BOTIC-SOFOF-24-28-13-0081 GRUPO 2

Módulo 3: Fundamentos de Programación en JavaScript (60 Horas) / ELE069950089611

AE1_ABPRO-Ejercicio grupal [Actividad Evaluada]

AE1_ABPRO-Ejercicio grupal [Actividad Evaluada]

Ejercicio grupal

El navegador web no solo actúa como un visor de páginas, sino que representa un entorno de ejecución completo para JavaScript y un conjunto de herramientas esenciales para desarrolladores. Esta actividad tiene como propósito fomentar la comprensión compartida sobre cómo se ejecuta JavaScript en el navegador y qué herramientas están disponibles para facilitar el desarrollo, depuración y prueba de código.

Instrucciones

A través de una discusión colaborativa y una investigación conjunta, deberán desarrollar una presentación o documento que responda con claridad y profundidad las siguientes preguntas. Cada equipo presentará su análisis ante el grupo.

1. **¿Cómo actúa el navegador como entorno de ejecución de JavaScript?**
   Investiga qué motor de JavaScript usa el navegador asignado a tu equipo (Chrome, Firefox, Safari, etc.) y describe cómo procesa el código JavaScript. Mencionen también los elementos del navegador que forman parte de este entorno (DOM, BOM, event loop, etc.).

2. **¿Qué herramientas ofrece el navegador para apoyar el desarrollo?**
   Exploren las herramientas para desarrolladores que están integradas en su navegador. Describan las funciones principales de al menos tres paneles: Consola, Inspector, Red, Almacenamiento o Performance.

3. **¿Qué puede y qué no puede hacer JavaScript dentro del navegador?**
   Hagan una lista clara y argumentada de las posibilidades del lenguaje en este entorno (interacción con el DOM, manejo de eventos, acceso a almacenamiento local, etc.) y de sus limitaciones (por ejemplo, acceso directo al sistema de archivos o bases de datos locales sin intermediarios).

4. **¿Qué papel cumple la consola de comandos en el desarrollo y depuración de JavaScript?**
   Analicen su uso práctico, cómo se pueden hacer pruebas rápidas, leer errores o seguir el flujo de ejecución. Agreguen ejemplos hipotéticos o discusiones sobre cómo un desarrollador podría utilizarla.

5. **¿Qué otras herramientas externas existen para el desarrollo de JavaScript?**
   Investiguen al menos dos herramientas adicionales fuera del navegador (por ejemplo: Visual Studio Code, ESLint, Prettier, extensiones de Chrome, etc.) que

ayuden a desarrollar, formatear o depurar código JavaScript. Expliquen cómo se integran con el flujo de trabajo.

Al finalizar, cada equipo compartirá sus conclusiones y reflexiones con el resto del grupo, fomentando así el diálogo sobre buenas prácticas y herramientas clave en el desarrollo moderno con JavaScript en el entorno del navegador.

Entrega:

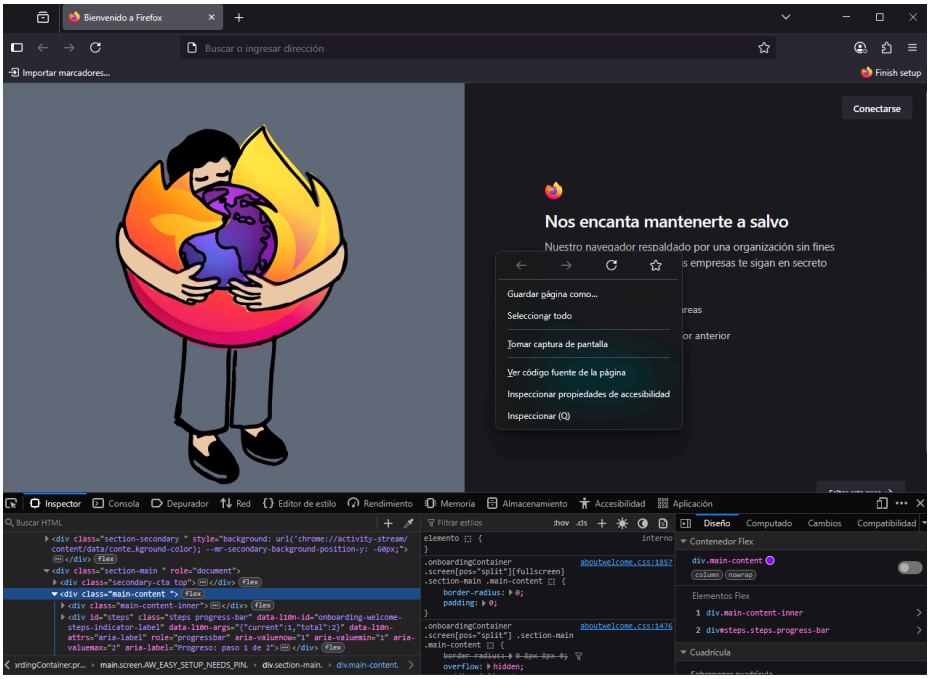
- Documento word o PDF con su análisis y respuestas.
- Duración: 1 jornada de clases.
- Ejecución: Grupal.

Estado de la entrega

Estado de la entrega	Todavía no se han realizado envíos
Estado de la calificación	Sin calificar
Última modificación	-

DESARROLLO AE1_ABPRO-Ejercicio grupal
GRUPO 2

1. ¿Cómo actúa el navegador como entorno de ejecución de JavaScript?



El navegador no es simplemente una ventana para visualizar páginas web, sino que es un entorno de ejecución para JavaScript. Integra un motor de interpretación, diversas APIs y un sistema de gestión de eventos, permitiendo que los scripts se ejecuten de forma eficiente, reactiva y en estrecha interacción con el contenido de la página.

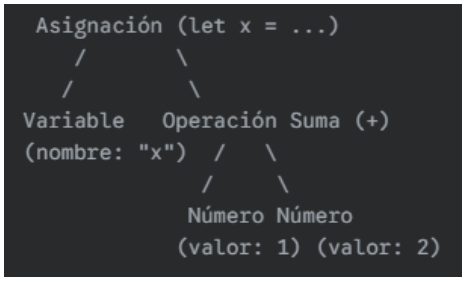
¿Qué motor usa tu navegador?
En Firefox, se utiliza SpiderMonkey. Este motor se encarga de convertir nuestro código JavaScript en instrucciones que la computadora puede entender. Primero lo interpreta y luego lo optimiza, transformándolo en lenguaje máquina justo en el momento en que se necesita. A este proceso se le llama compilación Just-In-Time.

También navegamos con otros exploradores que cuentan con motores distintos para ejecutar JavaScript. En Chrome, por ejemplo, se utiliza V8, mientras que en Safari emplea JavaScriptCore. Aunque cada motor tiene sus particularidades, todos comparten el objetivo de interpretar y ejecutar el código de forma eficiente.

El proceso clave de SpiderMonkey es el siguiente:

```
function sumar(a, b) {
  return a + b;
}
```

- El código JavaScript se **parsea** en un AST.



- El AST se convierte en **bytecode**.

```
; Bytecode conceptual para 'sumar'
LOAD_ARG 0
LOAD_ARG 1
ADD
RETURN
```

- El bytecode es ejecutado inicialmente por un **intérprete** rápido.

- Las partes del código que se ejecutan con frecuencia ("hot code") son **compiladas Just-In-Time** a código máquina por diferentes niveles de compiladores (Baseline JIT, WarpMonkey) para optimizar el rendimiento.

```
01010100 10001001 11100101
10001011 01000101 00001000
10000011 11000000 00000001
11000011
```

- Un **recolector de basura** gestiona automáticamente la memoria.

¿Qué hace posible que el navegador ejecute JavaScript?

Cuando escribimos código JavaScript, no se ejecuta en el vacío. El navegador nos ofrece un entorno con herramientas que permiten interactuar con la página, manejar eventos y comunicarse con el exterior de forma eficiente y dinámica. Estos son los componentes clave que hacen eso posible:

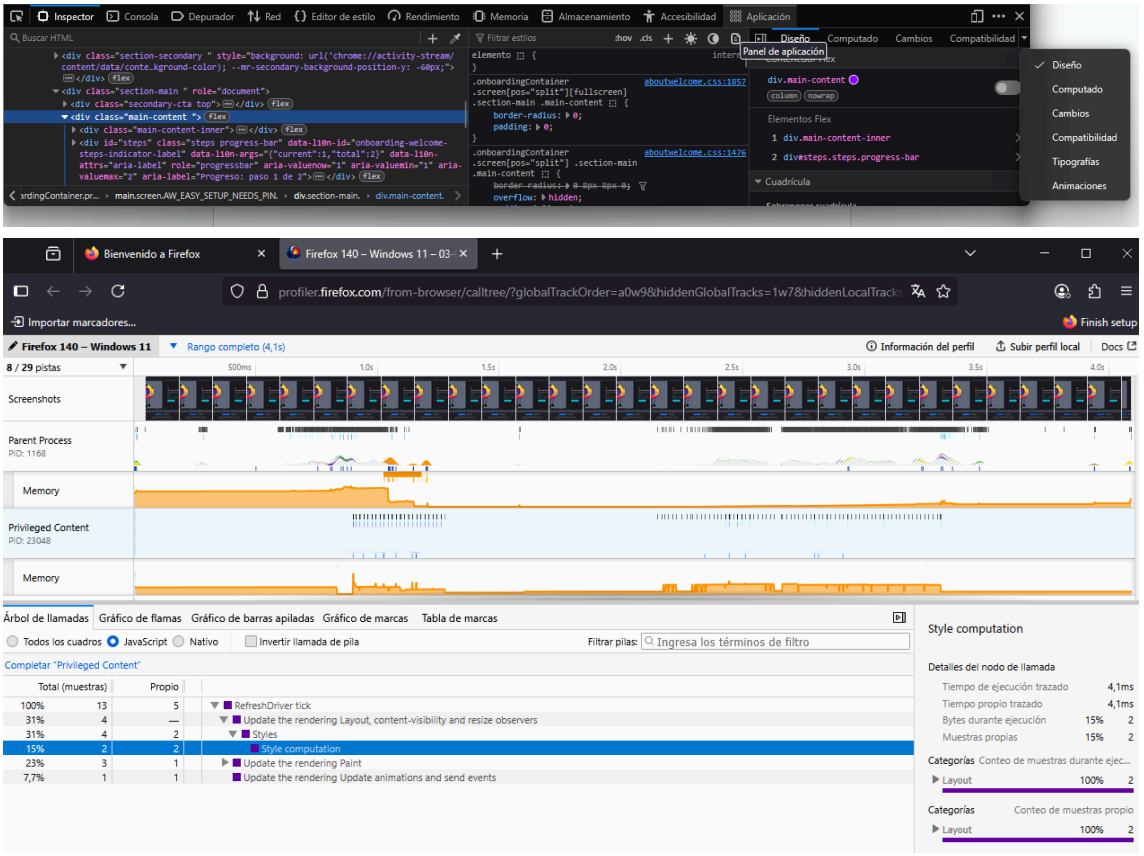
- ✔ DOM (Document Object Model): representa la estructura del documento HTML como un árbol. Podemos usar JavaScript para acceder a sus elementos y cambiar textos, estilos o atributos mientras la página está cargada.

- ✔ BOM (Browser Object Model): nos permite interactuar con el propio navegador, mostrar alertas, manejar el historial, modificar la barra de direcciones o abrir nuevas ventanas, todo a través del objeto ***window***.

- ✔ Event Loop: coordina la ejecución entre tareas síncronas y asíncronas. Mientras el código principal se ejecuta en el Call Stack, las operaciones asíncronas (como ***fetch*** o eventos) se gestionan en la Task Queue, y el Event Loop los pasa al Call Stack.

- ✔ APIs del navegador: son interfaces que nos brinda el navegador para tareas específicas. Incluyen temporizadores (***setTimeout***, ***setInterval***), manejo de almacenamiento (***localStorage***, ***sessionStorage***), solicitudes HTTP (***fetch***, ***XMLHttpRequest***) y manipulación del DOM y BOm mediante ***document*** y ***window***.

2. ¿Qué herramientas ofrece el navegador para apoyar el desarrollo?



Los navegadores modernos, como Chrome, incluyen herramientas integradas muy útiles para los desarrolladores. Se pueden abrir con clic derecho → "Inspeccionar" o con la tecla F12. Estas herramientas ofrecen varios paneles que permiten revisar el código, detectar errores y mejorar el rendimiento de una página web sin salir del navegador. A continuación, describiremos las tres principales:

- ✔ Consola: permite ver errores de JavaScript, mensajes personalizados con `console.log()` y probar fragmentos de código en tiempo real. También muestra advertencias, trazas de pila y facilita la inspección de objetos y arrays, todo sin necesidad de recargar la página.

- ✔ Inspector (Elements): muestra la estructura del DOM y los estilos CSS aplicados. Podemos editar el HTML y modificar propiedades visuales en vivo para ver cómo impactan el diseño, además de inspeccionar márgenes, padding, y reglas activas o heredadas.

- ✔ Red (Network): muestra todas las solicitudes que realiza la página (HTML, CSS, JS, imágenes, fetch, etc.), junto con su tiempo de carga, tamaño y estado. Es ideal para detectar archivos lentos o faltantes, y para analizar el rendimiento general. Incluye filtros y opciones para simular conexiones lentas.

3.

¿Qué puede y qué no puede hacer JavaScript dentro del navegador?

¿Qué sí puede hacer JavaScript en el navegador?

- ✔ Manipular el DOM: agregar, eliminar o modificar elementos de la página en tiempo real, lo que permite actualizar contenido, estilos o estructura sin recargar.

- ✔ Escuchar y responder a eventos: detectar interacciones del usuario como clics, teclas, movimientos del mouse, entre otros, y responde con funciones personalizadas.

- ✔ Almacenamiento de datos local: puede guardar información en el navegador mediante `localStorage`, `sessionStorage`, útil para recordar preferencias o mantener estados entre visitas.

- ✔ Hacer peticiones a servidores: utilizar `fetch` o `XMLHttpRequest` para enviar y recibir datos desde servicios externos (APIs), lo que permite cargar contenido dinámico sin recargar la página.

- ✔ Crear animaciones y temporizadores: permite ejecutar acciones después de cierto tiempo (`setTimeout`, `setInterval`), y crear efectos visuales con ayuda de CSS o Canvas.

¿Qué no puede hacer JavaScript en el navegador?

- ✔ Acceder directamente al sistema de archivos: no puede leer ni modificar archivos, a menos que el usuario seleccione uno manualmente (por ejemplo, con un `input`).

- ✔ Conectarse directamente a bases de datos locales: JavaScript no puede interactuar con bases de datos desde el navegador sin un servidor intermediario que maneje esa conexión.

- ✔ Ejecutar programas del sistema ni modificar configuraciones del dispositivo: no tiene permisos para afectar el sistema operativo, instalar software ni acceder a configuraciones del sistema.

4. ¿Qué papel cumple la consola de comandos en el desarrollo y depuración de JavaScript?

La consola del navegador es una herramienta clave para los desarrolladores. Funciona como un espacio de pruebas en tiempo real que permite:

- ✔ **Probar fragmentos de código rápidamente**, sin tener que guardarlos ni actualizar la página.

- ✔ **Leer errores y advertencias,** ya que muestra mensajes detallados cuando algo falla, ayudando a detectar problemas en el código.

- ✔ **Seguir el flujo de ejecución,** utilizando funciones como `console.log()`, `console.error()` o `console.table()` para monitorear el comportamiento del programa y verificar valores intermedios.

Ejemplo práctico: Supongamos que queremos comprobar si una variable está recibiendo correctamente el valor esperado:

```
let Sala = "Grupo 2";
console.log("Módulo 3: Fundamentos de Programación en JavaScript, está formado por el  " + Sala);
```

Este mensaje aparecerá en la consola y nos permitirá confirmar que la variable `sala` contiene el valor correcto.

Además, un desarrollador puede usar la consola para inspeccionar objetos, evaluar condiciones o probar funciones de forma interactiva, lo que la convierte en una aliada esencial para aprender como para depurar de forma efectiva.

5. ¿Qué otras herramientas externas existen para el desarrollo de JavaScript?

Además del navegador, existen herramientas externas que potencian el desarrollo, formateo y depuración del código JavaScript. Estas se integran fácilmente al flujo de trabajo y mejoran tanto la productividad como la calidad del código.

1. Visual Studio Code (VSCode): es uno de los editores de código más populares. Ofrece resaltado de sintaxis, autocompletado inteligente, integración con Git y soporte para ejecutar JavaScript con Node.js. Además, permite instalar extensiones como Prettier y ESLint para automatizar tareas comunes.

2. ESLint: es una herramienta de análisis estático que detecta errores y sugiere buenas prácticas. Por ejemplo, avisa si se usa una variable sin declarar o si hay inconsistencias en el estilo del código. Ayuda a mantener un código limpio y coherente.

3. Prettier: se encarga de formatear el código automáticamente según reglas definidas. Esto evita discusiones sobre estilo entre miembros del equipo y asegura que todo el código luzca uniforme, sin importar quién lo haya escrito.

Estas herramientas suelen usarse en conjunto dentro de VS Code. Por ejemplo, al guardar un archivo, Prettier puede formatearlo automáticamente, y ESLint puede señalar errores o advertencias en tiempo real, lo que agiliza el desarrollo y reduce fallos.

Conclusión y reflexión final

Como equipo, aprendimos que el navegador no solo sirve para visualizar páginas web, sino que también actúa como un entorno completo donde se ejecuta JavaScript. Muchas de las interacciones que vemos, como clics, animaciones o cambios dinámicos, son posibles gracias a cómo el navegador interpreta y ejecuta el código.

Además, herramientas como la consola, el inspector de elementos y el panel de red son fundamentales para depurar, probar y entender cómo funciona una página desde adentro.

Por fuera del navegador, descubrimos que editores como VS Code, junto con extensiones como ESLint y Prettier, son aliados calve para escribir código más claro, ordenado y profesional. Estas herramientas no solo facilitan el trabajo individual, sino que también mejoran la colaboración en equipo.

En resumen, aprender a usar estas herramientas desde el principio nos ayuda a desarrollar con mayor confianza, comprender mejor el entorno web y adoptar buenas prácticas que nos acompañen a lo largo de nuestra formación como programadores.