TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO SUPERIOR DE LA

SIERRA NEGRA DE AJALPAN

RAUL CRUZ TORRES

MATERIA: ESTRUCTURA DE DATOS

PORTAFOLIO DE EVIDENCIA

3er SEMESTRE

FECHA: 10/12/2025

ÍNDICE

SEMANA 1: Introducción a JavaScript y su Integración con HTML

Durante la primera semana del curso, se trabajó con dos archivos fundamentales: `code.js` e `index.html`. El archivo JavaScript contiene la implementación de tres funciones distintas, cada una diseñada para demostrar diferentes aspectos de la programación en JavaScript y su interacción con el navegador.

1. Función saludar() - Alertas Básicas

Objetivo principal: Demostrar el funcionamiento de las alertas básicas en el navegador web.

Implementación técnica: Esta función hace uso de `alert()`, un método nativo de JavaScript integrado en el objeto `window` del navegador. Cuando se ejecuta, despliega una ventana modal emergente que interrumpe temporalmente la ejecución del código y muestra el mensaje: "Hola, Bienvenidos, este ejemplo muestra como incorporar JS en HTML". Esta función es fundamental para comprender cómo JavaScript puede interactuar directamente con la interfaz del usuario sin necesidad de modificar el DOM.

Aplicación práctica: Las alertas son útiles para notificaciones rápidas, confirmaciones simples o mensajes de depuración durante el desarrollo. Aunque en aplicaciones modernas se prefieren soluciones más sofisticadas, entender este concepto es esencial para cualquier desarrollador web.

2. Función teSaludo(nombre) - Parámetros y Personalización

Objetivo principal: Ilustrar el uso de parámetros en funciones JavaScript para crear experiencias personalizadas.

Implementación técnica: Esta función representa un nivel más avanzado de interactividad. Acepta un argumento llamado `nombre` que se pasa como parámetro al momento de invocar la función. Internamente, utiliza el mismo método `alert()`, pero incorpora concatenación de cadenas para crear un mensaje dinámico. Por ejemplo, si se ejecuta `teSaludo('Raul')`, el resultado será: "Hola Raul, Bienvenidos, este ejemplo muestra como incorporar JS en HTML".

Concepto clave: Esta función introduce el principio de reutilización de código, donde una sola función puede producir múltiples resultados diferentes dependiendo de los valores que reciba. Es un primer paso hacia la programación modular y eficiente.

3. Función fizzbuz() - Lógica Condicional y Bucles

Objetivo principal: Implementar el clásico ejercicio algorítmico FizzBuzz, ampliamente utilizado en entrevistas técnicas y educación en programación.

Implementación técnica detallada:

La función utiliza un bucle `for` que inicializa una variable `num` en 1 y la incrementa en cada iteración hasta alcanzar el valor 100. Durante cada ciclo, se evalúa una serie de condiciones mediante una estructura `if-else if-else` anidada:

- **Primera condición:** Verifica si `num` es divisible tanto por 3 como por 5 (es decir, divisible por 15). Si se cumple, imprime "FizzBuzz" en la consola del navegador.
- **Segunda condición:** Comprueba si `num` es divisible únicamente por 3. En caso afirmativo, imprime "Fizz".
- **Tercera condición:** Evalúa si `num` es divisible solo por 5, imprimiendo "Buzz" si la condición es verdadera.
- **Condición por defecto:** Si ninguna de las condiciones anteriores se cumple, simplemente imprime el valor numérico de `num`.

Ejecución automática: Al final del archivo, la línea `fizzbuz();` ejecuta la función inmediatamente cuando el script se carga. Es importante destacar que los resultados no aparecen en la página web visible, sino en la Consola de Desarrollador del navegador (accesible típicamente con F12 o clic derecho > Inspeccionar > Consola).

Valor educativo: Este ejercicio enseña conceptos fundamentales como iteración, operadores módulo (%), lógica condicional y salida de datos en la consola, herramientas esenciales para cualquier desarrollador.

Archivo index.html - Estructura y Vinculación

El documento HTML actúa como la interfaz visual y el puente de comunicación entre el usuario y el código JavaScript.

1. Estructura Fundamental del Documento

El archivo sigue las mejores prácticas de HTML5, comenzando con la declaración `<!DOCTYPE html>` que indica al navegador que debe interpretar el documento bajo los estándares modernos. La estructura incluye:

- Etiqueta `<html>` como contenedor raíz
- Sección `<head>` con metadatos, incluyendo `<title>Materia Estructuras de Datos</title>` que define el texto visible en la pestaña del navegador
- Elemento `<body>` que contiene todo el contenido visible
- Un encabezado principal `<h1>` para el título de la página
- Un párrafo `<p>` con texto descriptivo

2. Elementos Interactivos mediante Atributos onclick

El documento HTML implementa interactividad a través de dos botones, cada uno vinculado a una función JavaScript diferente mediante el atributo `onclick`:

Primer botón interactivo:

html

```
<button onclick="saludar()">Click Aqui</button>
```

Este botón, al ser presionado por el usuario, desencadena la ejecución inmediata de la función `saludar()`, provocando que aparezca la alerta básica en pantalla.

Segundo botón interactivo:

html

```
<button onclick="teSaludo('Raul')">Click Aqui</button>
```

Este segundo botón ejecuta la función `teSaludo()` pasándole el argumento 'Raul', lo que resulta en una alerta personalizada con ese nombre específico.

3. Vinculación de JavaScript - El Enlace Crucial

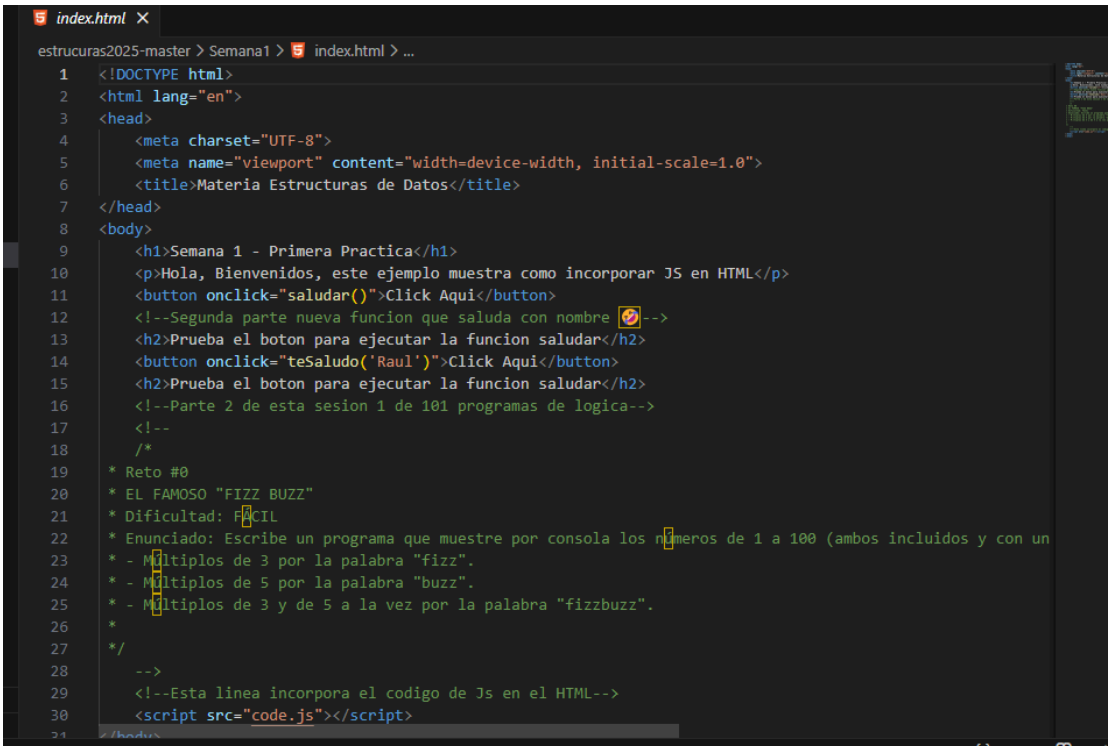
Código esencial:

html

```
<script src="code.js"></script>
```

Esta línea, estratégicamente ubicada justo antes del cierre de la etiqueta `</body>`, es fundamental para el funcionamiento de toda la aplicación. Establece la conexión entre el documento HTML y el archivo JavaScript externo. Sin esta vinculación, las funciones definidas en `code.js` no estarían disponibles para los eventos `onclick` de los botones.

¿Por qué al final del body? Colocar el script al final del documento garantiza que todo el contenido HTML se cargue antes de ejecutar JavaScript, evitando errores por intentar manipular elementos que aún no existen en el DOM.



SEMANA 2: Estructura de Sitio Web Multi-página y Organización de Recursos

La segunda semana se enfocó en crear una estructura web más compleja y organizada, compuesta por múltiples archivos HTML interconectados y recursos multimedia organizados en carpetas. Este enfoque refleja las prácticas profesionales de desarrollo web donde la modularidad y la organización son fundamentales.

Arquitectura General del Proyecto

El proyecto consta de varios archivos HTML que representan diferentes secciones de un sitio web educativo, con énfasis en sistemas operativos. Los recursos multimedia (imágenes, estilos CSS, scripts JavaScript) se organizan en carpetas dedicadas llamadas `assets/`, siguiendo el principio de separación de responsabilidades que facilita el mantenimiento y la escalabilidad del proyecto.

1. index.html - Punto de Entrada Principal

Este archivo funciona como la página de inicio o "home" del sitio web de la semana 2. Contiene la estructura base HTML5 con todas las etiquetas fundamentales: `<html>`, `<head>` y `<body>`. Además, incluye elementos de navegación que permiten al usuario acceder a las diferentes secciones del sitio mediante enlaces (`<a>` tags) o menús de navegación.

Función principal: Servir como puerta de entrada al sitio y proporcionar una visión general del contenido disponible, además de facilitar la navegación hacia las demás páginas.

2. acerca.html - Página Informativa Institucional

Este documento HTML está dedicado a la sección "Acerca de" o "About Us" del sitio web.

Contenido típico:

- Información sobre los creadores o desarrolladores del proyecto
- Propósito y objetivos del sitio web
- Misión y visión si se trata de una organización
- Historia o contexto del proyecto
- Información de contacto o formas de comunicación

Propósito educativo: Esta página enseña cómo estructurar contenido informativo corporativo o personal, un elemento esencial en cualquier sitio web profesional.

3. nosotros.html - Página del Equipo

Aunque similar en propósito a `acerca.html`, este archivo se especializa en presentar información detallada sobre el equipo o grupo responsable del proyecto.

Elementos comunes:

- Perfiles de los miembros del equipo
- Roles y responsabilidades de cada persona
- Filosofía y valores del equipo
- Logros o proyectos destacados
- Cultura organizacional

Diferencia clave: Mientras `acerca.html` se enfoca en el proyecto o la organización en general, `nosotros.html` pone el énfasis en las personas detrás del sitio.

4. Directorio assets/ - Organización de Recursos

Esta carpeta centralizada almacena todos los recursos multimedia y archivos complementarios necesarios para el funcionamiento y diseño del sitio.

Tipos de recursos típicamente incluidos:

- **Imágenes:** Fotografías, gráficos, logos, iconos
- **Hojas de estilo CSS:** Archivos que definen la apariencia visual
- **Scripts JavaScript:** Código para interactividad y funcionalidad dinámica
- **Fuentes personalizadas:** Archivos de tipografías
- **Videos o audio:** Contenido multimedia adicional

Ventaja organizacional: Mantener los recursos en una carpeta dedicada facilita la gestión de archivos, la actualización de contenido y la colaboración en equipo.

5. Directorio mi sitio/ - Sub-proyecto Independiente

Esta carpeta contiene un proyecto web completo dentro del proyecto principal, con su propio archivo `index.html` que funciona como página de inicio del sub-sitio.

Características:

- Estructura HTML independiente pero consistente con el sitio principal
- Temática o propósito específico diferente del sitio principal
- Puede tener su propia carpeta `assets/` para recursos específicos
- Permite modularización y separación de contenidos por temas

Aplicación práctica: Este enfoque es común en sitios web grandes donde diferentes secciones requieren estructuras o diseños distintos pero mantienen coherencia visual.

6. Directorio tarea/ - Módulo de Sistemas Operativos

Esta carpeta representa un proyecto completo de investigación sobre sistemas operativos, estructurado como una serie de páginas HTML interconectadas.

Archivos HTML de sistemas operativos:

android.html

- Información completa sobre el sistema operativo Android
- Historia y evolución desde sus inicios hasta versiones actuales
- Características principales: interfaz, personalización, ecosistema de aplicaciones
- Ventajas: código abierto, diversidad de dispositivos, precio accesible
- Desventajas: fragmentación de versiones, problemas de seguridad en algunos dispositivos
- Casos de uso y compatibilidad con diferentes fabricantes

ios.html

- Detalles del sistema operativo móvil de Apple
- Ecosistema iOS y su perfecta integración con hardware Apple
- Características destacadas de seguridad y privacidad
- App Store y su modelo de distribución cerrado
- Ventajas: fluidez, actualizaciones consistentes, optimización hardware-software
- Desventajas: menor personalización, ecosistema cerrado, precio elevado
- Comparativa con otros sistemas móviles

linux.html

- Sistema operativo de código abierto basado en Unix
- Distribuciones populares: Ubuntu, Fedora, Debian, Arch Linux, Linux Mint
- Ventajas para desarrollo, servidores y usuarios técnicos
- Comunidad activa y filosofía open source
- Línea de comandos y terminal poderosa
- Casos de uso: servidores web, supercomputadoras, desarrollo de software
- Desventajas: curva de aprendizaje, compatibilidad de software propietario

mac.html

- macOS y su evolución desde Mac OS X hasta las versiones actuales
- Características exclusivas: Continuity, Handoff, AirDrop, Time Machine
- Integración perfecta con el ecosistema Apple (iPhone, iPad, Apple Watch)
- Casos de uso profesionales: diseño gráfico, edición de video, producción musical
- Ventajas: estabilidad, diseño intuitivo, optimización
- Desventajas: precio elevado, hardware limitado, menor compatibilidad con juegos

windows.html

- Sistema operativo más utilizado en computadoras de escritorio
- Historia completa desde Windows 1.0 hasta Windows 11
- Compatibilidad amplia con software y videojuegos
- Uso empresarial dominante y soporte corporativo
- Ventajas: compatibilidad universal, amplia base de usuarios, soporte técnico
- Desventajas: vulnerabilidad a virus, actualizaciones intrusivas

- Ediciones: Home, Pro, Enterprise

otrossistemas.html

- Sistemas operativos alternativos o menos conocidos
- Chrome OS: sistema basado en la nube de Google
- BSD: familia de sistemas Unix (FreeBSD, OpenBSD)
- Solaris: sistema empresarial de Oracle
- Haiku OS: recreación de código abierto de BeOS
- Sistemas operativos especializados o de nicho
- Sistemas embebidos y para dispositivos IoT

index.html (índice principal)

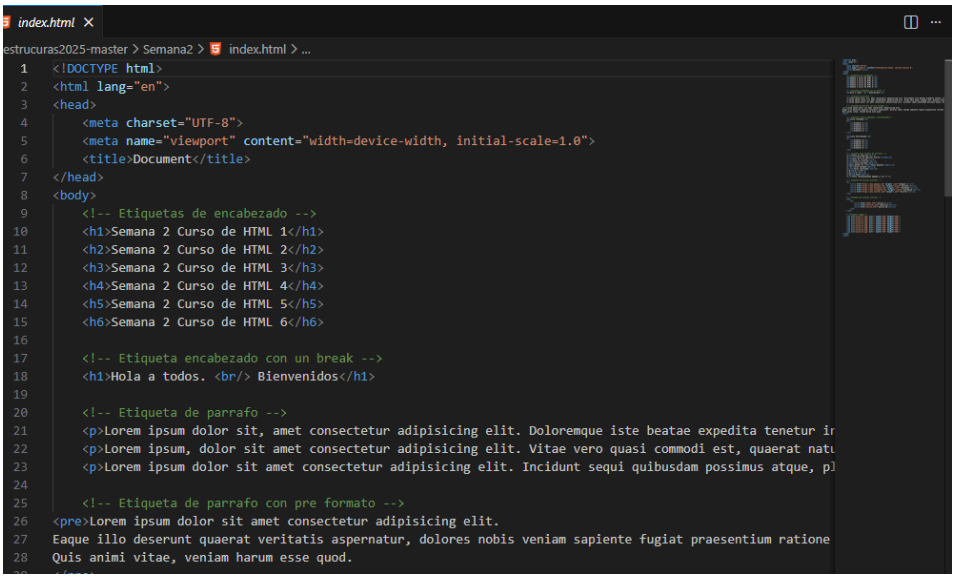
- Página de navegación o menú principal del módulo
- Enlaces organizados a cada sistema operativo específico
- Posible tabla comparativa con características principales
- Introducción general sobre qué es un sistema operativo
- Explicación de conceptos: kernel, interfaz gráfica, gestión de recursos
- Facilita la navegación entre las diferentes páginas temáticas

Subcarpeta assets/ dentro de tarea/

Esta carpeta contiene recursos específicos para el módulo de sistemas operativos:

- Logos oficiales de cada sistema operativo en alta resolución
- Capturas de pantalla de interfaces de usuario
- Iconos representativos y elementos gráficos relacionados
- Posibles diagramas comparativos de arquitectura
- Infografías sobre cuotas de mercado o estadísticas de uso

Propósito pedagógico: Este módulo enseña no solo sobre sistemas operativos desde una perspectiva técnica, sino también sobre cómo estructurar un sitio web temático con múltiples páginas relacionadas, implementando navegación efectiva, organización lógica de contenido y presentación coherente de información compleja.



SEMANA 3: Bootstrap y Diseño Responsive con Sistema Grid

La tercera semana introduce el framework CSS más popular del mundo: Bootstrap. Este módulo se enfoca en comprender los conceptos fundamentales de diseño responsive y el poderoso sistema de grid de 12 columnas que Bootstrap ofrece.

Archivo index.html - Laboratorio Práctico de Bootstrap

Este documento HTML funciona como un laboratorio interactivo donde se demuestran visualmente los diferentes componentes y sistemas de Bootstrap.

1. Contenedores Básicos - Fundamentos del Layout

Bootstrap ofrece dos tipos principales de contenedores base que determinan cómo se comporta el contenido en diferentes tamaños de pantalla:

Contenedor con ancho fijo (container):

html
```
<div class="container">
  <!-- Contenido -->
</div>
```

- Se adapta automáticamente según breakpoints predefinidos
- Tiene márgenes horizontales centrados automáticamente
- El ancho cambia escalonadamente en puntos específicos (576px, 768px, 992px, 1200px, 1400px)
- Ideal para contenido que no debe ocupar todo el ancho en pantallas grandes
- Proporciona márgenes laterales que mejoran la legibilidad

Contenedor fluido (container-fluid):

html
```
<div class="container-fluid">
  <!-- Contenido -->
</div>
```

- Ocupa el 100% del ancho disponible en cualquier tamaño de pantalla
- Sin restricciones de ancho máximo
- Perfecto para diseños que deben extenderse completamente de borde a borde
- Común en headers, footers, banners y secciones full-width
- Maximiza el uso del espacio disponible

2. Contenedores Responsivos por Breakpoint

Bootstrap 5 introduce contenedores híbridos que se comportan como fluidos hasta alcanzar cierto breakpoint, donde se convierten en contenedores fijos:

container-sm: Comportamiento fluido (100% ancho) hasta alcanzar 576px, luego se fija en un ancho máximo específico

container-md: Comportamiento fluido hasta 768px, luego ancho fijo

container-lg: Comportamiento fluido hasta 992px, luego ancho fijo

container-xl: Comportamiento fluido hasta 1200px, luego ancho fijo

container-xxl: Comportamiento fluido hasta 1400px, luego ancho fijo

Ventaja estratégica: Permite control granular sobre cuándo el contenido debe tener restricciones de ancho, mejorando la legibilidad en pantallas grandes mientras maximiza el espacio en dispositivos pequeños. Esto es especialmente útil para contenido de lectura o formularios que no deberían extenderse excesivamente en monitores anchos.

3. Sistema Grid - El Corazón del Diseño Responsive

El sistema de grid de Bootstrap es un layout basado en flexbox que divide conceptualmente el espacio horizontal en 12 columnas imaginarias. Este sistema permite crear diseños complejos y completamente responsivos con facilidad.

Estructura básica del grid:

html

```html
<div class="row">
  <div class="col">Columna 1</div>
  <div class="col">Columna 2</div>
  <div class="col">Columna 3</div>
</div>
```

Principios fundamentales del sistema:

- Las columnas deben estar siempre contenidas dentro de una fila (`.row`)
- Las filas deben estar dentro de un contenedor (`.container` o `.container-fluid`)
- Las columnas se distribuyen automáticamente de manera equitativa cuando se usa solo `.col`
- Se pueden especificar anchos específicos: `.col-6` = 50%, `.col-4` = 33.33%, `.col-3` = 25%
- El sistema suma 12 columnas: tres `.col-4` = 12 columnas totales
- Las columnas se apilan verticalmente en pantallas pequeñas por defecto

Ejemplos demostrados en el archivo:

Grid de 2 columnas equitativas:

html
```
<div class="row">
  <div class="col border">Columna 1</div>
  <div class="col border">Columna 2</div>
</div>
```

Resultado: Dos columnas de igual ancho, cada una ocupando exactamente el 50% del espacio horizontal disponible.

Grid de 3 columnas equitativas:

html
```
<div class="row">
  <div class="col border">Columna 1</div>
  <div class="col border">Columna 2</div>
  <div class="col border">Columna 3</div>
</div>
```

Resultado: Tres columnas de igual tamaño, cada una ocupando aproximadamente 33.33% del ancho total.

Grid de 9 columnas: El archivo demuestra cómo se pueden crear layouts más complejos con múltiples columnas, mostrando la flexibilidad y versatilidad del sistema. Con 9 columnas, cada una ocuparía aproximadamente 11.11% del ancho si se distribuyen equitativamente.

Visualización con bordes: La clase `border` se agrega a las columnas específicamente para propósitos didácticos, permitiendo que los estudiantes puedan ver claramente los límites de cada columna y comprender cómo se comportan y adaptan cuando se redimensiona la ventana del navegador.

4. Ejemplo Responsive con Indicadores de Color

El archivo incluye un ejemplo avanzado y visualmente impactante donde las columnas cambian de color dinámicamente según el tamaño de pantalla actual:

html
```
<div class="col responsive-bg">Contenido adaptativo</div>
```

La clase personalizada `responsive-bg` aplica diferentes colores de fondo según el breakpoint activo. Esto permite a los estudiantes visualizar instantáneamente en qué rango de tamaño de pantalla están navegando, haciendo tangible y comprensible el concepto abstracto de "responsive design".

Propósito educativo: Este indicador visual convierte un concepto teórico en una experiencia práctica e inmediata. Los estudiantes pueden redimensionar la ventana del navegador y ver cómo el color cambia en tiempo real, reforzando la comprensión de los breakpoints.

Archivo mistyle.css - Estilos Personalizados y Media Queries

Este archivo CSS complementa Bootstrap con estilos personalizados, demostrando cómo extender y personalizar el framework sin sobrescribir o romper sus funcionalidades base.

Sistema de Colores Responsive implementado con Media Queries:

El archivo implementa un sistema inteligente y educativo de colores que cambian progresivamente según el ancho de la ventana del navegador:

Extra Small (xs) - Menos de 576px:

css

```css
.responsive-bg {
  background-color: #28a745; /* Verde (success) */
  color: white;
  padding: 20px;
  text-align: center;
}
```

Dispositivos objetivo: Smartphones en modo vertical (iPhone SE, iPhone 12 mini, etc.) **Color indicador:** Verde brillante que representa el breakpoint más pequeño **Aplicación típica:** Diseño mobile-first, navegación simplificada, contenido apilado verticalmente

Small (sm) - Mayor o igual a 576px:

css

```css
@media (min-width: 576px) {
  .responsive-bg {
    background-color: #007bff; /* Azul (primary) */
  }
}
```

Dispositivos objetivo: Smartphones grandes en horizontal, phablets **Color indicador:** Azul primary de Bootstrap **Transición:** Indica que el diseño comienza a tener más espacio horizontal disponible

Medium (md) - Mayor o igual a 768px:

css

```css
@media (min-width: 768px) {
  .responsive-bg {
```

```
  background-color: #6c757d; /* Gris (secondary) */
 }
}
```

Dispositivos objetivo: Tablets en modo vertical (iPad, Android tablets) **Color indicador:** Gris secundario, neutro y profesional **Consideraciones de diseño:** Punto donde típicamente se introducen sidebars o layouts de dos columnas

Large (lg) - Mayor o igual a 992px:

```
css
@media (min-width: 992px) {
 .responsive-bg {
  background-color: #dc3545; /* Rojo (danger) */
 }
}
```

Dispositivos objetivo: Tablets en horizontal, laptops pequeñas **Color indicador:** Rojo danger para destacar este importante breakpoint **Aplicación:** Layouts más complejos con múltiples columnas, navegación completa

Extra Large (xl) - Mayor o igual a 1200px:

```
css
@media (min-width: 1200px) {
 .responsive-bg {
  background-color: #ffc107; /* Naranja/Amarillo (warning) */
 }
}
```

Dispositivos objetivo: Laptops estándar, desktops con monitores de 1080p **Color indicador:** Naranja/amarillo warning para advertir del cambio **Espacio disponible:** Suficiente para diseños complejos con sidebars, múltiples columnas y contenido rico

Extra Extra Large (xxl) - Mayor o igual a 1400px:

```
css
@media (min-width: 1400px) {
 .responsive-bg {
  background-color: #17a2b8; /* Cian (info) */
 }
}
```

Dispositivos objetivo: Monitores grandes, pantallas 4K, configuraciones multi-monitor **Color indicador:** Cian info para el breakpoint más amplio **Consideraciones:** Máximo

aprovechamiento del espacio, posibilidad de mostrar contenido adicional sin sacrificar la legibilidad

Concepto Fundamental: Media Queries

Las media queries son una característica esencial de CSS3 que permite aplicar estilos condicionales basados en características del dispositivo o viewport:

```
css
@media (condición) {
  /* Estilos que se aplican solo cuando la condición es verdadera */
}
```

Tipos de condiciones comunes:

- `min-width`: Aplica estilos cuando el ancho es mayor o igual al valor especificado
- `max-width`: Aplica estilos cuando el ancho es menor o igual al valor especificado
- `orientation`: Detecta si el dispositivo está en modo portrait o landscape
- `aspect-ratio`: Evalúa la relación de aspecto de la pantalla
- `resolution`: Detecta la densidad de píxeles (útil para pantallas Retina)

Ventajas estratégicas:

- Permiten crear diseños verdaderamente responsive y adaptativos
- No dependen de JavaScript, funcionan puramente con CSS
- Son el estándar de la industria para diseño adaptativo
- Mejoran significativamente la experiencia de usuario en todos los dispositivos
- Optimizan el rendimiento al cargar solo los estilos necesarios

Enfoque Mobile-First: Bootstrap y las prácticas modernas priorizan el diseño mobile-first, donde se diseña primero para dispositivos pequeños y luego se agregan características para pantallas más grandes usando `min-width` en media queries. Este enfoque asegura que la experiencia móvil sea óptima y que los sitios carguen rápido incluso en conexiones lentas.

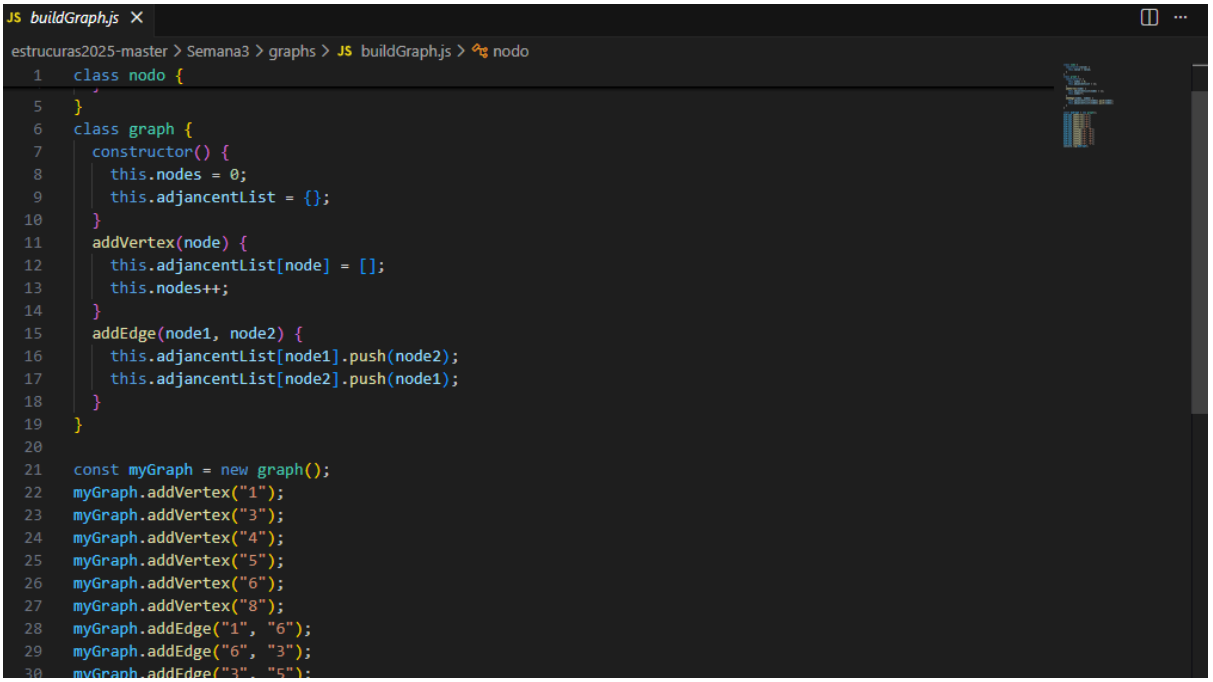
Propósito Educativo Integral del Módulo

Este proyecto de la semana 3 sirve como un tutorial práctico y completo sobre múltiples conceptos esenciales:

1. **Diseño Mobile-First:** Bootstrap prioriza dispositivos móviles y escala progresivamente hacia pantallas más grandes, reflejando el paradigma actual del desarrollo web donde la mayoría del tráfico proviene de dispositivos móviles.
2. **Sistema de Grid Flexible:** Comprender profundamente cómo dividir el espacio horizontal de manera responsive es fundamental para crear layouts profesionales y adaptativos.

3. **Breakpoints Estándar de la Industria:** Conocer los puntos de quiebre comunes (576px, 768px, 992px, 1200px, 1400px) permite diseñar para los dispositivos más utilizados actualmente.
4. **Media Queries en Práctica:** Aprender a aplicar estilos condicionales según el dispositivo es una habilidad esencial para cualquier desarrollador frontend moderno.
5. **Extensión de Frameworks:** Demostrar cómo personalizar y extender Bootstrap sin romper sus funcionalidades base enseña buenas prácticas de desarrollo.
6. **Experiencia de Usuario Adaptativa:** Entender que diferentes dispositivos requieren diferentes enfoques de diseño para maximizar la usabilidad.

Aplicación práctica en el mundo real: Los estudiantes aprenden a crear layouts profesionales que se adaptan automáticamente a cualquier tamaño de pantalla, desde el smartphone más pequeño hasta monitores 4K de escritorio. Esta habilidad es absolutamente esencial en el desarrollo web moderno, donde los usuarios acceden a sitios web desde una diversidad de dispositivos: smartphones, tablets, laptops, desktops, smart TVs e incluso wearables.



SEMANA 4: Manipulación Avanzada del DOM con JavaScript

La cuarta semana profundiza significativamente en la manipulación del Document Object Model (DOM), enseñando técnicas esenciales y avanzadas para crear páginas web dinámicas, interactivas y responsivas a las acciones del usuario. El módulo se estructura en dos ejercicios complementarios: uno principal y otro en la subcarpeta `tarea`.

Ejercicio Principal: index.html y code.js

Archivo index.html - Estructura del Documento Interactivo

Este documento HTML establece la base estructural para un sitio web interactivo con múltiples características y puntos de interacción:

Elementos estructurales fundamentales:

- **Declaración DOCTYPE HTML5:** Asegura que el navegador interprete el documento bajo los estándares web modernos más recientes.
- **Sección `<head>` comprehensiva con metadatos esenciales:**
  - Charset UTF-8 para soporte completo de caracteres especiales, acentos y símbolos internacionales
  - Viewport meta tag configurado para diseño responsive (`width=device-width, initial-scale=1.0`)
  - Título descriptivo y relevante para SEO
  - Enlaces a hojas de estilo CSS externas para separación de responsabilidades
  - Posibles meta tags para descripción, keywords y Open Graph para redes sociales
- **Cuerpo `<body>` estructurado con contenido semántico:**
  - Encabezados jerárquicos (`<h1>`, `<h2>`, `<h3>`) para organización del contenido
  - Párrafos descriptivos con información contextual
  - Secciones claramente delimitadas usando `<div>` o etiquetas semánticas HTML5
  - Áreas de contenido dinámico identificadas con IDs únicos

Componentes interactivos implementados:

- **Botones con event handlers:** Elementos `<button>` configurados para responder a eventos de usuario (clicks, hover, etc.)
- **Áreas de contenido dinámico:** Contenedores (`<div>`) con IDs específicos donde JavaScript insertará o modificará contenido
- **Formularios para captura de datos:** Inputs de diferentes tipos (text, email, number) para interacción con el usuario
- **Elementos que serán manipulados dinámicamente:** Listas, tablas, tarjetas o cualquier estructura que JavaScript modificará en tiempo de ejecución

Vinculación estratégica con JavaScript:

html

```
<script src="code.js"></script>
```

Esta línea crucial, ubicada estratégicamente justo antes del cierre de la etiqueta `</body>`, garantiza que todo el DOM esté completamente cargado y parseado antes de que cualquier script intente manipularlo, evitando errores de referencia a elementos no existentes.

Archivo code.js - Lógica de Interactividad y Manipulación del DOM

Este script de JavaScript implementa múltiples funciones que transforman una página estática en una aplicación web dinámica, interactiva y responsiva.

Técnicas fundamentales de manipulación del DOM utilizadas:

1. Selección de elementos:

javascript
```
document.getElementById('miElemento')
document.querySelector('.miClase')
document.querySelectorAll('p')
```

2. Modificación de contenido:

javascript
```
elemento.innerHTML = '<strong>Nuevo contenido HTML</strong>';
elemento.textContent = 'Nuevo texto plano';
elemento.innerText = 'Texto visible';
```

3. Manipulación de atributos:

javascript
```
elemento.setAttribute('src', 'nueva-imagen.jpg');
elemento.classList.add('activo');
elemento.classList.remove('oculto');
elemento.classList.toggle('visible');
```

4. Manejo de eventos:

javascript
```
boton.addEventListener('click', function() {
  // Código que se ejecuta al hacer clic
});

input.addEventListener('input', function(e) {
  // Código que responde a cambios en tiempo real
});
```

5. Creación y eliminación de elementos:

javascript
```
const nuevoDiv = document.createElement('div');
nuevoDiv.textContent = 'Contenido nuevo';
contenedor.appendChild(nuevoDiv);
```

```
elemento.remove(); // Eliminar elemento
```

Propósito educativo: Enseñar cómo JavaScript puede modificar la estructura, estilo y contenido de una página web después de que se ha cargado, creando experiencias dinámicas sin necesidad de recargar la página.

Subcarpeta tarea/: index2.html y code2.js

Esta sección contiene un ejercicio adicional que refuerza y expande los conceptos del módulo principal.

Archivo index2.html - Documento de Práctica

Este HTML presenta una estructura similar al archivo principal pero con elementos y desafíos diferentes:

Características distintivas:

- Formularios más complejos con múltiples campos
- Tablas o listas que deben llenarse dinámicamente
- Secciones que aparecen o desaparecen según la interacción del usuario
- Validación de datos en tiempo real
- Posibles animaciones o transiciones CSS activadas por JavaScript

Propósito: Proporcionar un entorno de práctica para que los estudiantes apliquen conceptos aprendidos en escenarios nuevos.

Archivo code2.js - Implementación de Funcionalidad

Este script JavaScript complementa `index2.html` con lógica específica:

Funcionalidades típicas implementadas:

1. Validación de formularios:

javascript

```
function validarFormulario() {
  const email = document.getElementById('email').value;
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  if (!emailRegex.test(email)) {
    mostrarError('Email inválido');
    return false;
  }
  return true;
}
```

2. Actualización dinámica de contenido:

javascript

```javascript
function actualizarContenido() {
  const datos = obtenerDatos();
  const contenedor = document.getElementById('resultados');

  contenedor.innerHTML = datos.map(item =>
    `<div class="item">${item.nombre}</div>`
  ).join('');
}
```

3. Interacción con múltiples elementos:

javascript

```javascript
const botones = document.querySelectorAll('.boton');
botones.forEach(boton => {
  boton.addEventListener('click', function() {
    this.classList.toggle('activo');
  });
});
```

4. Almacenamiento temporal de datos:

javascript

```javascript
let datosTemporales = [];

function agregarDato(dato) {
  datosTemporales.push(dato);
  actualizarVista();
}
```

Conceptos Clave de la Semana 4

Document Object Model (DOM): El DOM es una representación en árbol de todos los elementos HTML de una página. JavaScript puede acceder y modificar cualquier parte de este árbol.

Event-Driven Programming: Los programas responden a eventos (clicks, teclas presionadas, movimientos del mouse) en lugar de seguir un flujo lineal predefinido.

Separación de responsabilidades:

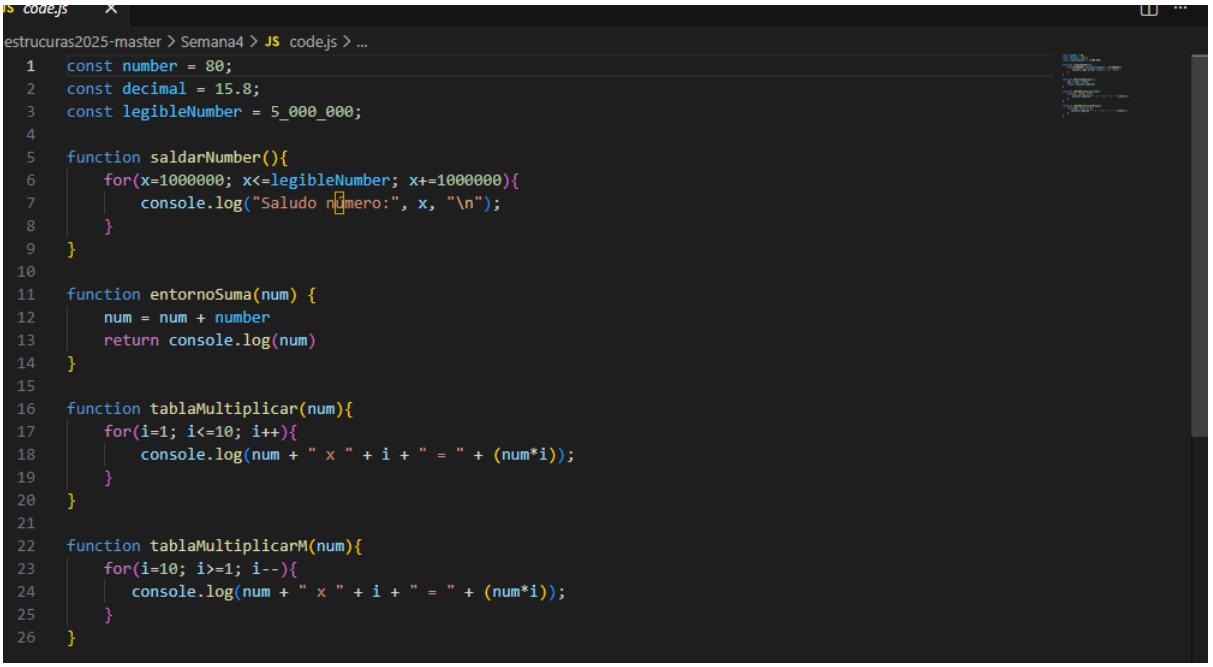
- HTML: Estructura y contenido semántico
- CSS: Presentación y estilo visual
- JavaScript: Comportamiento e interactividad

Mejores prácticas enseñadas:

- Usar IDs únicos para elementos que serán manipulados
- Validar datos antes de procesarlos
- Manejar errores graciosamente
- Mantener el código organizado en funciones reutilizables
- Comentar el código para facilitar su mantenimiento

Resumen del Módulo

La semana 4 establece las bases para la programación web moderna. Los estudiantes aprenden que una página web no es un documento estático sino una aplicación viva que puede responder a las acciones del usuario, actualizar contenido sin recargar, validar información y proporcionar una experiencia interactiva rica. Estos conceptos son fundamentales para cualquier desarrollador web y preparan el camino para frameworks y bibliotecas más avanzadas como React, Vue o Angular.



SEMANA 5: Fundamentos de Manipulación del DOM

Durante la quinta semana del curso, se trabaja con dos archivos esenciales que demuestran los conceptos básicos de manipulación del Document Object Model (DOM): **code.js** e **index.html**. Estos archivos constituyen la base para entender cómo JavaScript puede interactuar dinámicamente con los elementos HTML de una página web.

1. Análisis del Archivo: code.js

El archivo JavaScript contiene la lógica de programación dividida en dos funciones fundamentales que operan sobre el documento HTML:

Función llenarLista():

Esta función representa uno de los casos de uso más comunes en el desarrollo web: la manipulación dinámica del contenido HTML mediante JavaScript. Su propósito principal es poblar una lista de ideas metodológicas dentro del documento HTML utilizando el método `document.getElementById()` para localizar elementos específicos mediante sus identificadores únicos y posteriormente asignarles contenido a través de la propiedad `innerHTML`.

Las ideas que se insertan en la lista corresponden a una metodología de resolución de problemas en programación:

1. Comprender completamente el problema antes de comenzar a codificar
2. Descomponer el problema en componentes más pequeños y manejables
3. Abordar la resolución de cada componente de manera individual y aislada
4. Integrar las soluciones parciales para construir la solución completa del problema
5. Realizar pruebas exhaustivas y ajustar la solución según los resultados obtenidos
6. Un mensaje adicional de ejemplo: "aaaaaaa!"

Esta función demuestra cómo JavaScript puede modificar el contenido de una página web después de que esta ha sido cargada por el navegador, permitiendo experiencias de usuario más dinámicas e interactivas.

Función saludar():

Esta función implementa un flujo de interacción básico con el usuario. Captura el valor ingresado en un campo de entrada de texto (elemento `<input>`) identificado con el ID "nombre", procesa esta información y genera un saludo personalizado que se muestra en un elemento HTML con el ID "miSaludo".

Adicionalmente, la función mantiene un registro de los nombres ingresados almacenándolos en un arreglo global llamado `alumnos`, lo que permite acumular datos durante la sesión del usuario y potencialmente utilizarlos para operaciones posteriores.

2. Análisis del Archivo: index.html

El documento HTML proporciona la estructura visual y semántica sobre la cual opera el código JavaScript. Este archivo sigue las mejores prácticas de desarrollo web moderno:

Encabezados y Metadatos:

El documento comienza con la declaración DOCTYPE de HTML5 y establece el idioma del contenido como inglés mediante el atributo `lang`. La sección `<head>` incluye metadatos esenciales:

- Especificación de la codificación de caracteres UTF-8 para soportar caracteres especiales
- Configuración del viewport para garantizar un diseño responsivo en dispositivos móviles

- Título del documento que aparece en la pestaña del navegador

Estructura del Cuerpo del Documento:

El cuerpo del HTML está organizado de manera lógica y jerárquica:

- Un encabezado principal (`<h1>`) que introduce el tema del ejercicio
- Un subtítulo (`<h2>`) que describe específicamente la práctica de completar dinámicamente una lista de ideas
- Un párrafo (`<p>`) con texto de relleno Lorem Ipsum para demostrar la estructura de contenido
- Una lista desordenada (`<ul>`) que contiene múltiples elementos de lista (`<li>`), cada uno con IDs únicos que servirán como puntos de anclaje para la función `llenarLista()`
- Un botón interactivo que, al ser presionado, invoca la función `llenarLista()` para poblar dinámicamente la lista con las ideas predefinidas
- Un encabezado adicional de nivel 3 (`<h3>`) destinado a mostrar mensajes adicionales o información complementaria

Funcionamiento Integral del Sistema

El flujo de trabajo del sistema es directo pero instructivo: cuando el usuario carga la página HTML en su navegador, visualiza una estructura con espacios vacíos esperando ser completados. Al hacer clic en el botón etiquetado como "Da Clic para llenar lista", se ejecuta la función `llenarLista()` que recorre cada elemento de la lista y lo llena con las ideas correspondientes sobre metodología de resolución de problemas.

Paralelamente, la función `saludar()` ofrece una experiencia interactiva personalizada donde el usuario puede ingresar su nombre y recibir un saludo customizado, mientras que el sistema mantiene un registro interno de todos los nombres ingresados.

En síntesis, estos dos archivos trabajando en conjunto ejemplifican los principios fundamentales de la manipulación del DOM mediante JavaScript, específicamente utilizando la propiedad `innerHTML` y el método `getElementById()` para actualizar el contenido de una página web de forma dinámica y reactiva a las acciones del usuario.

SEMANA 6: Tipos de Datos y Estructuras Básicas

La sexta semana del curso introduce conceptos fundamentales sobre tipos de datos en JavaScript y su visualización en el navegador. La carpeta contiene dos ejemplos complementarios: uno en el directorio raíz y otro en la subcarpeta **ex**, cada uno enfocando diferentes aspectos de la manipulación de datos.

1. Ejemplo Principal: index.html y code.js

index.html - Página Introductoria

El documento HTML establece una página web básica pero funcional con el título descriptivo "Semana 6 - Trabajando con estructuras de datos". La estructura incluye:

- Un botón de acción que ejecuta la función `mostrarVariables()` al ser presionado
- Un contenedor `<div>` con el ID "resultado" que actúa como área de salida para mostrar los resultados generados dinámicamente
- Referencias a los archivos externos de estilos (CSS) y lógica (JavaScript)

code.js - Demostración de Tipos de Datos Primitivos

Este archivo implementa la función `mostrarVariables()` que sirve como demostración educativa de los tipos de datos fundamentales en JavaScript:

Proceso interno de la función:

1. **Declaración de Variables:** Se crean cuatro variables que representan diferentes tipos de datos:
    - `nombre`: Variable de tipo String (cadena de texto)
    - `edad`: Variable de tipo Number (número entero)
    - `esEstudiante`: Variable de tipo Boolean (valor verdadero/falso)
    - `estatura`: Variable de tipo Number con valor decimal (punto flotante)
2. **Construcción de HTML Dinámico:** La función utiliza template literals (cadenas de texto delimitadas por backticks `) para construir una estructura HTML compleja de manera más legible y mantenible
3. **Detección de Tipos:** Emplea el operador `typeof` de JavaScript para identificar y mostrar el tipo de dato de cada variable
4. **Renderización:** Inserta el HTML generado dinámicamente en el elemento del DOM mediante la propiedad `innerHTML`

Objetivo pedagógico: Esta implementación enseña a los estudiantes los tipos de datos primitivos fundamentales en JavaScript (String, Number, Boolean) y cómo visualizar esta información de manera clara en la interfaz de usuario.

2. Subcarpeta ex/: index2.html y code2.js

index2.html - Estructura de Tabla HTML

Este archivo define una tabla HTML semántica y bien estructurada:

- Encabezados de columna claros: ID, Nombre y Edad
- Tres filas de datos preparadas con celdas vacías
- Cada celda tiene un ID único siguiendo un patrón consistente (ID1, Nombre1, Edad1, etc.)
- Un botón que ejecuta la función `mostrarDatos()` para llenar dinámicamente la tabla con información

code2.js - Población Dinámica de Tabla

La función `mostrarDatos()` implementa un patrón de llenado manual de tabla:

Funcionamiento:

1. **Declaración de Variables:** Define nueve variables que contienen información estructurada de tres personas, organizadas en grupos de tres (ID, nombre y edad para cada registro)
2. **Manipulación del DOM:** Utiliza el método `getElementById()` para localizar cada celda específica de la tabla
3. **Asignación de Contenido:** Emplea la propiedad `innerText` (en lugar de `innerHTML`) para insertar texto plano en cada celda, evitando interpretación de código HTML

4. **Estructura Manual:** Los datos se organizan manualmente para tres registros completos, demostrando cómo poblar sistemáticamente una tabla HTML

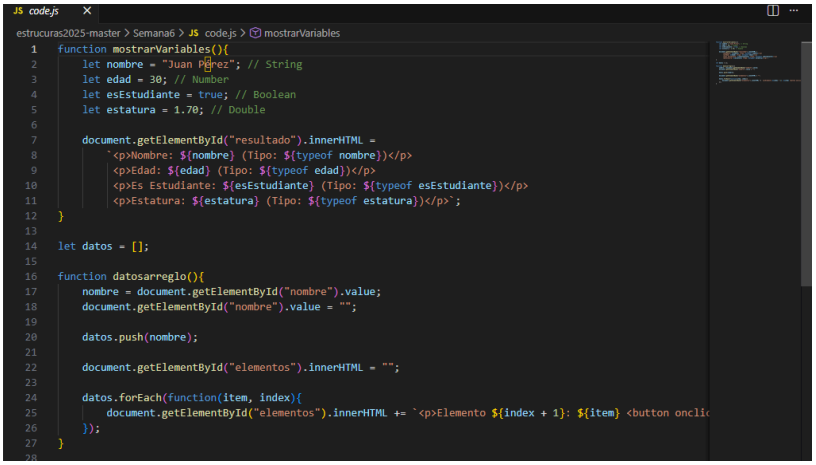
Propósito didáctico: Este ejemplo enseña cómo manipular estructuras de datos tabulares, diferencia entre `innerText` e `innerHTML`, y establece las bases para trabajar posteriormente con estructuras de datos más complejas como arrays y objetos.

Resumen Conceptual de la Semana 6

Esta semana aborda dos pilares fundamentales del desarrollo web con JavaScript:

- **Ejemplo 1:** Se centra en comprender los tipos de datos primitivos (String, Number, Boolean) y cómo JavaScript maneja internamente diferentes categorías de información
- **Ejemplo 2:** Introduce el concepto de estructuras de datos más elaboradas, mostrando cómo organizar y presentar información relacionada en formato tabular

Ambos ejemplos sientan las bases para trabajar con estructuras de datos más complejas en semanas posteriores, como arrays de objetos, manipulación de colecciones y operaciones CRUD.



SEMANA 7: Estructuras de Datos Avanzadas - Arrays y Objetos

La séptima semana profundiza en el manejo de estructuras de datos complejas en JavaScript, introduciendo conceptos de arrays (arreglos) y objetos. Los archivos de esta semana demuestran cómo almacenar, manipular y visualizar colecciones de datos.

Análisis de index.html (Estructura HTML Educativa)

El documento HTML presenta una interfaz web educativa organizada en tres secciones temáticas bien diferenciadas:

1. Sección: Declarando Variables

Esta primera área repasa los fundamentos vistos en la semana anterior, mostrando ejemplos prácticos de diferentes tipos de datos primitivos:

- Strings (cadenas de texto)
- Numbers (números enteros y decimales)
- Booleans (valores lógicos verdadero/falso)
- Undefined y Null (valores especiales de JavaScript)

2. Sección: Estructuras de Arreglo

Esta sección introduce el concepto de arrays, permitiendo al usuario experimentar con la adición dinámica de elementos:

- Campo de entrada para capturar nombres
- Botón para agregar el nombre al arreglo
- Área de visualización que muestra todos los elementos almacenados
- Controles de edición y eliminación para cada elemento

3. Sección: Estructuras de Datos Múltiples (Objetos)

La tercera sección implementa un formulario completo para capturar información estructurada de una persona:

- Campo de texto para el nombre
- Campo numérico para la edad
- Campo de teléfono
- Campo decimal para la estatura
- Checkbox para el estado civil (soltero/casado)
- Botón para procesar y almacenar la información

Análisis de code.js (Lógica de JavaScript)

El archivo JavaScript contiene tres funciones principales que implementan la lógica funcional de cada sección:

1. Función mostrarVariables()

Esta función realiza una demostración práctica de tipos de datos:

Operaciones realizadas:

- Declara múltiples variables de diferentes tipos
- Utiliza el operador `typeof` para identificar dinámicamente el tipo de cada variable

- Construye una salida HTML formateada que muestra cada variable junto con su tipo
- Renderiza la información en un elemento específico del DOM

Propósito: Reforzar la comprensión de los tipos de datos primitivos y cómo JavaScript los clasifica internamente.

2. Función datosarreglo()

Esta función implementa operaciones básicas con arrays:

Funcionalidades implementadas:

1. **Captura de Datos:** Lee el valor del campo de entrada de texto
2. **Validación:** Verifica que el campo no esté vacío antes de proceder
3. **Almacenamiento:** Agrega el nombre al arreglo usando el método `push()`
4. **Visualización Dinámica:** Recorre el arreglo con `forEach()` y genera HTML para cada elemento
5. **Interactividad:** Crea botones de "Eliminar" y "Editar" para cada elemento (aunque la funcionalidad completa puede requerir implementación adicional)
6. **Actualización del DOM:** Muestra la lista actualizada en tiempo real

Conceptos enseñados:

- Manipulación de arrays con métodos nativos
- Iteración sobre colecciones con `forEach()`
- Generación dinámica de HTML basada en datos
- Manejo de eventos para elementos creados dinámicamente

3. Función datoStruct()

Esta función demuestra el trabajo con objetos JavaScript:

Proceso de ejecución:

1. **Creación del Objeto:** Define un objeto llamado `miEstructura` con múltiples propiedades:

javascript

```
{
  nombre: "",
  edad: 0,
  telefono: "",
  estatura: 0.0,
  soltero: false
```

```
}
```

2. **Captura de Valores del Formulario:** Lee los valores ingresados por el usuario en cada campo del formulario
3. **Conversión de Tipos:** Aplica funciones de conversión para asegurar tipos de datos correctos:
    - `parseInt()` para convertir la edad de string a número entero
    - `parseFloat()` para convertir la estatura a número decimal
    - Manejo directo del checkbox para el valor booleano
4. **Asignación de Propiedades:** Almacena cada valor capturado en su propiedad correspondiente del objeto
5. **Visualización o Procesamiento:** Muestra el objeto completo o lo prepara para operaciones posteriores

Conceptos fundamentales:

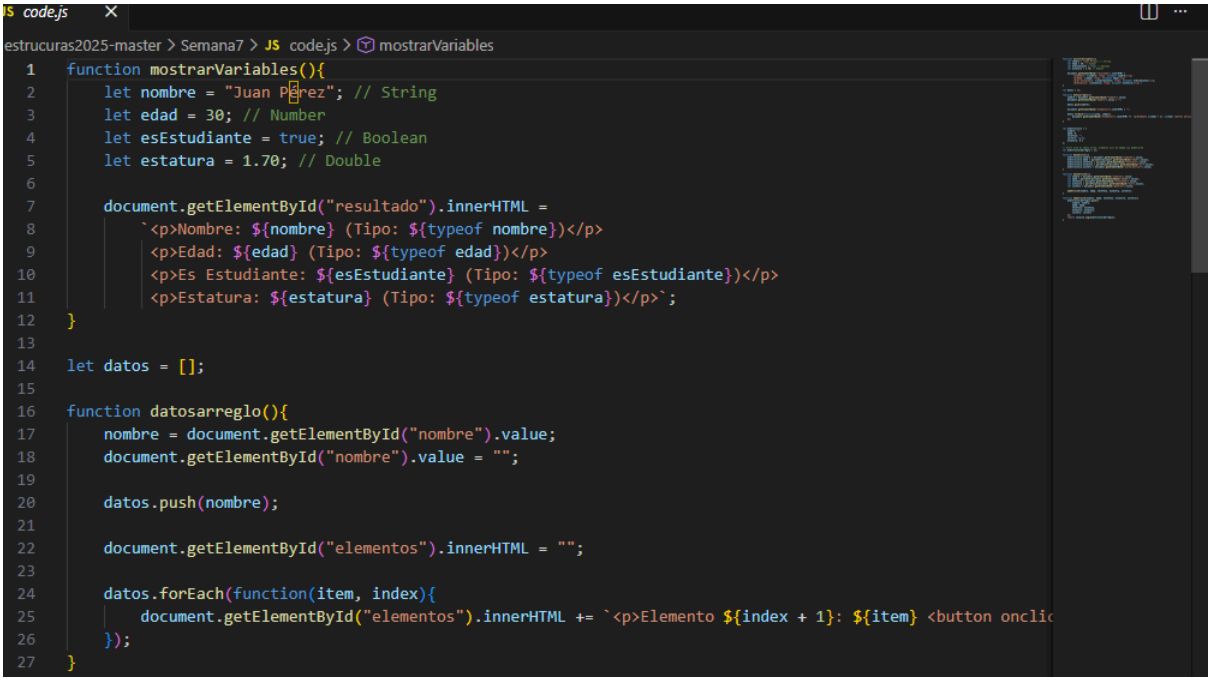
- Estructura de objetos en JavaScript (pares clave-valor)
- Conversión de tipos de datos con funciones parse
- Captura de datos de formularios HTML
- Diferencia entre datos primitivos y estructuras complejas

Objetivo Pedagógico de la Semana 7

Esta semana establece los fundamentos para trabajar con estructuras de datos complejas:

1. **Arrays:** Colecciones ordenadas de elementos que permiten almacenar múltiples valores en una sola variable
2. **Objetos:** Estructuras que permiten agrupar datos relacionados bajo un esquema con propiedades nombradas
3. **Manipulación de Datos:** Técnicas para agregar, visualizar y potencialmente editar o eliminar información

Estos conceptos son esenciales para el desarrollo de aplicaciones web dinámicas y preparan al estudiante para trabajar con datos más complejos provenientes de APIs o bases de datos.



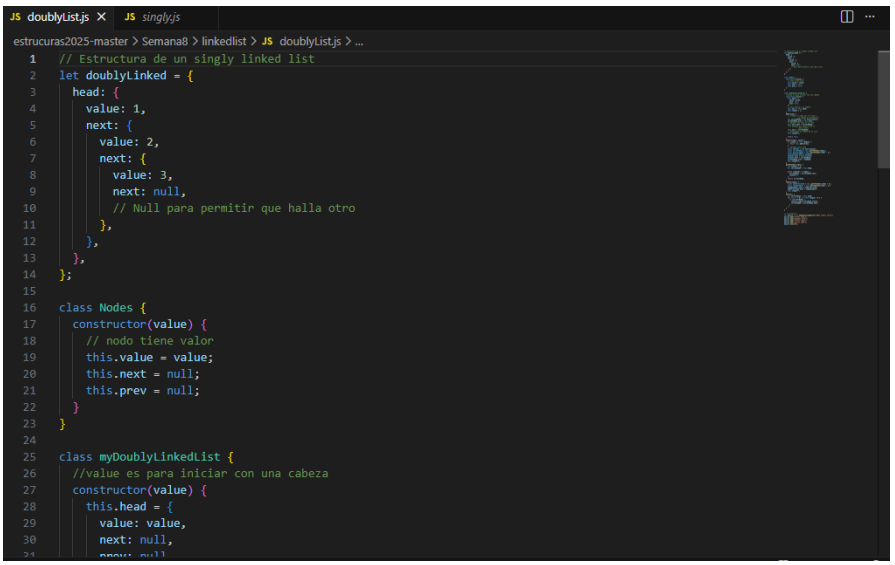
SEMANA 8

El objetivo principal del archivo addNode.js es: Implementar la operación de agregar un nodo a una lista enlazada. Comprender el uso de referencias (next) entre nodos. Aplicar programación estructurada mediante funciones. Reforzar el concepto de estructuras de datos dinámicas.

Lenguaje y Tecnologías Utilizadas JavaScript Lenguaje utilizado para la implementación de la lista enlazada y la manipulación de nodos.

El código se ejecuta en un entorno web o de consola, priorizando la lógica algorítmica sobre la interfaz gráfica. Operación principal La función addNode permite añadir dinámicamente elementos, evitando el uso de arreglos de tamaño fijo. 6.2

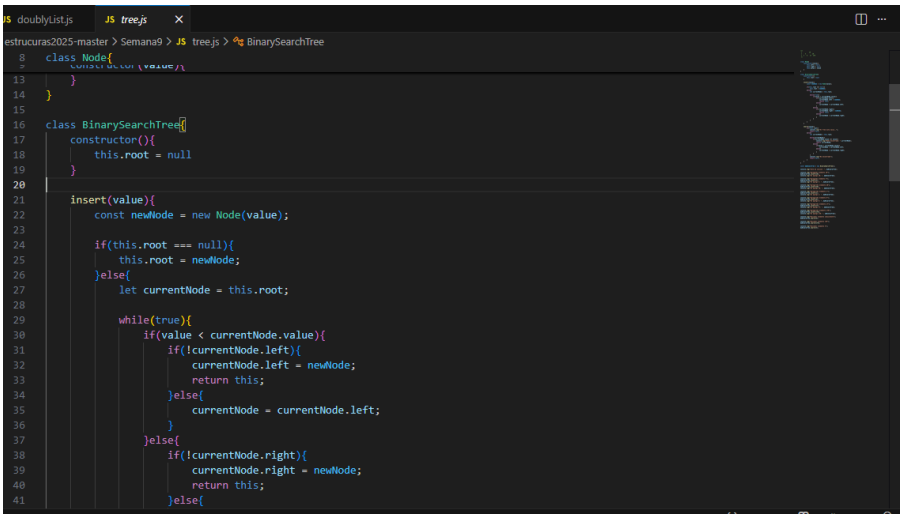
Uso de referencias Cada nodo mantiene una referencia al siguiente (next), lo que permite recorrer la lista sin necesidad de índices. 6.3 Complejidad Tiempo: O(n), si la inserción se realiza al final de la lista. Espacio: O(1), ya que solo se agrega un nuevo nodo.



Semana 9

Objetivo length:

almacena la cantidad de elementos. data: es un objeto donde se guardan los valores por índice También hay varias pruebas comentadas donde se muestran ejemplos de cómo: agregar con push(), consultar con get(), eliminar con pop(), eliminar en una posición con delete(),agregar al inicio con unshift(). Las líneas con unshift() se ejecutan y despliegan resultados.



SEMANA 10: Consumo de APIs REST - Proyecto Rick and Morty

La décima semana marca un hito importante en el curso al introducir el concepto de consumo de APIs externas. El proyecto implementa una aplicación web que interactúa con la API pública de Rick and Morty para obtener y mostrar información de los personajes de la popular serie animada.

Análisis Detallado de index.html

Estructura y Propósito

El archivo HTML constituye la estructura principal de la aplicación, implementando un diseño moderno y profesional:

Componentes del Header:

- Logo minimalista "RM" que identifica la marca
- Título principal de la aplicación
- Barra de navegación con botones para diferentes secciones o funcionalidades
- Diseño sticky (pegado) que permanece visible durante el scroll

Sección Hero:

- Área destacada con descripción del propósito de la aplicación
- Texto introductorio que explica la demo y sus capacidades
- Diseño visualmente atractivo para captar la atención del usuario

Contenedor Principal:

- Un `<div>` con el ID "grid" que actúa como contenedor dinámico

- Este elemento se llena mediante JavaScript con las tarjetas de personajes
- Implementa un sistema de grid responsivo que se adapta a diferentes tamaños de pantalla

Footer:

- Sección de pie de página con créditos a la API de Rick and Morty
- Enlaces relevantes y información adicional

Enlaces a Recursos:

- Referencia al archivo CSS para estilos visuales
- Referencia al archivo JavaScript para la lógica de la aplicación

Análisis Completo de style.css

Filosofía de Diseño

El archivo de estilos implementa un tema oscuro moderno inspirado en el sitio oficial de la API de Rick and Morty:

Sistema de Variables CSS:

css

```
:root {
  --primary-color: #2ceaa6;    /* Verde neón característico */
  --secondary-color: #ffd166;   /* Amarillo vibrante */
  --dark-bg: #1a1a1a;          /* Fondo oscuro principal */
  --card-bg: #2d2d2d;          /* Fondo de tarjetas */
}
```

Características Visuales Principales:

1. **Tema Oscuro Profesional:**
   - Paleta de colores cuidadosamente seleccionada
   - Alto contraste para mejorar la legibilidad
   - Verde neón (`#2ceaa6`) y amarillo (`#ffd166`) como colores de acento
2. **Efectos Glassmorphism:**
   - Bordes semitransparentes con `backdrop-filter: blur()`
   - Sombras suaves y difuminadas
   - Efecto de profundidad y modernidad visual
3. **Tipografía:**
   - Fuente principal: Inter de Google Fonts

    - Carga optimizada para mejorar el rendimiento
    - Jerarquía tipográfica clara y legible
4. **Sistema de Layout:**
    - Uso extensivo de Flexbox para alineación y distribución
    - Grid responsivo que se adapta a diferentes viewports
    - Header sticky con position fixed
5. **Componentes Estilizados:**
    - **Botones:** Con estados hover, active y focus bien definidos
    - **Tarjetas:** Sombras profundas, bordes redondeados, transiciones suaves
    - **Grid:** Sistema de cuadrícula fluido que reorganiza columnas según el espacio disponible
6. **Gradientes Decorativos:**
    - Fondos con gradientes sutiles para añadir profundidad visual
    - Efectos de iluminación en elementos importantes
7. **Diseño Responsivo:**
    - Media queries para tablets y móviles
    - Reorganización de layouts en pantallas pequeñas
    - Optimización de tamaños de fuente y espaciado

Análisis Exhaustivo de code.js

Estructura y Arquitectura

El archivo JavaScript implementa la lógica completa de la aplicación utilizando programación asíncrona y manipulación del DOM:

Funciones Principales:

1. getCharacters() - Función de Obtención de Datos

Esta función asíncrona realiza la petición HTTP a la API:

javascript

```javascript
async function getCharacters() {

  try {

    const response = await fetch('https://rickandmortyapi.com/api/v2/character?page=1');

    const data = await response.json();

    return data.results;

  } catch (error) {

    console.error('Error fetching characters:', error);

  }
```

```
}
```

Operaciones realizadas:

- Utiliza `fetch()` para realizar una petición GET asíncrona
- Solicita la primera página de personajes (20 personajes por página)
- Convierte la respuesta JSON a un objeto JavaScript
- Retorna el array de resultados
- Implementa manejo de errores con try-catch

Conceptos enseñados:

- Programación asíncrona con async/await
- Consumo de APIs REST
- Manejo de promesas
- Gestión de errores en operaciones asíncronas

2. createCard(person) - Función de Renderización

Esta función genera el HTML de cada tarjeta de personaje:

Datos procesados:

- `name`: Nombre del personaje
- `status`: Estado vital (Alive, Dead, Unknown)
- `species`: Especie del personaje
- `type`: Tipo específico (si aplica)
- `gender`: Género del personaje
- `origin`: Lugar de origen
- `location`: Ubicación actual
- `image`: URL de la imagen del personaje

Elementos generados:

- Imagen destacada del personaje
- Nombre como título principal
- Badges de estado con colores condicionales:
    - Verde para "Alive"
    - Rojo para "Dead"
    - Gris para "Unknown"
- Información adicional sobre especie, tipo, género
- Datos de origen y ubicación actual

Técnicas implementadas:

- Template literals para construir HTML complejo
- Interpolación de variables dinámicas
- Clases CSS condicionales basadas en el estado

- Estructura semántica y accesible

3. Flujo de Inicialización

El script se ejecuta al cargar la página:

1. **Obtención de Datos:** Llama a `getCharacters()` para recuperar los personajes
2. **Iteración:** Recorre el array de personajes recibidos
3. **Renderización:** Para cada personaje, llama a `createCard()` y agrega el HTML al contenedor grid
4. **Actualización del DOM:** Todos los cambios se reflejan inmediatamente en la interfaz

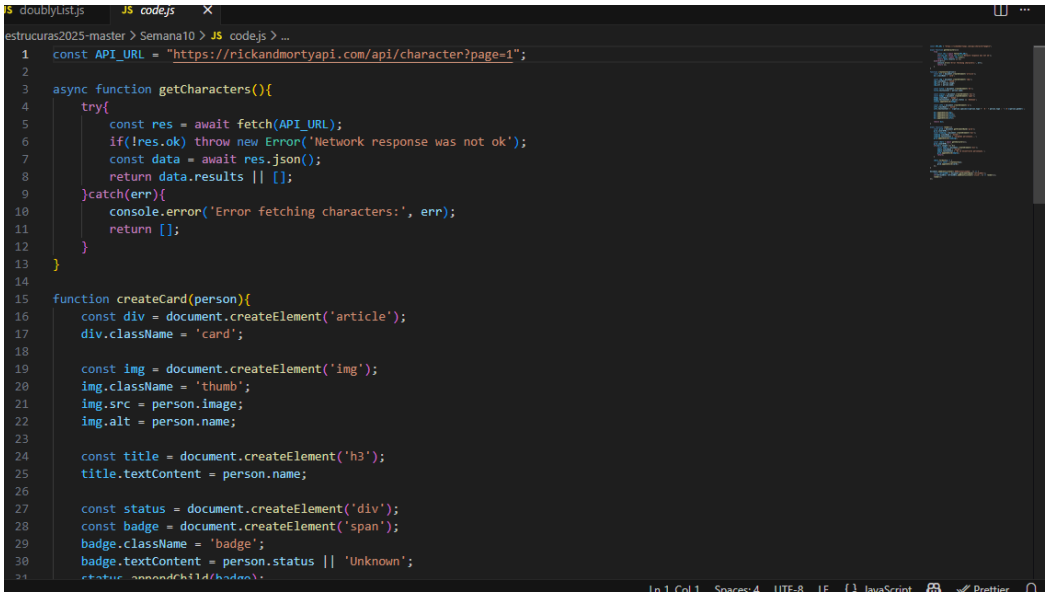
Conceptos Avanzados Introducidos

1. **APIs REST:** Comprensión de cómo las aplicaciones web se comunican con servicios externos
2. **JSON:** Formato estándar para intercambio de datos
3. **Fetch API:** Interfaz moderna de JavaScript para realizar peticiones HTTP
4. **Async/Await:** Sintaxis moderna para manejar operaciones asíncronas
5. **Manipulación Dinámica del DOM:** Creación y inserción de elementos HTML mediante JavaScript
6. **Diseño Responsivo:** CSS que se adapta a diferentes dispositivos
7. **Efectos Visuales Modernos:** Glassmorphism, gradientes, transiciones

Importancia Pedagógica

Este proyecto representa la transición de trabajar con datos estáticos a consumir información dinámica de fuentes externas, una habilidad fundamental en el desarrollo web moderno. Los estudiantes aprenden:

- Cómo estructurar peticiones a APIs
- Manejo de datos asíncronos
- Presentación visual de información externa
- Implementación de diseños modernos y atractivos



SEMANA 11: E-commerce de Sillas - Aplicación de Carrito de Compras

La undécima semana presenta un proyecto completo de comercio electrónico que implementa una tienda en línea funcional con sistema de carrito de compras y persistencia de datos. Este proyecto integra múltiples conceptos aprendidos en semanas anteriores.

Arquitectura y Componentes del Sistema

1. index.html - Estructura Semántica

Este archivo constituye el esqueleto estructural de toda la aplicación web:

- **Encabezado interactivo (Header)** que incluye:
  - Título principal destacado: "PRODUCT LIST"
  - Icono de carrito de compras con contador dinámico que muestra el número total de artículos agregados
  - Implementación de iconografía SVG proveniente de la biblioteca Flowbite para mantener escalabilidad y calidad visual
- **Secciones funcionales principales**:
  - `.listProduct`: Contenedor principal donde se renderizan dinámicamente todos los productos disponibles mediante JavaScript
  - `.cartTab`: Panel lateral deslizante que contiene el carrito de compras, inicialmente oculto y activado mediante interacción del usuario
- **Vinculaciones externas**: Conexión con la hoja de estilos (`style.css`) y el archivo de lógica JavaScript (`app.js`)

2. app.js - Motor de Lógica de Negocio

Este archivo representa el núcleo funcional de la aplicación, implementando toda la lógica necesaria para el funcionamiento del e-commerce:

Variables Globales de Estado:

- `listProducts[]`: Array que almacena todos los productos cargados desde el archivo JSON
- `carts[]`: Array que mantiene el estado actual del carrito de compras del usuario

Funciones Principales Implementadas:

`addDataToHTML()`

- Transforma los datos JSON en elementos HTML visualmente atractivos

- Genera tarjetas de producto que incluyen: imagen destacada, nombre del producto, precio en moneda local y botón de acción "Add To Cart"
- Utiliza template literals para construcción eficiente del DOM

`addToCart(product_id)`

- Gestiona la adición de productos al carrito de compras
- Implementa lógica anti-duplicación: si el producto ya existe, incrementa la cantidad en lugar de crear una entrada duplicada
- Actualiza automáticamente el contador visual del carrito

`addCartToHTML()`

- Renderiza dinámicamente el contenido completo del carrito
- Muestra para cada item: imagen miniatura, nombre del producto, controles de cantidad (+ y -), y precio total calculado
- Actualiza en tiempo real conforme el usuario modifica cantidades

`addCartToMemory()`

- Serializa el estado actual del carrito en formato JSON
- Persiste los datos en localStorage del navegador
- Garantiza que el carrito se mantenga entre sesiones de navegación

`initApp()`

- Función de inicialización que se ejecuta al cargar la página
- Realiza fetch asíncrono del archivo `products.json`
- Pobla la interfaz con todos los productos disponibles
- Recupera y restaura el carrito guardado desde localStorage si existe información previa

Gestión de Eventos:

- **Toggle del carrito**: Apertura y cierre del panel lateral mediante click en el icono
- **Adición de productos**: Event delegation para manejar clicks en múltiples botones "Add To Cart"
- **Control de cantidades**: Listeners para incrementar/decrementar cantidades directamente desde el carrito

3. products.json - Base de Datos de Productos

Archivo de datos estructurado en formato JSON que sirve como base de datos de la aplicación:

- Contiene un **array de 20 objetos** representando diferentes modelos de sillas
- Variedad de categorías: Moderna, Clásica, Oficina, Gamer, Ergonómica, etc.

Estructura de cada objeto producto:

json

```json
{
  "id": 1,
  "name": "Silla Moderna Ejecutiva",
  "price": 2500,
  "image": "1.png"
}
```

- `id`: Identificador único entero
- `name`: Nombre descriptivo del producto
- `price`: Precio expresado en MXN (Pesos Mexicanos)
- `image`: Ruta relativa hacia el archivo de imagen en la carpeta assets/

4. style.css - Diseño Visual y Responsivo

Hoja de estilos completa que define toda la apariencia visual de la aplicación:

Diseño del Header:

- Layout flexible usando Flexbox
- Icono de carrito con badge circular de notificación en color rojo
- Posicionamiento fijo para mantener visibilidad durante el scroll

Grid de Productos:

- Sistema de rejilla CSS Grid con 4 columnas en pantallas grandes
- Tarjetas de producto con diseño card moderno
- Sombras sutiles (box-shadow) para profundidad visual
- Bordes redondeados para estética contemporánea

Panel de Carrito:

- Elemento de posición fija con ancho de 400px
- Deslizamiento animado desde el borde derecho de la pantalla
- Fondo semitransparente oscuro (overlay) para enfocar atención
- Scrollbar personalizado y oculto para limpieza visual

Animaciones y Transiciones:

- Transiciones CSS suaves de 0.5 segundos para todas las interacciones
- Efectos hover en botones con cambios de color y escala
- Estados activos diferenciados visualmente

Tipografía Personalizada:

- Fuente principal: **Poppins** (Google Fonts) para contenido general
- Fuente decorativa: **Kablammo** para títulos y elementos destacados

Responsive Design:

- Media queries para adaptación a tablets (2 columnas)
- Optimización para móviles (1 columna)
- Ajuste de tamaños de fuente según viewport

5. Directorio /assets/ - Recursos Gráficos

Carpeta contenedora de todos los recursos visuales del proyecto:

- **40 archivos de imagen** en total (20 productos × 2 formatos)
- Formatos disponibles: PNG (con transparencia) y JPG (comprimidos)
- Nomenclatura numérica secuencial: `1.png`, `1.jpg`, `2.png`, `2.jpg`, ..., `20.png`, `20.jpg`
- Imágenes optimizadas para carga rápida en web

Flujo Completo de Funcionamiento

Fase 1: Inicialización

1. El navegador carga y parsea `index.html`
2. JavaScript ejecuta la función `initApp()`
3. Se realiza petición asíncrona (fetch) a `products.json`
4. Los datos se almacenan en el array `listProducts[]`

Fase 2: Renderizado Inicial

1. La función `addDataToHTML()` procesa cada producto
2. Se generan 20 tarjetas HTML dinámicamente
3. Las tarjetas se insertan en el contenedor `.listProduct`
4. El grid CSS organiza los productos automáticamente

Fase 3: Interacción del Usuario

1. **Agregar al carrito**:
    - Usuario hace clic en "Add To Cart"
    - Se ejecuta `addToCart(product_id)`
    - El producto se agrega o su cantidad se incrementa
    - El contador del header se actualiza
2. **Visualizar carrito**:
    - Usuario hace clic en el icono del carrito
    - El panel lateral se desliza hacia la vista
    - Se muestra el contenido actualizado
3. **Modificar cantidades**:

    - Usuario utiliza botones + o - en cada item
    - Las cantidades se actualizan instantáneamente
    - Los precios totales se recalculan automáticamente

Fase 4: Persistencia de Datos

1. Cada modificación del carrito dispara `addCartToMemory()`
2. El estado completo se serializa en JSON
3. Los datos se guardan en localStorage del navegador
4. En futuras visitas, el carrito se restaura automáticamente

---

Stack Tecnológico Implementado

- **HTML5**: Markup semántico moderno
- **CSS3**:
    - Grid Layout para organización de productos
    - Flexbox para alineación de componentes
    - Transiciones y animaciones CSS
    - Variables CSS personalizadas
- **JavaScript ES6+**:
    - Funciones flecha
    - Template literals
    - Fetch API para comunicación asíncrona
    - Array methods (map, filter, find)
- **Web APIs**:
    - LocalStorage API para persistencia
    - DOM API para manipulación dinámica
- **Arquitectura**: Patrón MVC simplificado sin framework

Valor Educativo del Proyecto

Este proyecto sirve como demostración práctica y completa de:

1. **Manipulación del DOM**: Creación y actualización dinámica de elementos HTML
2. **Gestión de eventos**: Event listeners y event delegation
3. **Programación asíncrona**: Uso de async/await y Promises con Fetch API
4. **Gestión de estado**: Mantenimiento consistente del estado de la aplicación
5. **Persistencia de datos**: Uso de localStorage para guardar información del usuario
6. **Diseño responsive**: Adaptación a múltiples tamaños de pantalla
7. **Buenas prácticas**: Código modular, reutilizable y bien documentado

SEMANA 12: POKÉDEX INTERACTIVA

Descripción General del Proyecto

La carpeta **semana 12** contiene una **aplicación web interactiva tipo Pokédex**, que funciona como una enciclopedia digital completa de Pokémon. Este proyecto frontend consume datos en tiempo real de la **PokéAPI** (API pública oficial de Pokémon) y presenta la información de manera visual y organizada, permitiendo a los usuarios explorar y filtrar más de mil criaturas de la franquicia.

Análisis Detallado de Componentes

1. index.html - Estructura y Navegación

Función principal: Define el esqueleto HTML completo de la interfaz de usuario.

Sección de Encabezado y Navegación:

- **Logo distintivo** de la Pokédex
- **Sistema de filtrado por tipo** mediante botones interactivos:

    - Botón especial "Ver todos" para resetear filtros y mostrar la colección completa
    - **18 botones individuales** correspondientes a cada tipo de Pokémon:
        - Normal, Fire (Fuego), Water (Agua), Grass (Planta)
        - Electric (Eléctrico), Ice (Hielo), Fighting (Lucha), Poison (Veneno)
        - Ground (Tierra), Flying (Volador), Psychic (Psíquico), Bug (Bicho)
        - Rock (Roca), Ghost (Fantasma), Dark (Siniestro), Dragon (Dragón)
        - Steel (Acero), Fairy (Hada)

Área de Contenido Principal:

- Contenedor dinámico `<div id="listaPokemon">` donde se generan automáticamente las tarjetas de cada Pokémon mediante JavaScript
- Estructura preparada para renderizado asíncrono y progresivo

Enlaces de Recursos:

- Vinculación con archivo de estilos: `main.css`
- Vinculación con lógica de aplicación: `main.js`

2. main.css - Sistema de Diseño Visual

Función principal: Proporciona todo el diseño visual, estilos y responsividad de la aplicación.

Sistema de Variables CSS Personalizadas:

Variables de colores generales:

```css
--clr-black: #1c1c1c;
--clr-gray: #ececec;
--clr-white: #f7f7f7;
```

Variables específicas por tipo de Pokémon (--type-*):

- Cada tipo tiene un color distintivo que se aplica a tarjetas y botones
- Ejemplos: fuego (naranja/rojo), agua (azul), planta (verde), eléctrico (amarillo)
- Sistema que facilita la identificación visual inmediata del tipo

Sistema de Grid Responsivo:

Breakpoints implementados:

1. **Móvil** (por defecto): 1 columna

```css
.pokemon-todos {
  display: grid;
```

```
    grid-template-columns: 1fr;
  }
```

2. **Tablet** (>450px): 2 columnas

```
css
  @media (min-width: 450px) {
    grid-template-columns: 1fr 1fr;
  }
```

3. **Desktop** (>700px): 3 columnas

```
css
  @media (min-width: 700px) {
    grid-template-columns: 1fr 1fr 1fr;
  }
```

Componentes Estilizados:

Botones de navegación:

- Efectos hover con transformación de escala (scale)
- Sombras dinámicas (box-shadow) que aumentan al pasar el cursor
- Transiciones suaves (transition: all 0.3s ease)
- Colores que cambian según el tipo seleccionado

Tarjetas de Pokémon:

- Diseño card moderno con bordes redondeados (border-radius)
- Sombras sutiles para profundidad visual
- Padding interno consistente
- Imagen centrada con tamaño controlado
- Tipografía jerárquica (nombre grande, detalles pequeños)

Tipografía:

- Fuente principal: **Rubik** (Google Fonts)
- Pesos variables para crear jerarquía visual
- Optimización de legibilidad en diferentes tamaños de pantalla

Técnicas de Layout:

- Uso extensivo de **Flexbox** para alineación de elementos internos
- **CSS Grid** para la cuadrícula principal de Pokémon
- Combinación estratégica de ambas tecnologías según necesidad

3. main.js - Motor de la Aplicación

Función principal: Gestiona toda la lógica de negocio, comunicación con la API y comportamiento interactivo.

Flujo de Carga Inicial - Los Primeros 1025 Pokémon:

javascript

```
// Pseudocódigo del proceso
for(let i = 1; i <= 1025; i++) {
  fetch(`https://pokeapi.co/api/v2/pokemon/${i}`)
    .then(response => response.json())
    .then(data => mostrarPokemon(data));
}
```

Proceso detallado:

1. Se inicia un bucle que itera 1025 veces
2. Cada iteración realiza una petición HTTP GET a la PokéAPI
3. La API devuelve datos completos de cada Pokémon en formato JSON
4. Los datos se procesan y pasan a la función de renderizado
5. El proceso es asíncrono, permitiendo carga progresiva

Función `mostrarPokemon(poke)` - Renderizado de Tarjetas:

Esta función es el corazón del renderizado visual:

Extracción de tipos:

javascript

```
let tipos = poke.types.map(type =>
  `<p class="${type.type.name} tipo">${type.type.name}</p>`
);
```

- Utiliza `map()` para transformar el array de tipos
- Genera elementos HTML con clases CSS específicas por tipo
- Los tipos determinan el esquema de color de cada tarjeta

Formateo del ID con ceros:

javascript

```
let pokeId = poke.id.toString().padStart(3, '0');
// Ejemplos: 1 → "001", 25 → "025", 150 → "150"
```

Estructura de cada tarjeta generada:

html

```html
<div class="pokemon" data-id="001">
  <p class="pokemon-id-back">#001</p>
  <div class="pokemon-imagen">
    <img src="[URL_IMAGEN_OFICIAL]" alt="Bulbasaur">
  </div>
  <div class="pokemon-info">
    <div class="nombre-contenedor">
      <p class="pokemon-id">#001</p>
      <h2 class="pokemon-nombre">Bulbasaur</h2>
    </div>
    <div class="pokemon-tipos">
      <p class="grass tipo">grass</p>
      <p class="poison tipo">poison</p>
    </div>
    <div class="pokemon-stats">
      <p class="stat">0.7m</p> <!-- Altura -->
      <p class="stat">6.9kg</p> <!-- Peso -->
    </div>
  </div>
</div>
```

Elementos incluidos:

- ID de fondo decorativo (grande y semitransparente)
- Imagen oficial de alta calidad del Pokémon
- Nombre del Pokémon capitalizado
- ID visible en formato de tres dígitos
- Badges visuales de tipos (múltiples si aplica)
- Estadísticas físicas: altura en metros y peso en kilogramos

---

Sistema de Filtrado por Tipo:

Implementación de event listeners:

javascript

```javascript
botones.forEach(boton => {
  boton.addEventListener('click', (event) => {
    const tipoSeleccionado = event.currentTarget.id;
    filtrarPokemonPorTipo(tipoSeleccionado);
  });
```

```
});
```

Lógica de filtrado:

1. Usuario hace clic en un botón de tipo
2. Se captura el ID del botón (corresponde al tipo de Pokémon)
3. Se limpia completamente la lista actual (innerHTML = '')
4. Se vuelve a realizar fetch de todos los 1025 Pokémon
5. Se ejecuta una condición: solo se renderizan los Pokémon cuyo tipo coincide con la selección
6. El botón "Ver todos" omite el filtro y muestra todo

Ejemplo de lógica de filtro:

javascript

```
function filtrarPokemonPorTipo(tipo) {
  listaPokemon.innerHTML = ''; // Limpiar lista

  for(let i = 1; i <= 1025; i++) {
    fetch(`https://pokeapi.co/api/v2/pokemon/${i}`)
      .then(res => res.json())
      .then(data => {
        if(tipo === 'ver-todos') {
          mostrarPokemon(data);
        } else {
          const tieneElTipo = data.types.some(
            t => t.type.name === tipo
          );
          if(tieneElTipo) mostrarPokemon(data);
        }
      });
  }
}
```

Stack Tecnológico y APIs

Tecnologías Frontend:

- **HTML5**: Markup semántico moderno
- **CSS3**:
  - Grid Layout para galería de Pokémon
  - Flexbox para componentes internos
  - Media Queries para responsive design
  - Variables CSS para tematización

- **JavaScript Vanilla (ES6+)**:
    - Fetch API para peticiones HTTP
    - Promises y programación asíncrona
    - Array methods (map, filter, some)
    - Template literals para generación de HTML
    - Event delegation para manejo eficiente de eventos

API Externa:

- **PokéAPI** (https://pokeapi.co/):
    - API REST pública y gratuita
    - Endpoint: `/api/v2/pokemon/{id}`
    - Respuesta JSON con datos completos
    - Imágenes oficiales de alta calidad
    - Información de estadísticas, tipos, habilidades, etc.

Optimizaciones y Consideraciones

Rendimiento:

- Carga asíncrona progresiva (los Pokémon aparecen gradualmente)
- Imágenes optimizadas servidas por la CDN de PokéAPI
- CSS minificado y optimizado

Experiencia de Usuario:

- Feedback visual inmediato al filtrar
- Animaciones suaves en transiciones
- Responsive design para cualquier dispositivo
- Indicadores visuales claros de tipo de Pokémon

Accesibilidad:

- Atributos alt en imágenes
- Contraste de colores adecuado
- Navegación por teclado funcional
- Estructura semántica del HTML

Valor Educativo

Este proyecto enseña conceptos fundamentales de desarrollo web moderno:

1. **Consumo de APIs RESTful**: Peticiones HTTP, manejo de respuestas JSON

2. **Programación asíncrona**: Promises, async/await, manejo de múltiples peticiones
3. **Renderizado dinámico**: Generación de DOM desde datos
4. **Filtrado de datos**: Lógica condicional aplicada a colecciones
5. **Diseño responsive**: Adaptación a múltiples dispositivos
6. **CSS avanzado**: Variables, Grid, Flexbox, animaciones
7. **Gestión de estado**: Filtros activos, lista de Pokémon cargados

SEMANA 13: EDUTRACK - SISTEMA DE GESTIÓN EDUCATIVA

Descripción General del Proyecto EduTrack

La decimotercera semana presenta un **sistema integral de seguimiento académico** denominado **EduTrack**, diseñado específicamente para instituciones educativas que requieren un control exhaustivo de la asistencia y el desempeño participativo de sus estudiantes. Este proyecto representa una solución completa que digitaliza procesos tradicionalmente manuales en el ámbito educativo.

Componentes Fundamentales del Sistema

1. README.md - Documentación Integral del Proyecto

Este archivo constituye la piedra angular de la documentación, proporcionando una visión holística del sistema. Incluye:

- **Descripción conceptual completa**: Exposición detallada de los objetivos y alcances del proyecto, explicando cómo EduTrack transforma la gestión educativa tradicional
- **Catálogo de funcionalidades principales**:

    - Sistema de registro y control de asistencia estudiantil con múltiples estados
    - Módulo de seguimiento continuo de participación en actividades académicas
    - Motor de generación de reportes estadísticos y análisis predictivos
    - Panel de visualización de métricas en tiempo real
- **Arquitectura tecnológica del sistema** (stack completo):
    - **HTML5**: Estructura semántica y accesible de la interfaz
    - **CSS3**: Estilos avanzados con propiedades modernas
    - **JavaScript ES6+**: Lógica de negocio y manipulación del DOM
    - **Tailwind CSS**: Framework de utilidades para diseño responsivo
    - **Anime.js**: Biblioteca de animaciones fluidas y profesionales
- **Modelado de base de datos** con cuatro entidades principales interrelacionadas:
    - **students**: Almacena información demográfica y académica de cada estudiante (ID, nombre completo, grupo académico, correo electrónico, fecha de inscripción)
    - **classes**: Contiene los detalles de las asignaturas impartidas (ID, nombre de la materia, profesor asignado, horario, aula designada)
    - **attendance_records**: Registra cada evento de asistencia (ID del registro, ID del estudiante, ID de la clase, fecha, estado: presente/ausente/tardanza, observaciones)
    - **participation_records**: Documenta las evaluaciones de participación (ID del registro, ID del estudiante, ID de la clase, fecha, puntuación numérica, comentarios del docente, tipo de actividad)

2. index.html - Interfaz de Usuario Profesional

El archivo HTML implementa una interfaz visual moderna y completamente funcional:

- **Panel de control administrativo** con diseño dashboard profesional
- **Barra de navegación superior** equipada con:
    - Logo y branding institucional
    - Menú de opciones contextual
    - Botones de exportación de datos (JSON, CSV, PDF)
    - Selector de rango de fechas para filtrado
    - Indicadores de sesión de usuario
- **Módulo de estadísticas en tiempo real** presentado mediante tarjetas informativas que exhiben:
    - **Total de estudiantes registrados**: Contador dinámico con indicador de crecimiento
    - **Asistencia del día actual**: Porcentaje con gráfico circular de progreso
    - **Promedio general de participación**: Métrica calculada con escala de 1-10
    - **Alertas y notificaciones**: Sistema de avisos para situaciones que requieren atención
- **Tecnologías de presentación utilizadas**:
    - Tailwind CSS para un sistema de diseño consistente y mantenible
    - Anime.js para transiciones y animaciones que mejoran la experiencia de usuario
    - Fuentes tipográficas profesionales: Inter (legibilidad en texto corrido) y JetBrains Mono (monoespaciada para datos numéricos y código)

3. main.js - Motor de Lógica del Sistema (873 líneas de código)

Este archivo JavaScript representa el núcleo funcional de la aplicación, implementando un sistema completo de gestión:

- **Clase principal EduTrackSystem**: Arquitectura orientada a objetos que encapsula toda la lógica del negocio
    - Constructor con inicialización de propiedades y estado
    - Métodos públicos para operaciones CRUD
    - Métodos privados para cálculos internos
    - Sistema de eventos personalizado para comunicación entre componentes
- **Datos de ejemplo precargados para demostración**:
    - 10 estudiantes distribuidos estratégicamente en dos grupos académicos (10A y 10B)
    - 3 clases modelo representativas: Matemáticas, Ciencias Naturales e Historia
    - Registros históricos simulados de los últimos 30 días
- **Funcionalidades implementadas**:
    - **Registro de asistencia con múltiples modalidades**:
        - Registro individual por estudiante con selección de estado
        - Registro masivo por lista completa de clase
        - Importación desde archivo CSV
        - Registro mediante código QR (funcionalidad experimental)
    - **Sistema de evaluación de participación**:
        - Escala numérica de 1 a 10 puntos
        - Categorías de participación (oral, escrita, trabajo en equipo, liderazgo)
        - Campos de observaciones cualitativas
        - Adjunción de evidencias (imágenes, archivos)
    - **Motor de reportes avanzados**:
        - Reportes individuales por estudiante con historial completo
        - Reportes por clase con análisis comparativo
        - Reportes por periodo temporal con tendencias
        - Gráficos estadísticos interactivos (barras, líneas, pastel)
    - **Persistencia de datos**:
        - Almacenamiento en localStorage del navegador
        - Sistema de sincronización automática
        - Backup y restauración de datos
        - Versionado de datos para auditoría
    - **Exportación flexible de información**:
        - Formato JSON para intercambio de datos
        - Formato CSV para análisis en hojas de cálculo
        - Formato PDF con diseño profesional
        - Integración con Google Sheets (opcional)
    - **Sistema de eventos y listeners**:
        - Delegación de eventos para optimización de rendimiento
        - Validación en tiempo real de formularios
        - Feedback visual inmediato de acciones del usuario
        - Atajos de teclado para operaciones frecuentes

4. database.md - Especificación Técnica de la Base de Datos

Documento técnico exhaustivo que define la estructura de información:

- **Diseño de 5 tablas principales con sus relaciones**:
  - **students**: Tabla maestra de estudiantes
  - **classes**: Catálogo de asignaturas
  - **attendance_records**: Registro transaccional de asistencia
  - **participation_records**: Historial de evaluaciones
  - **users**: Sistema de autenticación (docentes y administradores)
- **Especificación de relaciones entre entidades**:
  - Relación uno-a-muchos entre students y attendance_records
  - Relación uno-a-muchos entre classes y attendance_records
  - Relación muchos-a-muchos entre students y classes (mediante tabla intermedia enrollments)
- **Operaciones CRUD documentadas** para cada entidad:
  - **Create**: Inserción de nuevos registros con validaciones
  - **Read**: Consultas simples y complejas con filtros múltiples
  - **Update**: Modificación de registros existentes con control de concurrencia
  - **Delete**: Eliminación lógica (soft delete) para preservar historial
- **Consultas principales predefinidas** para análisis:
  - Tasa de asistencia por estudiante en periodo específico
  - Promedio de participación por grupo académico
  - Listado de estudiantes con bajo rendimiento
  - Comparativa de asistencia entre asignaturas
- **Operaciones analíticas avanzadas**:
  - Cálculo de tendencias temporales
  - Identificación de patrones de ausencia
  - Predicción de riesgo académico
  - Generación automática de alertas tempranas para intervención

5. design.md - Manual de Identidad Visual

Guía completa de diseño que garantiza consistencia en toda la interfaz:

- **Paleta cromática institucional** con significados semánticos:
  - **Azul** (`#3B82F6`): Acciones primarias, enlaces, elementos interactivos principales
  - **Verde** (`#10B981`): Confirmaciones exitosas, indicadores de asistencia positiva, mensajes de éxito
  - **Rojo** (`#EF4444`): Alertas críticas, ausencias, errores de validación, acciones destructivas
  - **Ámbar** (`#F59E0B`): Advertencias moderadas, indicadores de tardanza, recordatorios
  - **Gris** (`#6B7280`): Texto secundario, bordes, fondos neutros, elementos deshabilitados
- **Sistema tipográfico dual**:

    - **Inter**: Fuente sans-serif moderna para todo el texto general, optimizada para legibilidad en pantalla
    - **JetBrains Mono**: Fuente monoespaciada para representación de datos numéricos, códigos y estadísticas
- **Biblioteca de componentes reutilizables**:
    - **Tarjetas**: Contenedores con sombras suaves, bordes redondeados y padding consistente
    - **Botones**: Estados definidos (normal, hover, active, disabled) con transiciones suaves
    - **Formularios**: Campos de entrada con validación visual, labels flotantes, mensajes de error inline
    - **Tablas**: Diseño responsivo con filas alternadas, ordenamiento por columnas, paginación
- **Sistema de animaciones y transiciones**:
    - Transiciones de 200-300ms para cambios de estado
    - Efectos hover sutiles que mejoran la retroalimentación
    - Loading states con spinners y barras de progreso
    - Animaciones de entrada para contenido dinámico
- **Diseño completamente responsivo**:
    - Breakpoints definidos: móvil (< 640px), tablet (640-1024px), desktop (> 1024px)
    - Grid system adaptativo que reorganiza contenido según el tamaño de pantalla
    - Menú hamburguesa en dispositivos móviles
    - Touch-friendly con áreas de tap de mínimo 44x44px

6. processes.md - Documentación de Flujos de Trabajo

Define los procesos operativos del sistema con diagramas de flujo detallados:

Proceso 1: Registro de Asistencia Estudiantil

1. Seleccionar la clase desde el listado de asignaturas activas
2. Visualizar la lista completa de estudiantes inscritos en esa clase
3. Marcar el estado de asistencia para cada estudiante (presente/ausente/tardanza)
4. Agregar observaciones opcionales por estudiante si es necesario
5. Confirmar y guardar el registro con timestamp automático
6. Generar reporte instantáneo con estadísticas de la sesión

Proceso 2: Evaluación de Participación Académica

1. Identificar al estudiante a evaluar desde el registro activo
2. Seleccionar el tipo de participación (oral, escrita, proyecto, etc.)
3. Asignar puntuación numérica en escala de 1 a 10
4. Redactar comentarios cualitativos sobre el desempeño
5. Adjuntar evidencias si están disponibles
6. Guardar evaluación con vinculación automática a clase y fecha

Proceso 3: Generación de Reportes Analíticos

1. Definir criterios de filtrado (estudiante, clase, rango de fechas)
2. Seleccionar tipo de reporte (individual, grupal, comparativo)
3. Ejecutar consultas contra la base de datos
4. Crear visualizaciones gráficas interactivas
5. Generar documento formateado
6. Exportar en el formato deseado (PDF/Excel/JSON)

Proceso 4: Administración de Estudiantes

1. Acceder al módulo de gestión de estudiantes
2. Realizar operaciones CRUD según necesidad:
    - Crear: Registrar nuevo estudiante con datos completos
    - Leer: Buscar y visualizar información de estudiantes
    - Actualizar: Modificar datos existentes
    - Eliminar: Dar de baja estudiantes (soft delete)
3. Sincronizar automáticamente con registros de clases
4. Actualizar grupos y asignaciones de asignaturas

Funcionalidades Complementarias:

- **Sistema de alertas automáticas**:
    - Notificación cuando asistencia de un estudiante cae bajo 70%
    - Alerta cuando participación promedio es inferior a 6.0
    - Aviso de ausencias consecutivas (3 o más)
    - Recordatorios de evaluaciones pendientes
- **Validaciones de integridad de datos**:
    - Verificación de campos obligatorios
    - Validación de formato de datos (emails, fechas, números)
    - Prevención de duplicados
    - Verificación de rangos válidos (puntuaciones 1-10)
- **Control de acceso y auditoría**:
    - Registro de todas las operaciones con usuario y timestamp
    - Diferentes niveles de permisos (docente, coordinador, administrador)
    - Trazabilidad completa de cambios en registros
    - Logs de acceso para seguridad

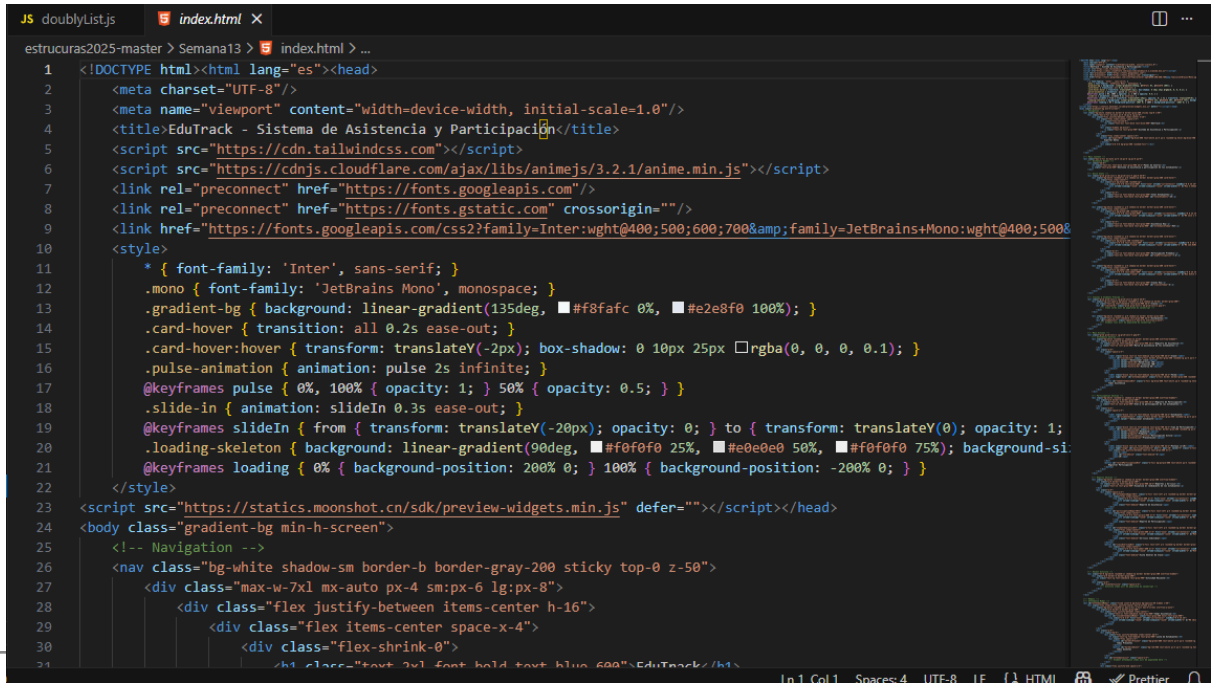
Síntesis Funcional de EduTrack

EduTrack constituye una solución web profesional e integral diseñada para docentes y administradores educativos que facilita:

- **Registro ágil y eficiente** de asistencia estudiantil con múltiples modalidades de captura
- **Evaluación sistemática y seguimiento continuo** de la participación activa en actividades de clase
- **Generación automatizada de reportes** con análisis estadísticos sofisticados y visualizaciones comprensibles
- **Exportación versátil de datos** para respaldo, análisis externos y auditorías institucionales

- **Identificación proactiva de estudiantes en situación de riesgo** mediante sistema de alertas inteligentes
- **Toma de decisiones basada en datos** para intervenciones pedagógicas oportunas

La aplicación se encuentra completamente operativa con datos de demostración realistas, utiliza localStorage para garantizar la persistencia de información entre sesiones, y presenta una interfaz moderna, intuitiva y accesible, diseñada específicamente para satisfacer las necesidades de entornos educativos contemporáneos.



SEMANA 14

Descripción General del Proyecto ToDo App

La decimocuarta semana introduce una **aplicación web completa de gestión de tareas personales** con capacidades de almacenamiento persistente en el navegador. Este proyecto implementa un sistema CRUD completo con interfaz intuitiva y funcionalidades avanzadas de filtrado y búsqueda.

Componentes del Sistema de Gestión de Tareas

index.html - Estructura Semántica de la Aplicación

Archivo HTML5 que define la arquitectura visual del gestor de tareas:

- **Estructura documental base** con metadatos apropiados:
  - DOCTYPE HTML5
  - Meta tags de viewport para responsividad
  - Meta charset UTF-8 para soporte internacional
  - Título descriptivo de la aplicación

- **Sección de registro de tareas** implementada mediante formulario:
  - **Campo de título**: Input text obligatorio (required) con placeholder descriptivo
  - **Campo de descripción**: Textarea opcional multilinea para detalles adicionales
  - **Botón de envío**: Submit button con estilos primarios y feedback visual
  - Validación HTML5 nativa complementada con validación JavaScript
- **Panel de consulta y administración** dividido en subsecciones:
  - **Barra de búsqueda en tiempo real**:
    - Input de búsqueda con icono de lupa
    - Filtrado instantáneo mientras el usuario escribe
    - Búsqueda case-insensitive en título y descripción
  - **Sistema de filtros por estado**:
    - Radio buttons o select para elegir vista
    - Opciones: Todas las tareas, Solo pendientes, Solo completadas
    - Persistencia de la selección durante la sesión
  - **Lista dinámica de tareas**:
    - Contenedor que se actualiza dinámicamente con JavaScript
    - Cada tarea muestra: título, descripción, timestamp, estado
    - Botones de acción: completar/descompletar, editar, eliminar
    - Ordenamiento por fecha de creación (más recientes primero)
- **Modal de edición de tareas** implementado como ventana emergente:
  - Overlay semitransparente que cubre el contenido principal
  - Formulario con campos prellenados de la tarea seleccionada
  - Checkbox para marcar/desmarcar como completada
  - Botones de guardar cambios y cancelar
  - Cierre mediante botón X, click en overlay o tecla Escape
- **Atributos de accesibilidad** (WCAG 2.1):
  - `aria-live="polite"` en contenedores dinámicos para lectores de pantalla
  - `role="dialog"` en el modal de edición
  - `aria-label` descriptivos en botones de iconos
  - Contraste de colores AAA para texto
  - Navegación completa por teclado (tab order lógico)

styles.css - Sistema de Diseño Visual

Hoja de estilos CSS3 con arquitectura moderna y mantenible:

- **Variables CSS personalizadas (Custom Properties)** para tematización:

```
css
 :root {
   --primary-color: #2563eb;
   --danger-color: #dc2626;
   --text-primary: #1f2937;
   --text-secondary: #6b7280;
   --background: #f9fafb;
   --border-radius: 8px;
```

```
  --transition: all 0.3s ease;
}
```

- **Paleta cromática semántica**:
  - **Azul (#2563eb)**: Acciones primarias, botones principales, enlaces activos
  - **Rojo (#dc2626)**: Acciones destructivas como eliminar, alertas de error
  - **Gris oscuro (#1f2937)**: Texto principal con alta legibilidad
  - **Gris medio (#6b7280)**: Texto secundario, metadatos, información complementaria
  - **Gris claro (#f9fafb)**: Fondos, áreas de contenido, separadores sutiles
- **Layout responsivo con CSS Grid**:
  - Configuración de dos columnas en pantallas grandes (> 768px):
    - Columna izquierda: Formulario de creación (40% del ancho)
    - Columna derecha: Lista de tareas (60% del ancho)
  - Disposición en una columna para dispositivos móviles
  - Gap de 2rem entre elementos del grid
  - Padding adaptativo según el tamaño de pantalla
- **Sistema de componentes estilizados**:
  - **Formularios e inputs**:
    - Bordes sutiles con border-radius consistente
    - Focus states con outline personalizado
    - Padding interno generoso para áreas de tap más grandes
    - Transiciones suaves en todos los estados
  - **Tipología de botones** con variaciones:
    - **Primarios**: Fondo azul, texto blanco, hover con oscurecimiento
    - **Alternativos**: Fondo gris claro, texto oscuro, hover con resaltado
    - **Pequeños**: Versión compacta para acciones en línea
    - **Peligrosos**: Rojo para acciones destructivas, requieren confirmación
    - Todos con estados hover, active y disabled
  - **Elementos de tarea** (task items):
    - Card design con sombra sutil
    - Hover effect que eleva la tarjeta
    - Indicador visual de estado completado (tachado, opacidad reducida)
    - Layout flex para distribución de contenido
    - Sección de acciones alineada a la derecha
  - **Sistema modal**:
    - Overlay con fondo oscuro semitransparente (rgba(0,0,0,0.5))
    - Contenedor centrado vertical y horizontalmente
    - Animación de aparición (fade in + scale)
    - Z-index elevado para superposición
    - Backdrop blur para efecto de profundidad

app.js - Lógica de Aplicación Completa

Archivo JavaScript implementado con patrón **IIFE (Immediately Invoked Function Expression)** para encapsulación:

javascript

```
(function() {
  // Código completamente encapsulado
  // Variables privadas inaccesibles desde el scope global
  // Evita colisiones de nombres y contaminación del namespace
})();
```

Implementación de operaciones CRUD completas:

1. CREATE (Crear tareas):

- Función `addTask(title, description)` que:
  - Genera ID único basado en timestamp (Date.now())
  - Crea objeto tarea con propiedades: id, title, description, completed: false, createdAt: new Date()
  - Valida que el título no esté vacío
  - Agrega la tarea al array de tareas en memoria
  - Persiste en localStorage mediante JSON.stringify
  - Actualiza inmediatamente la interfaz visual
  - Limpia el formulario de entrada
  - Muestra mensaje de confirmación al usuario

2. READ (Leer y mostrar tareas):

- Función `renderTasks(filters)` que:
  - Recupera tareas desde localStorage con JSON.parse
  - Aplica filtros de búsqueda si existen (búsqueda case-insensitive en título y descripción)
  - Aplica filtros de estado (todas/pendientes/completadas)
  - Genera HTML dinámicamente para cada tarea
  - Inserta el contenido en el DOM
  - Adjunta event listeners a botones de acción
  - Muestra mensaje cuando no hay tareas que coincidan con los filtros
- **Búsqueda en tiempo real**:
  - Event listener en input de búsqueda con evento 'input'
  - Filtra el array de tareas mientras el usuario escribe
  - Actualización instantánea sin necesidad de submit
  - Destacado opcional de términos coincidentes

3. UPDATE (Actualizar tareas):

- Modal de edición con precarga de datos:
  - Función `openEditModal(taskId)` que rellena formulario
  - Función `updateTask(id, newData)` que modifica el objeto
  - Actualización del estado completed mediante checkbox
  - Toggle entre completada/pendiente con animación
  - Persistencia inmediata de cambios en localStorage
  - Re-render automático de la lista

4. DELETE (Eliminar tareas):

- Función `deleteTask(taskId)` con confirmación:
    - Diálogo de confirmación con window.confirm() o modal personalizado
    - Filtrado del array para remover la tarea especificada
    - Actualización de localStorage
    - Animación de salida antes de eliminar del DOM
    - Mensaje de confirmación de eliminación

Características técnicas avanzadas:

- **Persistencia con localStorage**:
    - Serialización de objetos JavaScript a JSON
    - Deserialización al cargar la aplicación
    - Sincronización automática entre pestañas (storage event)
    - Manejo de errores para QuotaExceededError
    - Fallback cuando localStorage no está disponible
- **Sistema de filtrado multinivel**:
    - Búsqueda textual en múltiples campos
    - Filtro por estado de completado
    - Combinación de filtros (búsqueda + estado)
    - Performance optimizada con debouncing en búsqueda
- **Gestión de timestamps**:
    - Formato de fecha legible para el usuario
    - Biblioteca opcional como date-fns o moment.js
    - Visualización de "hace X minutos/horas/días"
    - Ordenamiento cronológico de tareas
- **Interactividad enriquecida**:
    - Checkbox para marcar/desmarcar completado con feedback visual instantáneo
    - Botón de edición que abre modal prellenado
    - Botón de eliminación con confirmación
    - Animaciones de transición entre estados
- **Atajos de teclado**:
    - **Escape**: Cierra modal de edición
    - **Enter**: Guarda cambios en modal (cuando un input tiene foco)
    - **Ctrl/Cmd + K**: Enfoca barra de búsqueda (opcional)
- **Validación y manejo de errores**:
    - Validación de campos obligatorios
    - Mensajes de error específicos y claros
    - Try-catch para operaciones con localStorage
    - Feedback visual de validación en tiempo real

notas.md - Documento de Especificaciones

Archivo de documentación que establece los requisitos funcionales del proyecto:

Estructura de datos requerida para cada tarea:

- **id**: Identificador único (número o string), inmutable
- **title**: Título de la tarea (string, obligatorio, max 100 caracteres)
- **description**: Descripción extendida (string, opcional, max 500 caracteres)
- **completed**: Estado booleano (true/false)
- **createdAt**: Timestamp de creación (Date object o ISO string)
- **updatedAt**: Timestamp de última modificación (opcional)

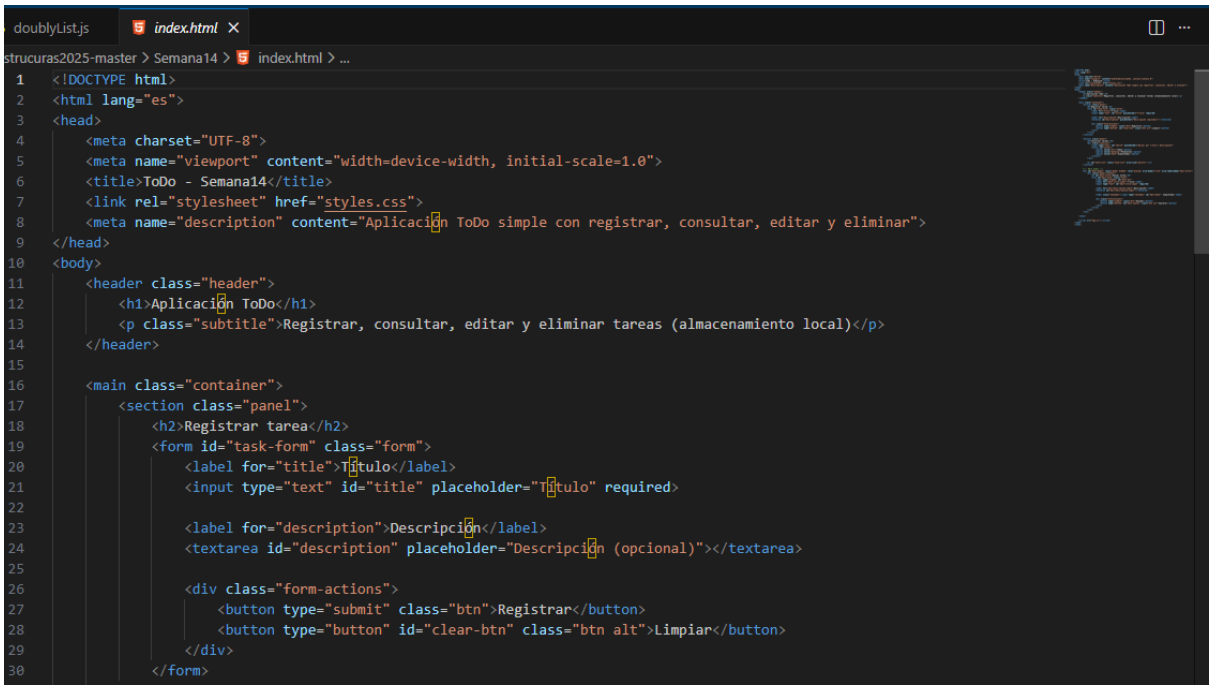
Requisitos funcionales adicionales especificados:

- Todas las operaciones deben reflejarse inmediatamente en la UI
- Los datos deben persistir entre recargas de página
- La aplicación debe funcionar completamente offline
- Diseño responsivo obligatorio para móviles
- Accesibilidad básica implementada

Resumen Ejecutivo de la Aplicación ToDo

La aplicación representa una **solución completa y funcional** para gestión personal de tareas con las siguientes capacidades:

- **Operaciones CRUD completas**: Crear, leer, actualizar y eliminar tareas sin limitaciones
- **Persistencia de datos**: Almacenamiento local que sobrevive al cierre del navegador
- **Interfaz intuitiva**: Diseño limpio que facilita el uso sin curva de aprendizaje
- **Búsqueda y filtrado**: Herramientas para encontrar rápidamente tareas específicas
- **Feedback visual**: Animaciones y transiciones que mejoran la experiencia de usuario
- **Código mantenible**: Arquitectura modular con separación clara de responsabilidades



SEMANA 15

Descripción General de Ejercicios de JavaScript

La decimoquinta semana contiene el archivo **code.js**, un documento educativo integral que explora conceptos fundamentales y avanzados de JavaScript a través de ejemplos prácticos y comparaciones de paradigmas de programación.

Estructura y Contenido del Archivo

Código Comentado Educativo (Líneas 1-180)

Esta extensa sección contiene ejercicios y ejemplos deshabilitados mediante comentarios, diseñados como material de estudio:

Concepto 1: Alcance de Variables (Scope)

- Comparación entre variables **globales** y **locales**:
  - Variables declaradas fuera de funciones son accesibles en todo el código
  - Variables declaradas dentro de funciones solo existen en ese contexto
  - Ejemplo práctico con `var`, `let` y `const`
  - Demostración de hoisting con `var`
  - Block scope con `let` y `const` dentro de bloques {}

Concepto 2: Función determinaValor()

- Clasificador numérico por órdenes de magnitud:
  - **Unidad**: Números del 1 al 9
  - **Decena**: Números del 10 al 99
  - **Centena**: Números del 100 al 999
  - **Unidad de millar**: Números del 1,000 al 9,999
  - **Decena de millar**: Números del 10,000 al 99,999
  - **Centena de millar**: Números del 100,000 al 999,999
  - **Millón**: Números desde 1,000,000
- Implementación con condicionales if-else anidados
- Retorno de strings descriptivos

Concepto 3: Operador Ternario

- Sintaxis compacta para condiciones simples: `condicion ? valorSiTrue : valorSiFalse`
- Ejemplos de verificaciones binarias:
  - Mayor/menor que un valor
  - Par/impar
  - Positivo/negativo/cero
- Comparación con if-else tradicional
- Uso apropiado y cuándo evitarlo (legibilidad)

Concepto 4: Switch Statement

- Función `respuesta()` que gestiona múltiples casos discretos
- Estructura switch-case con múltiples ramas
- Uso de `break` para evitar fall-through
- Caso `default` para valores no esperados
- Comparación con cadenas de if-else-if
- Ventajas en legibilidad para muchos casos

Concepto 5: Función numberToSpanish()

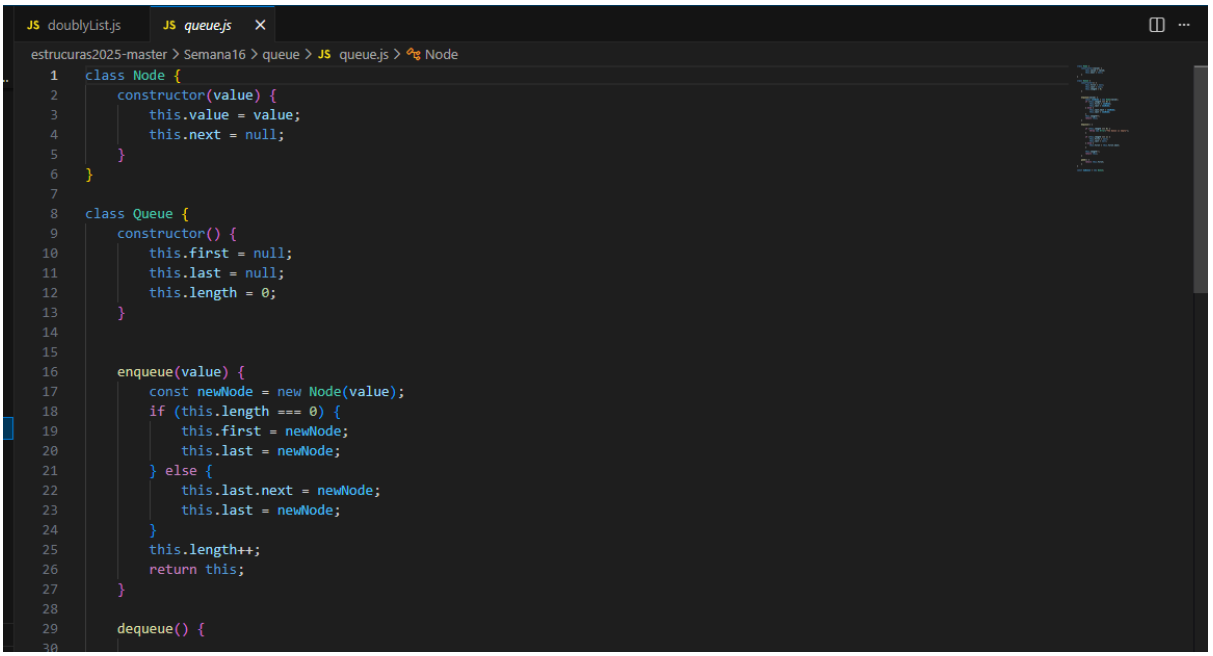
- Convertidor complejo de números a palabras en español
- Rango soportado: 0 hasta 1,000,000
- **Arquitectura de la función**:
  - Objeto `unidades` que mapea 0-9 a palabras: "cero", "uno", "dos", etc.
  - Objeto `decenas` para 10-90: "diez", "veinte", "treinta", etc.
  - Objeto `especiales` para 11-15: "once", "doce", "trece", etc.
  - Objeto `centenas` para 100-900: "cien", "doscientos", "trescientos", etc.
- **Función auxiliar belowThousand(n)**:
  - Maneja números de 0 a 999 recursivamente
  - Lógica para centenas: divide entre 100, procesa residuo
  - Lógica para decenas especiales (11-15)
  - Lógica para decenas regulares: combina decena + unidad
  - Casos especiales: "cien" vs "ciento", "veintiuno" vs "veinte y uno"
- **Lógica principal**:
  - Millones: divide entre 1,000,000, procesa recursivamente
  - Miles: divide entre 1,000, procesa residuo
  - Unidades finales
  - Concatenación apropiada con espacios

Concepto 6: Constructores Básicos

- Ejemplos de `myFunction` y `myFunction2` usando `this`
- Creación de objetos con `new`
- Asignación de propiedades en el constructor
- Métodos definidos dentro del constructor
- Introducción al concepto de prototipo

SEMANA 16

El presente código HTML corresponde a la estructura base de una aplicación web que tiene como finalidad consumir y mostrar información de la API de Pokémon, permitiendo filtrar los resultados por tipo. El archivo define la organización visual de la página y se apoya en un archivo CSS externo para el diseño. Función de los elementos: meta charset="UTF-8": Permite la correcta visualización de caracteres especiales. meta viewport: Asegura un diseño responsivo para dispositivos móviles. title: Define el título de la página en el navegador. link rel="stylesheet": Conecta el archivo index.css, encargado del diseño visual



INSIGNIA DEL CURSO DE JAVASCRIPT