón por encima, pulsar una tecla) que el programa puede detectar.

6.1 Formas de Manejar Eventos

1. **Atributo HTML (Antiguo):**

```
<button onclick="alert('Pulsado')"></button>
```

2. **Manejador Semántico (JS):** Fácil, pero limitado a una sola función por evento.

```
let caja = document.getElementById("caja");
caja.onclick = procesaEvento;
```

3. **Event Listeners (Moderno):** La forma recomendada. Permite múltiples funciones para un mismo evento.

```
let miElemento = document.querySelector("#uno");

function miFuncion(evento) {
  console.log(evento); // 'evento' es el Objeto Event
}

// A adir el listener
miElemento.addEventListener("click", miFuncion);

// Eliminar el listener (debe ser la misma referencia a la funci n)
miElemento.removeEventListener("click", miFuncion);
```

6.2 Flujo de Eventos (Bubbling y Capturing)

Cuando un evento ocurre en un elemento (ej. un `<div>`), también ocurre en su padre (`<body>`), su abuelo (`<html>`), etc.

- **Event Bubbling (Burbujeo):** El evento se propaga desde el elemento más interno **hacia afuera** (hacia `window`). Es el comportamiento por defecto.
- **Event Capturing (Captura):** El evento se propaga desde el elemento más externo (`window`) **hacia adentro** (hacia el elemento).

Al usar `addEventListener(evento, funcion, flujo)`, el tercer parámetro (`flujo`) es opcional. Si es `false` (defecto) usa Bubbling. Si es `true`, usa Capturing.

6.3 El Objeto Event

Cuando se dispara un evento, la función que lo maneja recibe automáticamente un objeto `Event` como primer parámetro. Este objeto contiene toda la información sobre lo que ocurrió (ej. `evento.target` para saber qué elemento originó el evento, `evento.key` en un evento de teclado, etc.).