

INSTITUTO TECNOLÓGICO SUPERIOR DE LAS SIERRA

NEGRA DE AJALPAN

INGENIERIA EN SISTEMAS COMPUTACIONALES

MATERIA: ESTRUCTURA DE DATOS

PORTAFOLIO DE EVIDENCIAS

DOCENTE: ARTURO BUSTAMANTE LAZCANO

ALUMNA: ARELI NATALY OLMOS COELLO

SEMESTRE: TERCERO

Contenido

[Semana 1 — Introducción práctica a JavaScript 5](#_Toc216673032)

[Lenguajes utilizados 5](#_Toc216673033)

[Objetivo del repositorio 5](#_Toc216673034)

[Descripción del contenido 5](#_Toc216673035)

[Semana 2 — Fundamentos de HTML y navegación web 9](#_Toc216673036)

[Lenguajes utilizados 9](#_Toc216673037)

[Objetivo del repositorio 9](#_Toc216673038)

[Descripción del contenido 10](#_Toc216673039)

[Semana 3 — Bootstrap, diseño responsivo y estructuras de datos 13](#_Toc216673040)

[Lenguajes utilizados 13](#_Toc216673041)

[Descripción del contenido 13](#_Toc216673042)

[Semana 4 — JavaScript: Números y Grafos 17](#_Toc216673043)

[Lenguajes utilizados 17](#_Toc216673044)

[Objetivo 17](#_Toc216673045)

[Descripción del contenido 18](#_Toc216673046)

[Análisis del contenido 19](#_Toc216673047)

[Semana 5 — Fundamentos de JavaScript y Estructuras Lineales 21](#_Toc216673048)

[Lenguajes utilizados 21](#_Toc216673049)

[Objetivo de la semana 22](#_Toc216673050)

[Descripción del contenido 22](#_Toc216673051)

[Análisis de contenido 26](#_Toc216673052)

[Semana 6 — Colecciones, POO y estructuras lineales 27](#_Toc216673053)

[Lenguajes utilizados 27](#_Toc216673054)

[Objetivos 27](#_Toc216673055)

[Descripción 28](#_Toc216673056)

[análisis de contenido 30](#_Toc216673057)

[Semana 7 — Estructuras de datos: HashTables, Set y Stack 32](#_Toc216673058)

[Lenguajes utilizados 32](#_Toc216673059)

[Objetivos de aprendizaje 32](#_Toc216673060)

[Análisis del contenido 35](#_Toc216673061)

[Semana 8 — Listas enlazadas: estructuras simples y dobles 37](#_Toc216673062)

[Lenguajes utilizados 37](#_Toc216673063)

[Objetivos 37](#_Toc216673064)

[Descripción de contenido por archivo 37](#_Toc216673065)

[Análisis del contenido 40](#_Toc216673066)

[Semana 9 — Implementación y análisis de estructuras de datos fundamentales 41](#_Toc216673067)

[Lenguajes utilizados 41](#_Toc216673068)

[Objetivos 41](#_Toc216673069)

[Descripción de contenido por archivo 41](#_Toc216673070)

[Análisis de contenido 48](#_Toc216673071)

[Semana 10 — Integración de estructuras de datos y consumo de APIs 50](#_Toc216673072)

[Lenguajes utilizados 50](#_Toc216673073)

[análisis de contenido 50](#_Toc216673074)

[Objetivos 50](#_Toc216673075)

[Descripción de contenido 51](#_Toc216673076)

[Semana 11 — Proyecto Ecommerce: carrito de compras, persistencia y render dinámico 57](#_Toc216673077)

[Lenguajes utilizados 57](#_Toc216673078)

[análisis de contenido 57](#_Toc216673079)

[Descripción por archivo 57](#_Toc216673080)

[Semana 12 — Pokédex: consumo de API y filtrado dinámico 62](#_Toc216673081)

[Lenguajes utilizados 62](#_Toc216673082)

[análisis de contenido 62](#_Toc216673083)

[Descripción por archivo 62](#_Toc216673084)

[Semana 13 — Recursión, ordenamiento y sistema web interactivo (EduTrack) 66](#_Toc216673085)

[Lenguajes utilizados 66](#_Toc216673086)

[análisis de contenido 66](#_Toc216673087)

[Descripción de contenido 66](#_Toc216673088)

[Semana 14 — Objetos y aplicación CRUD con almacenamiento local 74](#_Toc216673089)

[Lenguajes utilizados 74](#_Toc216673090)

[Análisis de contenido 74](#_Toc216673091)

[Descripción de contenido 74](#_Toc216673092)

[Objetivos 78](#_Toc216673093)

[Semana 15 — Estructuras de datos avanzadas en JavaScript 79](#_Toc216673094)

[Lenguajes utilizados 79](#_Toc216673095)

[Objetivo 79](#_Toc216673096)

[Descripción de los archivos 79](#_Toc216673097)

[análisis de contenido 84](#_Toc216673098)

[Semana 16 — Estructuras lineales y no lineales con integración de interfaz 85](#_Toc216673099)

[Lenguajes utilizados 85](#_Toc216673100)

[Objetivo 85](#_Toc216673101)

[Descripción de los archivos 85](#_Toc216673102)

[análisis de contenido 89](#_Toc216673103)

[INSIGNIA JAVASCRIPT 90](#_Toc216673104)

# Semana 1 — Introducción práctica a JavaScript

**Descripción del repositorio**

Este repositorio contiene los ejercicios y ejemplos desarrollados durante la **primera semana** de la materia *Estructuras de Datos*. El enfoque principal es comprender cómo funciona JavaScript en un entorno web, cómo se define y ejecuta una función, y cómo se integra el código JavaScript dentro de un archivo HTML para interactuar con el usuario.

## Lenguajes utilizados

* **JavaScript (ES6)**: definición de funciones, uso de operadores, estructuras de control y ciclos.
* **HTML5**: estructura básica de la página web e interacción con JavaScript mediante eventos (onclick).
* **Consola del navegador**: visualización de resultados del ejercicio de lógica (FizzBuzz).

## Objetivo del repositorio

El objetivo de este repositorio es:

* Comprender la **sintaxis básica de JavaScript**.
* Aprender a **crear y llamar funciones**.
* Integrar JavaScript con HTML para generar **interacción con el usuario**.
* Introducir conceptos de **lógica de programación**, operadores y ciclos mediante un problema clásico.

**Estructura de archivos**

* index.html

Archivo principal HTML que muestra la interfaz y ejecuta las funciones

* code.js

## Descripción del contenido

**1. Archivo code.js**

**Texto

El contenido generado por IA puede ser incorrecto.**

Este archivo contiene el código JavaScript trabajado durante la primera práctica:

* **Función saludar()**  
  Muestra un mensaje emergente (alert) para demostrar cómo se ejecuta una función JavaScript desde un botón en HTML.
* **Función** teSaludo(nombre)  
  Recibe un parámetro y personaliza el mensaje, introduciendo el concepto de parámetros en funciones.
* **Comentarios sobre operadores**  
  Se documentan operadores de asignación, comparación y aritméticos, explicando diferencias importantes como == y ===, lo cual es clave para evitar errores lógicos en JavaScript.
* **Función fizzBuzz()**  
  Implementación del reto clásico *FizzBuzz*, utilizando:
  + Ciclo for
  + Condicionales if / else if / else
  + Operador módulo (%)

El resultado se imprime en la consola del navegador, reforzando el uso de herramientas de depuración.

**2. Archivo index.htmlTexto

El contenido generado por IA puede ser incorrecto.**

Este archivo representa la parte visual del ejercicio:

* Define la **estructura básica de una página web**.
* Incluye encabezados y textos descriptivos para guiar al usuario.
* Contiene botones que ejecutan funciones JavaScript mediante el evento onclick.
* Integra el archivo code.js usando la etiqueta <script>, mostrando cómo se conectan ambos lenguajes.

**Análisis breve del contenido**

El repositorio refleja una **introducción progresiva y práctica** a JavaScript. No se limita a

La combinación de funciones simples con el reto FizzBuzz permite:

* Entender la relación entre **lógica y sintaxis**.
* Diferenciar entre interacción visual (alertas y botones) y salida por consola.
* Sentar bases sólidas para temas posteriores como arreglos, estructuras de datos y algoritmos más complejos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

# Semana 2 — Fundamentos de HTML y navegación web

**Descripción del repositorio**

Este repositorio agrupa las prácticas correspondientes a la **Semana 2**, enfocadas en el uso de **HTML como lenguaje de marcado** para la creación de sitios web estáticos. A lo largo de los archivos se trabajan etiquetas básicas, estructura semántica, enlaces, navegación entre páginas y la integración de recursos externos.

El material muestra una evolución clara: desde el uso elemental de etiquetas HTML hasta la construcción de un pequeño sitio con múltiples páginas y una maqueta visual apoyada en Bootstrap.

## Lenguajes utilizados

* **HTML5**: estructura del documento, etiquetas de texto, listas, enlaces, imágenes y navegación.
* **JavaScript**: uso básico para funciones asociadas al sitio (carpeta *mi sitio*).
* **Bootstrap 5**: incorporación de componentes visuales y estilos prediseñados.
* **Navegador web**: visualización y validación del funcionamiento de los archivos.

## Objetivo del repositorio

El objetivo principal de esta semana es:

* Comprender la **estructura base de un documento HTML**.
* Utilizar correctamente etiquetas de encabezado, texto y formato.
* Implementar **enlaces internos y externos**.
* Construir una **navegación funcional entre varias páginas**.
* Introducir el uso de frameworks visuales para mejorar la presentación.

**Estructura de archivos**

index.html Página principal con ejemplos de etiquetas HTML

nosotros.html Página informativa con navegación

acerca.html Página descriptiva del sitio

mi\_sitio

index.html Maqueta del sitio usando Bootstrap

script.js Archivo JavaScript asociado al sitio

assets Imágenes utilizadas en la práctica

## Descripción del contenido

**Páginas HTML principales**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.**

Los archivos HTML muestran el uso progresivo de las etiquetas más comunes:

* Encabezados de distintos niveles (h1 a h6) para jerarquizar información.
* Párrafos normales y preformateados para diferenciar bloques de texto.
* Listas ordenadas y desordenadas.
* Etiquetas de formato como negritas, cursiva, subíndices y superíndices.
* Enlaces externos con control de apertura en la misma pestaña o en una nueva.
* Navegación interna entre páginas del mismo sitio.
* Inclusión de imágenes mediante rutas relativas.

Estas páginas permiten comprender cómo se organiza visual y lógicamente un sitio web básico.

**Carpeta *mi sitio***

En esta sección se desarrolla una versión más elaborada del sitio:

* Se integra **Bootstrap** para crear una barra de navegación colapsable.
* Se utilizan contenedores y clases prediseñadas para estructurar el contenido.
* Se refuerza la separación entre estructura (HTML) y comportamiento (JavaScript).

Esta parte del repositorio introduce buenas prácticas de diseño web y prepara el terreno para trabajos más completos.

**Análisis breve del contenido**

El contenido de la Semana 2 demuestra un avance significativo respecto a la semana anterior. Se pasa de ejemplos aislados a la construcción de un **sitio con múltiples páginas y navegación funcional**.

Las prácticas no solo muestran el uso de etiquetas, sino que ayudan a entender su propósito dentro de un sitio real. La inclusión de Bootstrap aporta una primera aproximación al diseño moderno, sin perder de vista los fundamentos de HTML.

En conjunto, este repositorio consolida las bases necesarias para continuar con CSS, JavaScript y estructuras más complejas en el desarrollo web.

Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# Semana 3 — Bootstrap, diseño responsivo y estructuras de datos

## Lenguajes utilizados

* **HTML5**: estructura del contenido y organización de la página.
* **CSS3**: estilos personalizados y uso de *media queries*.
* **Bootstrap 5**: sistema de contenedores, grid responsivo y variables de color.
* **JavaScript (ES6)**: implementación de estructuras de datos y lógica de programación.
* **Consola del navegador**: validación del comportamiento interno de las estructuras.

## Descripción del contenido

**Contenedores**

Se utilizan distintos tipos de contenedores para comprender cómo Bootstrap adapta el contenido según el tamaño de pantalla:

* Contenedores fijos y fluidos.
* Contenedores con *breakpoints* (sm, md, lg, xl, xxl).

Estos ejemplos permiten visualizar cuándo el contenido ocupa todo el ancho de la pantalla y cuándo se ajusta de forma controlada.

Texto

El contenido generado por IA puede ser incorrecto.

**Sistema Grid (filas y columnas)**

Se implementa el sistema Grid de Bootstrap para distribuir el contenido de forma flexible:

* Uso de filas (row) y columnas (col) con tamaño automático.
* Distribución uniforme de hasta 12 columnas.
* Asignación manual de espacios a columnas específicas (col-2, col-6, col-8).

Estos ejercicios ayudan a entender cómo se reparte el espacio horizontal y cómo se construyen layouts adaptables.

**Estilos responsivos personalizadosCaptura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.**

Mediante el archivo mistyle.css se refuerza el concepto de diseño responsivo:

* Se utilizan *media queries* para detectar cambios de tamaño de pantalla.
* El color de fondo de las columnas cambia según el *breakpoint* activo.
* Se aprovechan variables de color de Bootstrap para mantener coherencia visual.

Este enfoque facilita identificar visualmente cuándo ocurre un cambio de resolución.

**Grafos (carpetas graphs.js y buildGraph.js)**

En esta sección se trabajan distintas formas de representar un grafo utilizando el mismo conjunto de nodos y conexiones.

Se exploran representaciones mediante:

* **Lista de aristas**, donde las relaciones se muestran como pares de nodos.
* **Lista de adyacencia**, implementada tanto con arreglos como con objetos.
* **Matriz de adyacencia**, que modela las conexiones de forma matricial.

Además, se incluye una implementación orientada a objetos que permite crear vértices de forma dinámica y establecer relaciones bidireccionales, mostrando la estructura resultante en consola.

Estas variantes permiten comparar cómo cambia la organización de la información según la representación elegida.

**Lista Doblemente Enlazada (carpeta doublyLinkedList)**

Se implementa una **lista doblemente enlazada**, donde cada nodo mantiene referencia al siguiente y al anterior:

* La clase Node define la estructura básica del nodo.
* La clase DoublyLinkedList administra la cabeza, la cola y la longitud de la lista.

Se incluyen operaciones fundamentales como:

* Inserción de elementos.
* Eliminación de nodos específicos.
* Recorrido de la lista en ambos sentidos.
* Limpieza de la estructura.

**Aplicación práctica con interfaz**

Como caso de uso, la lista doblemente enlazada se emplea para crear un sistema de **navegación entre películas**:

* Cada nodo almacena un objeto con información de una película.
* Botones HTML permiten avanzar o retroceder entre los elementos.
* El contenido se actualiza dinámicamente en la interfaz.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

# Semana 4 — JavaScript: Números y Grafos

Esta semana del repositorio está enfocada en **reforzar el uso de JavaScript** a través de dos bloques principales: el manejo de **tipos de datos numéricos y funciones básicas**, y una **implementación orientada a objetos de un grafo**. El objetivo es consolidar la lógica de programación antes de avanzar hacia estructuras y algoritmos más complejos.

## Lenguajes utilizados

* **JavaScript (ES6)**: variables, funciones, ciclos, clases y estructuras dinámicas.
* **HTML**: soporte teórico y carga de scripts.
* **Consola del navegador**: principal medio para observar resultados y comportamiento del código.

**Estructura de archivos**

* Semana 4
* index.html
* code.js
* graph
  + - index.html
    - main.js

## Objetivo

Comprender el uso de **números en JavaScript**, el control de flujo mediante ciclos y funciones, y aplicar estos conocimientos en la **construcción de un grafo** utilizando clases y objetos. Esta semana busca que el estudiante practique lógica pura y observe cómo los datos se organizan y relacionan entre sí.

## Descripción del contenido

**Tipos de datos numéricos y funciones (index.html y code.js)** **Texto

El contenido generado por IA puede ser incorrecto.**Texto

El contenido generado por IA puede ser incorrecto.

En esta parte se trabaja con variables numéricas y distintas formas de representarlas, incluyendo números enteros, decimales y números legibles mediante separadores. El archivo code.js contiene varias funciones que permiten:

* Ejecutar ciclos controlados para mostrar secuencias numéricas en consola.
* Realizar operaciones aritméticas utilizando valores externos.
* Generar tablas de multiplicar en orden ascendente y descendente.

El archivo HTML muestra la diferencia entre el uso literal de números y el uso del constructor Number.

**Implementación de grafos (carpeta graph)**

Dentro de la carpeta graph se desarrolla una implementación más estructurada de un **grafo dirigido**, utilizando programación orientada a objetos.

* La clase Graph administra los nodos mediante un Map.
* La clase Node representa cada nodo y mantiene una lista de sus conexiones.
* Se incluyen métodos para agregar nodos, crear aristas y mostrar la estructura completa del grafo en consola.

El ejemplo final construye un grafo con nodos identificados por letras y establece varias conexiones entre ellos, permitiendo visualizar claramente cómo se relacionan.

## Análisis del contenido

Las funciones numéricas ayudan a reforzar el uso de ciclos, parámetros y operaciones básicas, mostrando resultados directamente en consola para facilitar su comprensión.

Por otro lado, la implementación del grafo introduce una forma más organizada de trabajar con datos relacionados. Al separar la lógica en clases y métodos, se puede observar cómo JavaScript permite modelar estructuras complejas de manera clara y escalable. Esta práctica sirve como base para comprender recorridos, relaciones y representaciones de datos que se utilizarán en temas posteriores. Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Semana 5 — Fundamentos de JavaScript y Estructuras Lineales

Este apartado del repositorio corresponde a la **Semana 5** del curso de Estructuras de Datos. El contenido se enfoca en reforzar los fundamentos de JavaScript y su relación directa con el manejo de datos, preparando el terreno para la implementación y comprensión de estructuras lineales.

## Lenguajes utilizados

* **JavaScript (ES6)**: uso de variables, constantes, funciones, objetos, asincronía y clases.
* **HTML**: archivos base para ejecutar los ejemplos directamente en el navegador.
* **API Fetch**: obtención de datos externos y lectura de archivos JSON locales.
* **Consola del navegador**: verificación del comportamiento lógico y estructural.

**Estructura de archivos**

Semana5/

├── index.html

├── inner.html

├── code.js

├── introduccion/

│ ├── functions/

│ │ ├── index.html

│ │ └── main.js

│ └── variable/

│ ├── index.html

│ └── main.js

├── json/

│ ├── index.html

│ ├── main.js

│ └── people.json

└── linkedlist/

├── images/

├── index.html

├── main.js

└── code.js

## Objetivo de la semana

El objetivo de esta semana es comprender y aplicar los conceptos básicos del lenguaje JavaScript que son necesarios para el desarrollo de estructuras de datos. Se busca afianzar el manejo de variables, funciones, objetos y asincronía, así como introducir la implementación de una estructura lineal mediante una lista enlazada.

## Descripción del contenido

**Archivos raíz**

**index.html**  
Se utilizó como página principal para ejecutar los ejercicios de la semana. Desde este archivo se cargan los scripts que trabajan con valores numéricos y funciones, permitiendo observar los resultados directamente en la consola del navegador.

Texto

El contenido generado por IA puede ser incorrecto.

**inner.html**  
Se usó como una página alternativa para comprobar que el mismo código JavaScript puede ejecutarse correctamente en más de un archivo HTML. Su propósito fue validar la reutilización del código sin realizar modificaciones en la lógica.

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

**code.js**  
En este archivo se trabajó con variables numéricas, constantes y funciones. Se realizaron recorridos mediante ciclos, operaciones aritméticas y funciones para generar resultados como sumas y tablas de multiplicar, enfocándose en el uso de parámetros y el control del flujo del programa.

Texto

El contenido generado por IA puede ser incorrecto.

**Carpeta introducción**

**functions**

* **index.html**: se utilizó únicamente como soporte para cargar el archivo JavaScript y ejecutar los ejemplos de funciones.
* **main.js**: se desarrollaron funciones que realizan cálculos a partir de constantes y variables. También se trabajó con funciones que reciben parámetros y con valores por defecto, mostrando diferentes formas de invocar una función.

**variable**

* **index.html**: archivo base para ejecutar los ejemplos relacionados con el uso de variables.
* **main.js**: se trabajó con variables declaradas en distintos contextos. Se probó cómo cambia el valor de una variable cuando se declara a nivel global, dentro de una función y dentro de un bloque, observando los resultados en consola.

**Carpeta json**

* **index.html**: se utilizó como punto de carga para ejecutar los ejemplos relacionados con objetos y datos en formato JSON.
* **main.js**: se trabajó con objetos literales que contienen propiedades simples, arreglos y objetos anidados. También se realizaron pruebas de consumo de datos usando fetch, tanto desde una API externa como desde un archivo JSON local.
* **people.json**: se utilizó como archivo de datos para simular la lectura de información estructurada, permitiendo acceder a propiedades y recorrer arreglos desde JavaScript.

**Carpeta linkedlist**

* **index.html**: se utilizó para mostrar información en pantalla y permitir la interacción mediante un botón que avanza entre elementos.
* **main.js**: se implementó una lista enlazada simple, definiendo nodos y la lógica para agregar, eliminar y recorrer elementos. La lista se aplicó a un ejemplo donde cada nodo contiene información de una película.
* **code.js**: se empleó para realizar pruebas adicionales de la lista enlazada en consola, verificando el funcionamiento de sus operaciones básicas.

## Análisis de contenido

El contenido de esta semana refuerza la relación entre los fundamentos del lenguaje JavaScript y la construcción de estructuras de datos. El manejo adecuado de variables, funciones y objetos facilita la implementación de estructuras lineales como la lista enlazada. Además, el trabajo con JSON y asincronía permite comprender cómo se manejan datos en escenarios más cercanos a aplicaciones reales, sentando una base sólida para estructuras más complejas en semanas posteriores.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

# Semana 6 — Colecciones, POO y estructuras lineales

**Resumen general**

Semana 6 se enfoca en el uso práctico de **estructuras de datos básicas y colecciones propias de JavaScript**, combinadas con conceptos de **programación orientada a objetos** y **colas**. El material incluye ejemplos con arreglos, Map, clases, herencia, paso de valores por referencia y la implementación de una cola aplicada a un flujo de solicitudes.

## Lenguajes utilizados

* JavaScript (ES6)
* HTML

**Estructura de la carpeta**

* index.html, code.js: manejo de variables, tipos de datos y arreglos con interacción básica en HTML.
* map/: uso de la colección Map con operaciones CRUD y visualización en el DOM.
* poo/:
  + clases/: definición de clases, constructores, métodos y herencia.
  + references/: comparación entre paso por valor y paso por referencia.
* queue/: implementación de una cola y aplicación a un sistema de solicitudes procesadas en orden.

## Objetivos

* Comprender el uso de arreglos y colecciones clave-valor en JavaScript.
* Aplicar programación orientada a objetos mediante clases y herencia.
* Identificar diferencias entre valores primitivos y objetos por referencia.
* Implementar y utilizar una cola para controlar el orden de procesamiento.

Descripción **Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.**

**Archivos raíz**

* **Variables y tipos**:
  + Declaración de variables string, number, boolean y number decimal.
  + Uso de typeof para mostrar el tipo de cada variable en pantalla.
* **Arreglos**:
  + Uso de un arreglo dinámico para almacenar valores ingresados por el usuario.
  + Inserción de elementos con push().
  + Recorrido del arreglo con forEach().
  + Eliminación y edición de elementos a partir de su índice.

**Colección Map**

* Operaciones básicas:
  + set(key, value): insertar y actualizar valores.
  + get(key): obtener un valor a partir de su clave.
  + has(key): verificar existencia de una clave.
  + delete(key): eliminar un elemento.
  + clear(): vaciar la colección.
* Claves avanzadas:
  + Uso de objetos como claves dentro del Map.
  + Comparación de referencias entre objetos.
* Recorridos:
  + Iteración por valores (map.values()).
  + Iteración por claves (map.keys()).
  + Iteración completa clave–valor.
* Aplicación práctica:
  + Almacenamiento de pares código–cliente desde una interfaz HTML.
  + Renderizado dinámico del contenido en el DOM.

**Programación Orientada a Objetos (POO)**

* **Clases**:
  + Definición de clases mediante class.
  + Uso de constructores para inicializar atributos.
  + Definición de métodos de instancia.
* **Herencia**:
  + Uso de extends para crear una clase hija.
  + Llamada al constructor padre con super().
  + Sobrescritura de métodos.
* **Objetos**:
  + Creación de objetos mediante literales y mediante clases.

**Valores por referencia**

* Valores primitivos:
  + Paso de argumentos por valor.
  + Modificación interna sin afectar la variable original.
* Objetos:
  + Paso por referencia.
  + Modificación directa del objeto original desde una función.

**Cola (Queue)**

* Implementación de la estructura:
  + Clase Queue con encapsulación mediante campo privado.
  + Métodos principales:
    - enqueue(item): agregar un elemento al final.
    - dequeue(): eliminar el primer elemento.
    - peek(): consultar el primer elemento.
    - isEmpty(): validar si la cola está vacía.
    - size(): obtener el tamaño.
* Aplicación práctica:
  + Uso de la cola para almacenar solicitudes.
  + Procesamiento secuencial respetando el orden de llegada (FIFO).
  + Simulación de peticiones HTTP usando fetch y retardos controlados.

## análisis de contenido

En la Semana 6 se trabajó el uso de funciones recursivas como una alternativa al uso de ciclos tradicionales, aplicándolas a problemas clásicos. Se utilizó la recursión para resolver cálculos matemáticos y recorridos estructurados, reforzando el entendimiento del caso base y el caso recursivo como elementos clave para evitar ejecuciones infinitas.

También se implementaron estructuras jerárquicas simples, donde se representaron nodos con referencias a otros nodos, permitiendo simular relaciones padre-hijo. A través de estas estructuras se realizaron recorridos recursivos, observando cómo la llamada a funciones anidadas permite visitar cada elemento de manera ordenada.

El contenido de la semana se enfocó en comprender cómo la recursión permite dividir un problema en subproblemas más pequeños, así como en reconocer sus ventajas y limitaciones frente a soluciones iterativas.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

# Semana 7 — Estructuras de datos: HashTables, Set y Stack

**Resumen general**

Semana 7 se centra en el estudio y la implementación de **estructuras de datos especializadas** que controlan cómo se almacenan, acceden y recorren los datos. Se trabajan tablas hash, conjuntos (Set) y pilas (Stack), combinando lógica en JavaScript con ejemplos ejecutables desde HTML.

## Lenguajes utilizados

* JavaScript
* HTML

**Estructura de la carpeta**

* index.html
* code.js
* hashTables/
  + hashTables.js
* set/
  + index.html
  + main.js
* stack/
  + index.html
  + main.js

## Objetivos de aprendizaje

* Comprender cómo se modelan estructuras de datos más allá de arreglos y objetos simples.
* Implementar una tabla hash utilizando una función de dispersión.
* Utilizar Set para manejar colecciones sin elementos duplicados.
* Implementar una pila (Stack) y aplicar el principio LIFO (Last In, First Out).

**Descripción técnica y funciones habituales**

**Archivos principales**

**code.js**

Se trabajan estructuras de datos básicas como punto de partida:

* Variables de distintos tipos (string, number, boolean) y su visualización en el DOM.
* Arreglos dinámicos, donde se agregan y recorren elementos ingresados por el usuario.
* Objetos que agrupan múltiples atributos relacionados.
* Arreglos de objetos, donde cada elemento representa una estructura completa con varios campos (nombre, edad, teléfono, estatura, etc.).

Estas funciones permiten observar cómo los datos simples evolucionan hacia estructuras más complejas.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**Carpeta hashTables**

**hashTables.js**

Se implementa una **tabla hash desde cero**:

* Se define una clase hashTable que inicializa un arreglo de tamaño fijo.
* hashMethod(key): transforma una llave tipo texto en una posición del arreglo.
* set(key, value): inserta pares clave–valor, manejando colisiones mediante buckets.
* get(key): recupera el valor asociado a una llave.
* delete(key): elimina un elemento de la tabla.
* showKeys(): recorre la estructura y muestra las llaves almacenadas.

El ejemplo permite observar cómo una tabla hash optimiza búsquedas frente a un arreglo tradicional.

**Carpeta set**

**main.js**

Se trabaja con la estructura **Set**:

* add(value): inserta elementos sin permitir duplicados.
* delete(value): elimina elementos del conjunto.
* has(value): verifica la existencia de un elemento.
* Recorridos directos del conjunto mediante for...of.

Además, se incluye un ejemplo práctico donde se almacenan **funciones** dentro de un Set para garantizar que cada acción se ejecute una sola vez.

**Carpeta stack**

**main.js**

Se implementa una **pila personalizada**:

* Clase Stack con un arreglo interno privado.
* push(item): agrega un elemento a la pila.
* pop(): elimina el último elemento agregado.
* peek(): consulta el elemento superior sin eliminarlo.
* size() e isEmpty(): permiten inspeccionar el estado de la pila.

La pila se integra con HTML para agregar y eliminar nombres, mostrando visualmente cómo funciona el acceso LIFO.

## Análisis del contenido

En esta semana se refuerza cómo distintas estructuras controlan el acceso a los datos según reglas específicas. La tabla hash prioriza búsquedas rápidas, el Set garantiza unicidad y la pila restringe el acceso al último elemento. Cada implementación permite observar claramente el comportamiento interno de la estructura y su aplicación en escenarios prácticos.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

# Semana 8 — Listas enlazadas: estructuras simples y dobles

**Resumen general**

Semana 8 se centra en el análisis y la construcción de **listas enlazadas**, abordando tanto listas simplemente enlazadas como listas doblemente enlazadas. El trabajo muestra la estructura interna de los nodos, la forma en que se conectan entre sí y las operaciones básicas para recorrer, insertar, eliminar y buscar elementos.

## Lenguajes utilizados

* JavaScript

**Estructura de la carpeta**

* Código con ejemplos de listas simplemente enlazadas definidas de forma manual.
* Implementaciones con clases para listas doblemente enlazadas.
* Ejemplos de uso donde se recorren y manipulan los nodos desde la consola.

## Objetivos

* Comprender cómo se representa una lista enlazada a nivel de objetos.
* Diferenciar entre listas simplemente enlazadas y doblemente enlazadas.
* Implementar operaciones básicas sobre listas enlazadas sin utilizar arreglos.

## Descripción de contenido por archivo

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Ejemplo de lista simplemente enlazada (estructura manual)**

Se define una lista simplemente enlazada utilizando objetos anidados. Cada nodo contiene un valor y una referencia al siguiente nodo mediante la propiedad next.

* La lista inicia con una propiedad head.
* Cada nodo apunta al siguiente hasta llegar a null, que indica el final de la lista.
* Se incluye una función de recorrido que imprime los valores de los nodos en orden.

Este ejemplo permite visualizar claramente cómo se conectan los nodos sin necesidad de clases.

**Ejemplo de lista doblemente enlazada (estructura básica)**

Se presenta una estructura similar a la lista simple, pero preparada para evolucionar hacia una lista doblemente enlazada.

* Los nodos conservan la referencia next.
* Se deja la base para agregar referencias hacia el nodo anterior.

Este fragmento sirve como transición conceptual entre listas simples y listas dobles.

**Clase Nodes**

La clase Nodes define la estructura de cada nodo dentro de la lista:

* value: almacena el dato del nodo.
* next: referencia al siguiente nodo.
* prev: referencia al nodo anterior (en listas doblemente enlazadas).

Esta clase estandariza la creación de nodos y facilita la manipulación de la lista.

**Clase myDoublyLinkedList**

Se implementa una lista doblemente enlazada utilizando una clase personalizada.

* El constructor inicializa la lista con un nodo cabeza (head).
* Se mantiene una referencia a la cola (tail) para inserciones eficientes.
* Se lleva un control del tamaño mediante la propiedad length.

Funciones implementadas:

* add(value): agrega un nuevo nodo al final de la lista y actualiza las referencias next y prev.
* insert(index, value): inserta un nodo en una posición específica ajustando los enlaces entre nodos.
* getTheIndex(index): recorre la lista hasta encontrar el nodo en la posición indicada.
* remove(index): elimina un nodo actualizando las referencias del nodo anterior y siguiente.
* search(): recorre la lista desde la cabeza e imprime los valores de cada nodo.

**Ejemplo de uso**

Se crea una instancia de la lista doblemente enlazada y se agregan varios nodos de forma secuencial.

* Se añaden nodos con valores descriptivos.
* Se ejecuta el método de recorrido para verificar el orden y las conexiones.

Este ejemplo permite comprobar el funcionamiento interno de la estructura y observar cómo crece la lista dinámicamente.

## Análisis del contenido

En esta semana se observa cómo las listas enlazadas permiten almacenar datos de manera dinámica sin depender de índices fijos. La diferencia entre referencias simples y dobles facilita operaciones como inserciones intermedias y eliminaciones sin recorrer toda la estructura desde el inicio.

El enfoque práctico permite entender cómo se gestionan manualmente las conexiones entre nodos, reforzando la comprensión del uso de memoria y del recorrido secuencial en estructuras no indexadas.

**Qué se refuerza**

* Representación interna de listas enlazadas.
* Manejo explícito de referencias entre nodos.
* Ventajas de las listas doblemente enlazadas frente a las simples en operaciones de inserción y eliminación.

# Semana 9 — Implementación y análisis de estructuras de datos fundamentales

**Resumen general**

Semana 9 aborda la implementación detallada de las principales estructuras de datos utilizando JavaScript. Se trabaja cada estructura desde cero mediante clases y nodos, permitiendo analizar su comportamiento interno, sus operaciones básicas y las diferencias entre modelos lineales y no lineales.

## Lenguajes utilizados

* JavaScript

**Estructura de la carpeta**

* arrays.js — Implementación manual de un arreglo usando objetos.
* queue.js / queues.js — Colas (Queue) basadas en nodos enlazados.
* stack.js — Implementación de pila (Stack).
* singly\_linkedlist.js — Lista simplemente enlazada.
* doubly\_linkedlist.js — Lista doblemente enlazada.
* tree.js — Árbol binario de búsqueda (Binary Search Tree).
* graph.js — Grafo no dirigido con lista de adyacencia.
* hash.table.js — Tabla hash con función hash y manejo de colisiones.
* LICENCE — Archivo de licencia del proyecto.

## Objetivos

* Analizar cómo se construyen internamente las estructuras de datos.
* Identificar ventajas y limitaciones de cada estructura.
* Comparar métodos de inserción, eliminación y búsqueda.
* Reconocer patrones comunes como nodos, referencias y recorridos.

## Descripción de contenido por archivo

**Arreglos (arrays.js)**

* Se implementa una estructura similar a un arreglo utilizando un objeto.
* Propiedades:
  + length
  + data
* Operaciones implementadas:
  + push(item)
  + pop()
  + get(index)
  + delete(index)
  + shift()
  + unshift(item)
* Se observa el manejo manual de índices y el corrimiento de elementos. Texto

  El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**Colas / Queue (queue.js, queues.js)**

* Implementación basada en nodos enlazados.
* Modelo FIFO (First In, First Out).
* Propiedades:
  + first / top
  + last / bottom
  + length
* Métodos:
  + enqueue(value)
  + dequeue()
  + peek()
* Se analiza cómo los elementos se agregan al final y se eliminan desde el inicio.

**Pilas / Stack (stack.js)**

* Implementación de tipo LIFO (Last In, First Out).
* Uso de nodos enlazados.
* Propiedades:
  + top
  + bottom
  + length
* Métodos:
  + push(value)
  + pop()
  + peek()
* Se observa cómo el acceso siempre ocurre desde la parte superior.

**Lista simplemente enlazada (singly\_linkedlist.js)**

* Cada nodo contiene:
  + value
  + next
* Propiedades principales:
  + head
  + tail
  + length
* Operaciones:
  + append(value)
  + prepend(value)
  + insert(index, value)
  + remove(index)
* Se analiza el recorrido secuencial desde la cabeza hasta el nodo deseado.

**Lista doblemente enlazada (doubly\_linkedlist.js)** **Texto

El contenido generado por IA puede ser incorrecto.** Texto

El contenido generado por IA puede ser incorrecto.**Texto

El contenido generado por IA puede ser incorrecto.**

Cada nodo contiene:

* + value
  + next
  + prev
* Permite recorrido hacia adelante y hacia atrás.
* Operaciones:
  + append(value)
  + prepend(value)
  + insert(index, value)
  + remove(index)
* Se observa la diferencia frente a la lista simple en operaciones intermedias.

**Árbol binario de búsqueda (tree.js)** Texto

El contenido generado por IA puede ser incorrecto.

**Texto

El contenido generado por IA puede ser incorrecto.**

* Estructura jerárquica basada en nodos.
* Cada nodo contiene:
  + value
  + left
  + right
* Propiedad principal:
  + root
* Operaciones:
  + insert(value)
  + search(value)
* Se analiza cómo los valores se organizan manteniendo el orden del BST.

**Grafo (graph.js)**

**Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.**

* Representación mediante lista de adyacencia.
* Propiedades:
  + adjacentList
  + nodos
* Operaciones:
  + addVertices(node)
  + addEdge(node1, node2)
* Se observa la representación de relaciones no jerárquicas entre nodos.

**Tabla hash (hash.table.js**

**Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.**

* Uso de un arreglo como base de almacenamiento.
* Función hash personalizada basada en caracteres.
* Manejo de colisiones mediante encadenamiento.
* Operaciones:
  + set(key, value)
  + get(key)
  + delete(key)
  + getAllKey()
* Se analiza cómo múltiples claves pueden compartir una misma posición.

## Análisis de contenido

El contenido de esta semana está orientado a comprender cómo funcionan internamente las estructuras de datos, evitando el uso de implementaciones nativas de JavaScript. Cada archivo aborda una estructura distinta, mostrando explícitamente el manejo de nodos, referencias, índices y memoria lógica.

Se observa una progresión desde estructuras lineales simples (arreglos, pilas y colas), hacia estructuras más complejas como listas enlazadas, árboles binarios y grafos. Esto permite analizar diferencias en costos operativos, flexibilidad y casos de uso. Además, la implementación de tablas hash introduce el concepto de dispersión, colisiones y encadenamiento.

# Semana 10 — Integración de estructuras de datos y consumo de APIs

**Resumen general**

Semana 10 integra estructuras de datos clásicas con aplicaciones más completas en JavaScript. Se combinan pilas, árboles binarios, conjuntos, diccionarios y listas enlazadas con ejemplos prácticos, además de introducir el consumo de una API pública para mostrar datos dinámicos en una interfaz web.

## Lenguajes utilizados

* JavaScript
* HTML
* CSS

**Estructura de la carpeta**

* Archivos JavaScript con implementaciones de estructuras de datos:
  + Stack
  + Binary Search Tree
  + Set y operaciones de conjuntos
  + Linked List
  + Dictionary / HashTable
* Archivos HTML y CSS para consumo de API y renderizado de datos.
* Scripts para consumo de la **Rick and Morty API**.

## análisis de contenido

En esta semana se observa un **salto de estructuras aisladas a escenarios más integrados**. Por un lado, se refuerza la implementación manual de estructuras de datos (pilas, árboles, listas, conjuntos y tablas hash). Por otro, se introduce el consumo de datos externos mediante fetch, mostrando cómo los datos obtenidos pueden ser transformados y representados visualmente.

El contenido permite analizar cómo las estructuras de datos no solo funcionan a nivel lógico, sino cómo pueden apoyar aplicaciones reales que interactúan con servicios externos y el DOM.

## Objetivos

* Reforzar la implementación manual de estructuras de datos.
* Aplicar árboles binarios y pilas en escenarios concretos.
* Comprender operaciones de conjuntos (unión, intersección y diferencia).
* Consumir una API REST y renderizar datos dinámicamente.
* Integrar lógica de datos con interfaces web.

## Descripción de contenido

**Pila (Stack)**

* Implementación mediante nodos enlazados.
* Estructura LIFO (Last In, First Out).
* Propiedades:
  + top
  + bottom
  + length
* Métodos:
  + push(value): inserta un elemento en la parte superior.
  + pop(): elimina el elemento superior.
  + peek(): devuelve el nodo superior sin eliminarlo.
* Se observa el comportamiento de inserción y eliminación secuencial.

**Árbol binario de búsqueda (Binary Search Tree)**

* Estructura jerárquica basada en nodos con:
  + left
  + right
  + value
* Métodos:
  + insert(value): inserta nodos respetando el orden del BST.
  + search(value): búsqueda recursiva mostrando el recorrido (izquierda o derecha).
* Se visualiza cómo el árbol se va formando según el valor insertado.

**Recorridos de árboles binarios**

* Implementación de recorridos:
  + In-order
  + Pre-order
  + Post-order
* Uso de funciones de visita para imprimir valores.
* Permite analizar cómo cambia el orden de salida según el recorrido utilizado.

**Conjuntos (Set)**

* Uso del objeto Set de JavaScript.
* Operaciones implementadas manualmente:
  + Unión (A ∪ B)
  + Intersección (A ∩ B)
  + Diferencia (A − B)
* Se analiza la eliminación de duplicados y la comparación por referencia.

**Diccionarios y Tablas Hash**

* Implementación de diccionario usando objetos.
* Métodos:
  + set(key, value)
  + get(key)
  + delete(key)
  + keys()
  + values()
* Implementación alternativa de HashTable con:
  + Función hash
  + Resolución de colisiones
* Se observa el acceso eficiente a datos mediante claves.

**Listas enlazadas**

* Implementación de lista simplemente enlazada.
* Métodos:
  + push
  + pop
  + get(index)
  + isEmpty
* Se analiza el recorrido secuencial y el costo de acceso por índice.

**Consumo de API (Rick and Morty API)**

* Uso de fetch y async/await.
* Obtención de personajes desde una API REST.
* Procesamiento del JSON recibido.
* Renderizado dinámico de tarjetas en el DOM:
  + Imagen
  + Nombre
  + Estado
  + Especie
  + Género
* Manejo de errores y estados de carga.
* Separación clara entre lógica de datos y presentación.

**Ejecución**

1. Ejecutar los scripts de estructuras de datos en consola.
2. Abrir los archivos HTML en el navegador.
3. Verificar la carga dinámica de datos desde la API.
4. Probar inserciones, búsquedas y eliminaciones en las estructuras.
5. Analizar el comportamiento del árbol y de la pila según los datos insertados.

**Qué se observa**

* Integración entre estructuras de datos y aplicaciones reales.
* Uso combinado de lógica algorítmica y manipulación del DOM.
* Consumo de servicios externos como fuente de datos.
* Refuerzo del uso de estructuras como base para aplicaciones web dinámicas.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

# Semana 11 — Proyecto Ecommerce: carrito de compras, persistencia y render dinámico

**Resumen general**

Semana 11 desarrolla un proyecto tipo **ecommerce** donde se implementa un catálogo de productos, un carrito de compras interactivo y persistencia de datos en el navegador. El enfoque está en la manipulación del DOM, el manejo de eventos y el uso de localStorage para conservar el estado del carrito.

## Lenguajes utilizados

* JavaScript
* HTML
* CSS

**Estructura de la carpeta**

* index.html: estructura principal de la aplicación.
* app.js: lógica del carrito, renderizado y eventos.
* products.json: fuente de datos de los productos.
* style.css: estilos del catálogo y carrito.
* assets/: imágenes de los productos.

## análisis de contenido

Se construye una aplicación funcional donde los datos se cargan dinámicamente desde un archivo JSON, se renderizan en pantalla y se manipulan mediante eventos. El estado del carrito se mantiene sincronizado entre memoria, DOM y almacenamiento local, reforzando conceptos de interacción usuario–aplicación.

## Descripción por archivo

**index.html**

**Texto

El contenido generado por IA puede ser incorrecto.**

* Se define la estructura base del ecommerce.
* Se incluye un **header** con ícono de carrito y contador de productos.
* Se crea el contenedor .listProduct donde se insertan los productos desde JavaScript.
* Se define el panel lateral .cartTab que representa el carrito de compras.
* Se enlazan los estilos y el archivo app.js.

**app.js**

**Texto

El contenido generado por IA puede ser incorrecto.**

* Se seleccionan elementos del DOM relacionados con:
  + Carrito (.icon-cart, .cartTab, .listCart).
  + Lista de productos (.listProduct).
  + Contadores visuales.
* Se implementa la lógica para:
  + Mostrar y ocultar el carrito mediante clases CSS.
  + Cargar productos desde products.json usando fetch.
  + Renderizar productos dinámicamente en el HTML.
* Se define el método addToCart:
  + Agrega productos al carrito.
  + Incrementa la cantidad si el producto ya existe.
* Se gestiona el carrito mediante:
  + Un arreglo carts en memoria.
  + Persistencia con localStorage.
* Se renderiza el contenido del carrito:
  + Imagen, nombre, precio total y cantidad.
  + Actualización del contador del ícono del carrito.
* Se inicializa la aplicación con initApp, cargando productos y restaurando el carrito almacenado.

**products.json**

* Se define un arreglo de objetos que representan los productos.
* Cada producto incluye:
  + id
  + name
  + price
  + image
* Las imágenes referencian archivos dentro de la carpeta assets/.
* Este archivo actúa como una simulación de una fuente de datos externa.

**style.css**

* Se definen estilos generales y tipografías externas.
* Se maqueta el catálogo usando **CSS Grid**.
* Se estiliza el carrito lateral con animaciones de entrada y salida.
* Se diseñan botones interactivos con efectos hover y active.
* Se controlan transiciones visuales al mostrar el carrito (body.showCart).
* Se ajusta el diseño del listado del carrito y sus controles de cantidad.

**assets/**

* Contiene las imágenes de los productos.
* Cada archivo corresponde a un producto definido en products.json.
* Se utilizan para renderizar visualmente el catálogo y el carrito.

**Objetivos de aprendizaje**

* Consumir datos desde archivos JSON.
* Manipular el DOM de forma dinámica.
* Implementar un carrito de compras funcional.
* Persistir información usando localStorage.
* Coordinar estado entre datos, interfaz y almacenamiento. Interfaz de usuario gráfica, Aplicación

  El contenido generado por IA puede ser incorrecto.

# Semana 12 — Pokédex: consumo de API y filtrado dinámico

Resumen general

Semana 12 consiste en el desarrollo de una **Pokédex interactiva**, que consume datos desde una API pública y muestra la información de manera dinámica. Se implementa un sistema de filtrado por tipo de Pokémon, renderizado dinámico de tarjetas y diseño responsivo.

## Lenguajes utilizados

* JavaScript
* HTML
* CSS

**Estructura de la carpeta**

* index.html: estructura principal de la aplicación.
* .vscode/: configuración del entorno de desarrollo.
* css/: estilos visuales de la Pokédex.
* js/: lógica de consumo de API y renderizado.
* img/: recursos gráficos (logo).

## análisis de contenido

En esta semana se integra el consumo masivo de datos desde una API externa, el procesamiento de respuestas JSON y la manipulación del DOM para generar contenido dinámico. Se refuerza el uso de fetch, manejo de arreglos, funciones de transformación y filtrado por eventos, además de diseño adaptable a distintos tamaños de pantalla.

## Descripción por archivo

**index.html**

* Define la estructura base de la Pokédex.
* Incluye un encabezado con navegación y botones para filtrar Pokémon por tipo.
* Contiene el contenedor principal donde se insertan dinámicamente las tarjetas de Pokémon.
* Enlaza el archivo de estilos y el script principal.
* Usa identificadores (id y class) para permitir la manipulación desde JavaScript. Texto

  El contenido generado por IA puede ser incorrecto.Texto

  El contenido generado por IA puede ser incorrecto.

**.vscode/settings.json**

* Configura el puerto del servidor local (Live Server).
* Facilita la ejecución del proyecto en el navegador durante el desarrollo.

**css/main.css**

* Define variables CSS para colores asociados a cada tipo de Pokémon.
* Aplica estilos globales y reseteo básico.
* Maqueta la lista de Pokémon usando CSS Grid.
* Implementa diseño responsivo con media queries (1, 2 o 3 columnas según pantalla).
* Estiliza tarjetas, botones y estadísticas.
* Aplica colores dinámicos a botones y tipos de Pokémon según su categoría. Texto

  El contenido generado por IA puede ser incorrecto.

**js/main.js**

**Texto

El contenido generado por IA puede ser incorrecto.**

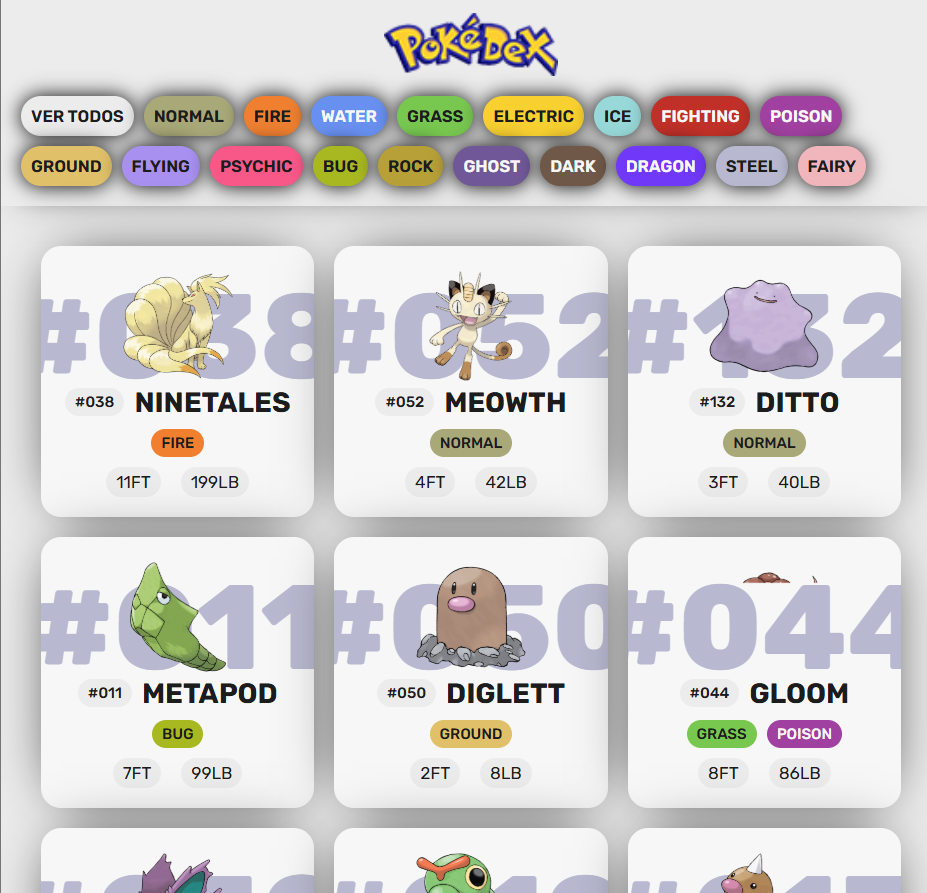
* Selecciona el contenedor principal donde se renderizan los Pokémon.
* Define la URL base de la PokéAPI.
* Realiza múltiples peticiones fetch para obtener información de cada Pokémon.
* Genera dinámicamente las tarjetas con:
  + Imagen oficial.
  + ID formateado.
  + Nombre.
  + Tipos.
  + Altura y peso.
* Implementa filtrado por tipo usando botones del encabezado.
* Limpia y vuelve a renderizar la lista según el filtro seleccionado.
* Utiliza métodos como map, join, some y manipulación directa del DOM.

**img/**

* Contiene el logo utilizado en la navegación de la Pokédex.
* Se emplea como recurso visual estático.

**Objetivos de aprendizaje**

* Consumir una API pública desde JavaScript.
* Procesar y mostrar grandes volúmenes de datos.
* Filtrar información dinámicamente mediante eventos.
* Manipular el DOM para generar interfaces dinámicas.
* Aplicar diseño responsivo con CSS Grid.



# Semana 13 — Recursión, ordenamiento y sistema web interactivo (EduTrack)

**Resumen general**

Semana 13 consolida el uso de **algoritmos clásicos** (recursión y ordenamiento) y los integra con el desarrollo de un **sistema web funcional completo**. El trabajo combina análisis algorítmico con una aplicación real que gestiona estudiantes, clases, asistencia, participación y reportes dinámicos.

## Lenguajes utilizados

* JavaScript
* HTML
* CSS (Tailwind CSS)

**Estructura de la carpeta**

* recursion/: funciones recursivas y variantes optimizadas.
* sorting/: algoritmos de ordenamiento clásicos.
* okcomputer\_design.html: interfaz gráfica completa del sistema.
* main.js: lógica, estado, renderizado y reportes del sistema EduTrack.

## análisis de contenido

El contenido de esta semana conecta la teoría algorítmica con una aplicación real. La recursión se analiza desde su impacto en complejidad y memoria, los algoritmos de ordenamiento se comparan en rendimiento, y todo esto se complementa con un sistema web que aplica estructuras de datos, programación orientada a objetos, manejo del DOM y persistencia local.

## Descripción de contenido

**Carpeta recursion/**

**Archivos y funciones**

* printNRecursive(n)
  + Imprime valores descendentes usando llamadas recursivas.
  + Introduce el concepto de **caso base**.
* countDownToZero(n)
  + Cuenta regresivamente hasta cero.
  + Muestra control explícito del fin de la recursión.
* getNthFibo(n) (iterativo)
  + Calcula Fibonacci con ciclo.
  + Optimiza uso de memoria frente a la versión recursiva.
* getNthFibo(n) (recursivo)
  + Implementa Fibonacci de forma directa.
  + Permite analizar complejidad exponencial.
* getNthFiboBetter(n, lastlast, last)
  + Versión recursiva optimizada.
  + Reduce la complejidad a O(n) usando acumuladores.

**Objetivo**

Comprender cómo la recursión afecta tiempo y espacio, y cuándo debe optimizarse o reemplazarse.

**Carpeta sorting/**

**Algoritmos implementados**

* **Merge Sort**
  + Divide el arreglo recursivamente y fusiona resultados.
  + Uso de memoria auxiliar.
  + Complejidad O(n log n).
* **Quick Sort**
  + Partición con pivote central.
  + Ordenamiento in-place.
  + Comparación de casos promedio y peor caso.
* **Selection Sort**
  + Búsqueda del mínimo por iteración.
  + Algoritmo simple con complejidad O(n²).

**Objetivo**

Comparar algoritmos por eficiencia, uso de memoria y facilidad de implementación.

**okcomputer\_design.html**

**Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.**

**Descripción**

Archivo que define la **interfaz completa del sistema EduTrack**, incluyendo:

* Dashboard con métricas generales.
* Registro de asistencia.
* Registro de participación.
* Listas dinámicas de clases y estudiantes.
* Modales para captura y reportes.
* Diseño responsivo con Tailwind CSS.
* Animaciones visuales con Anime.js.

**Función principal**

Servir como estructura visual y punto de interacción para toda la lógica implementada en main.js.

**main.js**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Captura de pantalla con letras y números

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Descripción técnica completa**

Este archivo implementa **todo el comportamiento del sistema EduTrack**, usando una arquitectura basada en clases.

**Gestión de datos**

* Arreglos para:
  + estudiantes
  + clases
  + asistencias
  + participaciones
  + sesiones
* Persistencia mediante localStorage.
* Sincronización automática entre pestañas.

**Inicialización**

* Carga de datos de ejemplo.
* Restauración de datos guardados.
* Inicialización de eventos y renderizado inicial.

**Renderizado dinámico**

* renderClassCards(): muestra tarjetas de clases con conteo de estudiantes.
* renderStudentList(): lista interactiva de estudiantes.
* Actualización automática del dashboard.

**Interacción con la UI**

* Delegación de eventos para:
  + seleccionar clases
  + seleccionar estudiantes
* Animaciones visuales de selección.
* Sincronización de selects con clics en tarjetas/listas.

**Asistencia**

* Apertura y cierre de modal.
* Registro individual o masivo (presentes / ausentes).
* Guardado con fecha y hora.
* Actualización inmediata de métricas.

**Participación**

* Registro por tipo y puntaje.
* Asociación con estudiante y clase.
* Cálculo automático de promedios.

**Dashboard**

* Total de estudiantes.
* Porcentaje de asistencia del día.
* Promedio general de participación.
* Número de clases activas.

**Actividad reciente**

* Combina asistencias y participaciones.
* Ordena por fecha y hora.
* Renderiza eventos recientes con etiquetas visuales.

**Cálculo de estadísticas**

* calculateAttendanceStats()
* calculateParticipationStats()
* calculateStudentAnalytics()
* calculateClassOverview()

Permiten obtener métricas detalladas por estudiante y por clase.

**Reportes**

* Renderizado de:
  + reporte de asistencia
  + reporte de participación
  + ranking general
  + resumen por clase
* Uso de tablas y tarjetas dinámicas.

**Utilidades**

* Exportación de datos a JSON.
* Sistema de notificaciones flotantes.
* Animaciones de elementos y modales.
* Efectos visuales al cargar la página.

**Objetivos de aprendizaje**

* Aplicar recursión de forma correcta y eficiente.
* Implementar y analizar algoritmos de ordenamiento.
* Construir aplicaciones web con lógica compleja.
* Gestionar estado y persistencia en frontend.
* Integrar algoritmos con sistemas reales.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# Semana 14 — Objetos y aplicación CRUD con almacenamiento local

**Resumen general**

La semana 14 se centra en el uso de **objetos en JavaScript** y en la construcción de una **aplicación CRUD completa** (Crear, Leer, Actualizar y Eliminar) tipo **ToDo**, utilizando **localStorage** para persistencia de datos y una interfaz web funcional.

## Lenguajes utilizados

* JavaScript
* HTML
* CSS

**Estructura de la carpeta**

* .vscode/: configuración del entorno de desarrollo.
* demo/: ejemplos básicos de objetos en JavaScript.
* app.js: lógica principal de la aplicación ToDo (CRUD).
* index.html: estructura de la interfaz de usuario.
* styles.css: estilos y diseño responsivo de la aplicación.

## Análisis de contenido

En esta semana se aplica JavaScript para manejar datos estructurados mediante objetos y arreglos, integrándolos con el DOM. El proyecto principal demuestra cómo crear una aplicación funcional de gestión de tareas, persistente entre recargas del navegador, reforzando conceptos de estado, eventos y manipulación dinámica de la interfaz.

## Descripción de contenido

**Carpeta .vscode**

**Descripción**

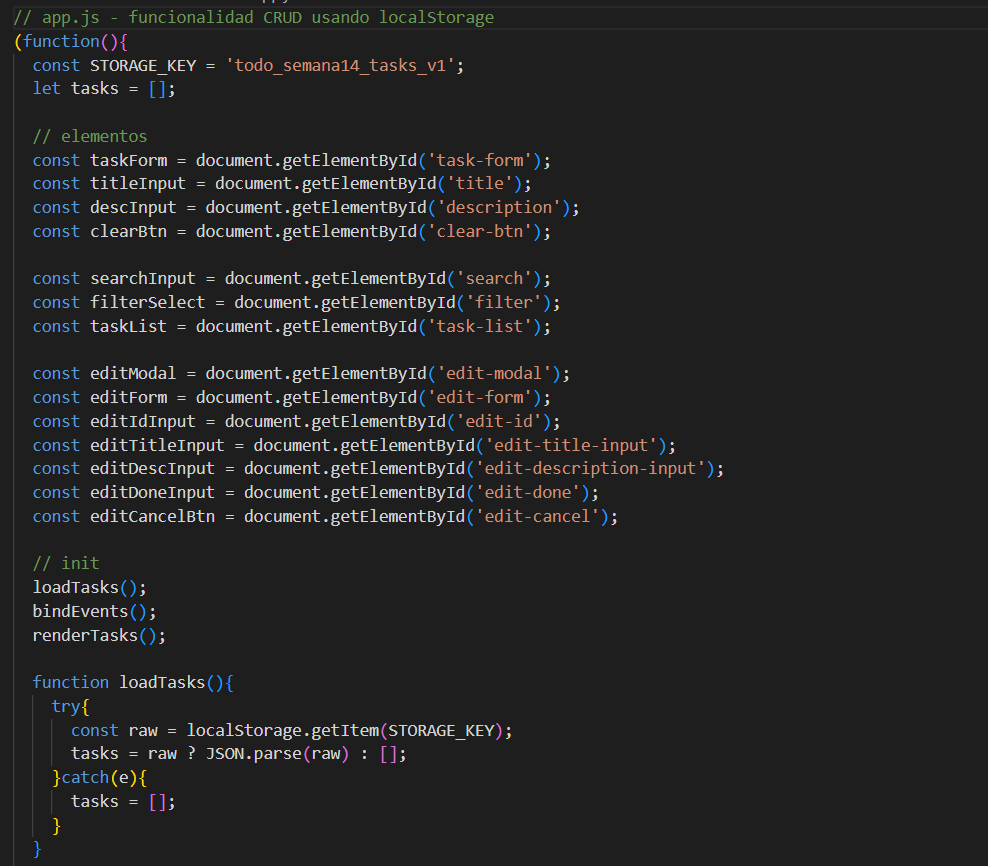
Configura el puerto del servidor local para ejecutar la aplicación de forma consistente durante el desarrollo.

**Carpeta demo**

**Descripción**

Ejemplo simple de un **objeto en JavaScript**, mostrando cómo agrupar propiedades relacionadas dentro de una sola estructura de datos. Sirve como introducción conceptual antes de aplicar objetos de forma más compleja en la aplicación principal.

**app.js**

****

**Descripción general**

Archivo principal que implementa la **funcionalidad CRUD completa** de la aplicación ToDo, encapsulada en una función autoejecutable para evitar contaminación del ámbito global.

**Funcionalidades implementadas**

* **Creación de tareas**
  + Registro de título y descripción.
  + Asignación de identificador único y fecha de creación.
* **Lectura y visualización**
  + Renderizado dinámico de la lista de tareas.
  + Mensaje informativo cuando no hay tareas.
* **Actualización**
  + Edición de título, descripción y estado (completada / pendiente).
  + Uso de un modal para edición.
* **Eliminación**
  + Eliminación de tareas con confirmación previa.

**Persistencia**

* Uso de localStorage para guardar tareas.
* Recuperación automática al cargar la página.
* Manejo de errores al leer datos almacenados.

**Filtros y búsqueda**

* Búsqueda por texto en título o descripción.
* Filtros por estado:
  + todas
  + pendientes
  + completadas

**Interacción con la UI**

* Manejo de eventos de formularios.
* Botones dinámicos para editar, completar y eliminar.
* Atajo de teclado (Escape) para cerrar el modal de edición.

**index.html**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Descripción**

Archivo que define la **estructura visual** de la aplicación ToDo.

**Secciones principales**

* Encabezado con título y descripción.
* Formulario para registrar tareas.
* Controles de búsqueda y filtrado.
* Lista dinámica de tareas.
* Modal para edición de tareas.

**Características**

* Uso de etiquetas semánticas.
* Atributos de accesibilidad (aria-live, role="dialog").
* Separación clara entre estructura (HTML), estilo (CSS) y lógica (JS).

**styles.css**

**Descripción**

Archivo de estilos que define la apariencia visual de la aplicación.

**Elementos destacados**

* Variables CSS para colores y estilos globales.
* Diseño limpio tipo panel/card.
* Botones diferenciados para acciones normales y destructivas.
* Modal centrado con fondo translúcido.
* Diseño responsivo con media queries para pantallas grandes.

## Objetivos

* Comprender y utilizar objetos en JavaScript.
* Implementar operaciones CRUD completas.
* Manipular el DOM de forma dinámica.
* Persistir datos en el navegador usando localStorage.
* Diseñar interfaces web funcionales y responsivas.

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

# Semana 15 — Estructuras de datos avanzadas en JavaScript

## Lenguajes utilizados

* JavaScript

## Objetivo

Implementar y analizar estructuras de datos avanzadas mediante ejercicios prácticos en JavaScript, reforzando el manejo de memoria, referencias, relaciones entre datos y diseño de algoritmos básicos.

**Estructura de la carpeta**

* Archivo JavaScript principal con ejercicios de lógica y fundamentos.
* Carpeta graph.
* Carpeta hashtable.
* Carpeta linkedlist.

## Descripción de los archivos

**Archivo JavaScript principal**

En este archivo se trabajaron ejercicios orientados al refuerzo de fundamentos del lenguaje que son necesarios para comprender estructuras complejas:

* Se utilizó el **manejo de variables globales y locales** para analizar el alcance (scope) y su impacto en la ejecución de funciones.
* Se implementaron **condicionales if / else y switch** para evaluar múltiples escenarios de decisión.
* Se empleó el **operador ternario** para simplificar validaciones lógicas.
* Se desarrollaron **funciones tradicionales** y **funciones con retorno**, comparando su comportamiento.
* Se utilizaron **funciones constructoras** con la palabra clave this para crear objetos dinámicamente.
* Se trabajó con **clases** y métodos para organizar operaciones matemáticas, reforzando la programación orientada a objetos.
* Se aplicó el uso de **arrow functions** dentro de constructores para manejar correctamente el contexto de this.

Este archivo funciona como base conceptual para entender cómo se construyen las estructuras de datos en las carpetas posteriores.

**Carpeta graph**

**Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.**

En esta carpeta se abordó el concepto de grafos desde distintas perspectivas prácticas:

* Se utilizó la **representación por lista de aristas** para modelar conexiones directas entre nodos.
* Se trabajó la **lista de adyacencia** usando arreglos y objetos para representar relaciones bidireccionales.
* Se implementó la **matriz de adyacencia** para visualizar conexiones mediante valores binarios.
* Se desarrolló una **clase Graph** para encapsular la lógica del grafo.
* Se utilizaron métodos para:
  + Agregar vértices dinámicamente.
  + Conectar nodos mediante aristas bidireccionales.
* Se reforzó el uso de **objetos como estructuras de almacenamiento** para manejar nodos y conexiones.

El enfoque fue comprender cómo cambia el comportamiento del grafo dependiendo de su representación interna.

**Carpeta hashtable**

**Texto

El contenido generado por IA puede ser incorrecto.**

En esta carpeta se implementó una tabla hash desde cero:

* Se utilizó un **arreglo de tamaño fijo** como estructura base de almacenamiento.
* Se implementó una **función hash** que convierte claves de texto en índices numéricos.
* Se manejaron **colisiones** mediante el uso de arreglos anidados (encadenamiento).
* Se desarrollaron métodos para:
  + Insertar pares clave–valor.
  + Recuperar valores a partir de una clave.
  + Eliminar elementos de la tabla.
  + Obtener todas las claves almacenadas.
* Se aplicaron recorridos de arreglos y validaciones de existencia para asegurar búsquedas correctas.

Este módulo permitió comprender cómo funcionan internamente los diccionarios y mapas en JavaScript.

**Carpeta linkedlist**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

En esta carpeta se trabajaron dos tipos de listas enlazadas:

**Lista simplemente enlazada**

* Se definió una **clase Node** con referencia al siguiente nodo.
* Se implementó una clase principal para:
  + Agregar elementos al inicio y al final.
  + Insertar nodos en posiciones específicas.
  + Eliminar nodos por índice.
* Se utilizó recorrido secuencial para acceder a nodos específicos.
* Se mantuvo un contador de longitud para controlar la estructura.

**Lista doblemente enlazada**

* Se amplió la estructura del nodo agregando referencias al nodo anterior.
* Se implementaron métodos que:
  + Permiten navegar hacia adelante y hacia atrás.
  + Ajustan correctamente los punteros prev y next.
* Se reforzó el manejo de referencias múltiples en memoria.

El objetivo fue entender la diferencia entre ambas estructuras y el costo de cada operación.

## análisis de contenido

pasó del uso de estructuras nativas a la **construcción manual de estructuras de datos**, permitiendo comprender cómo se almacenan, conectan y manipulan los datos internamente. El trabajo se enfocó más en el razonamiento estructural que en la interfaz, fortaleciendo la lógica algorítmica y el diseño de soluciones escalables.

# Semana 16 — Estructuras lineales y no lineales con integración de interfaz

## Lenguajes utilizados

* JavaScript
* HTML
* CSS

## Objetivo

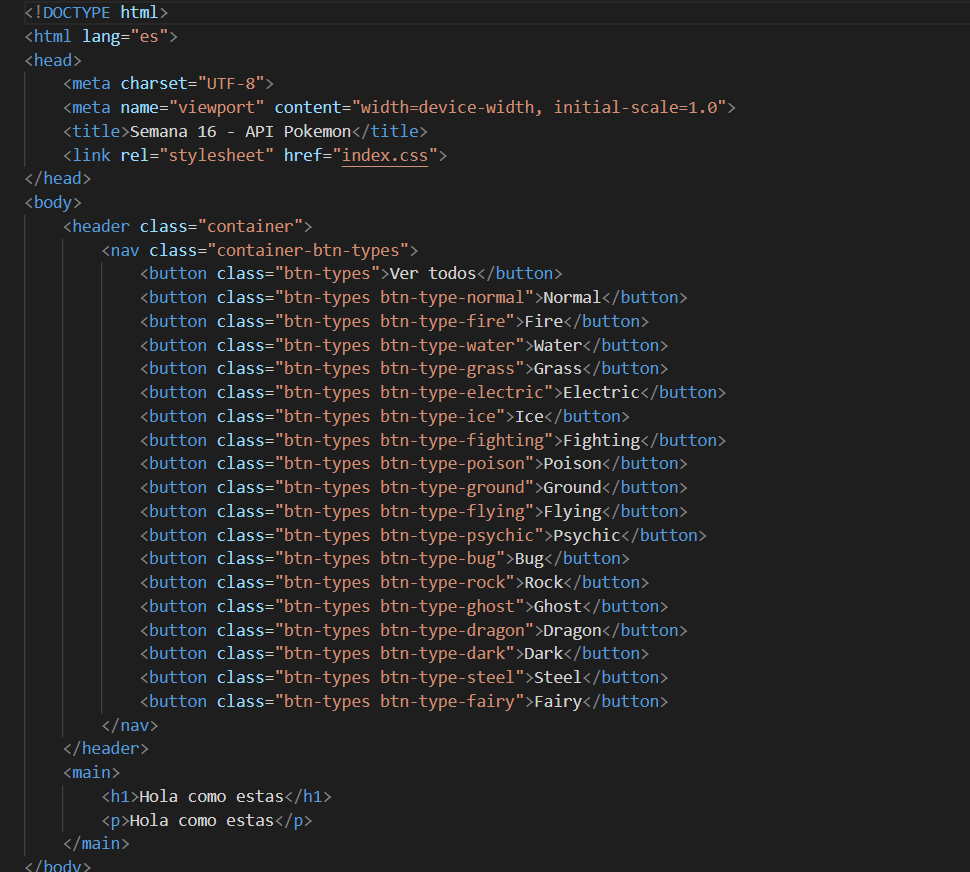
Aplicar estructuras de datos lineales y no lineales (cola, pila y árbol) junto con una interfaz básica en HTML/CSS, integrando lógica de datos, manipulación del DOM y consumo de API como parte de un ejercicio práctico de consolidación.

**Estructura de la carpeta**

* index.html
* index.css
* index.js
* Carpeta queue
* Carpeta stack
* Carpeta tree

## Descripción de los archivos

**index.html**

****

En este archivo se construyó la estructura base de la interfaz:

* Se utilizó **HTML semántico** para organizar encabezado (header), navegación (nav) y contenido principal (main).
* Se implementó un **menú de botones** que representan distintos tipos de Pokémon, pensado para interactuar con JavaScript.
* Se integró la hoja de estilos externa mediante link.
* Se dejó preparado el contenedor principal (main) para renderizar dinámicamente información desde JavaScript.
* Se trabajó una estructura mínima de interfaz, enfocada más en la lógica que en el contenido textual.

Este archivo funciona como punto de entrada visual y de interacción del usuario.

**index.css**

En este archivo se controló toda la presentación visual:

* Se utilizó un **reset básico de estilos** para eliminar márgenes y paddings por defecto.
* Se implementó una **tipografía personalizada desde Google Fonts**.
* Se trabajó con **Flexbox y Grid** para centrar contenido y distribuir elementos de forma responsiva.
* Se aplicaron **clases específicas por tipo de Pokémon**, reutilizadas tanto en botones como en tarjetas.
* Se manejaron colores personalizados para cada tipo, facilitando la identificación visual.
* Se configuró un **layout adaptable** mediante grid-template-columns con auto-fit y minmax.

El CSS se enfocó en la reutilización de clases y la coherencia visual entre botones y contenido dinámico.

**index.js**

****

Este archivo centraliza la lógica de interacción y manipulación del DOM:

* Se utilizó un **DocumentFragment** para mejorar el rendimiento al insertar múltiples elementos.
* Se trabajó con **querySelector y eventos globales** para manejar interacciones del usuario.
* Se definieron funciones asincrónicas para:
  + Realizar peticiones a una API externa.
  + Procesar los datos recibidos.
  + Pintar información dinámicamente en el DOM.
* Se utilizó el evento **DOMContentLoaded** para asegurar la carga correcta del script.
* Se implementó un **manejador global de clicks** para filtrar información según el botón presionado.
* Se separó la lógica de:
  + Petición de datos.
  + Renderizado general.
  + Renderizado por tipo.

Este archivo conecta la interfaz con la lógica de datos y sirve como puente entre estructuras y visualización.

**Carpeta queue**

En esta carpeta se implementó una **estructura de cola (Queue)**:

* Se definió una clase Node para almacenar el valor y la referencia al siguiente nodo.
* Se implementó una clase Queue que mantiene:
  + Referencia al primer elemento.
  + Referencia al último elemento.
  + Longitud total de la estructura.
* Se desarrollaron métodos para:
  + Insertar elementos al final de la cola.
  + Eliminar elementos del inicio.
  + Consultar el primer elemento sin eliminarlo.
* Se manejaron validaciones para evitar operaciones inválidas cuando la cola está vacía.

Esta carpeta refuerza el concepto de **FIFO (First In, First Out)** mediante implementación manual.

**Carpeta stack**

En esta carpeta se trabajó una **estructura de pila (Stack)**:

* Se utilizó una clase Node con referencia al siguiente elemento.
* Se implementó una clase Stack que mantiene:
  + Referencia al elemento superior.
  + Referencia al elemento inferior.
  + Conteo de elementos.
* Se desarrollaron métodos para:
  + Agregar elementos en la parte superior.
  + Remover el elemento superior.
  + Consultar el elemento superior sin modificar la pila.
* Se incluyeron validaciones para evitar extracciones en pilas vacías.

El enfoque fue reforzar el comportamiento **LIFO (Last In, First Out)** y el manejo de referencias.

**Carpeta tree**

En esta carpeta se implementó un **árbol binario de búsqueda**:

* Se definió una clase Node con referencias a:
  + Hijo izquierdo.
  + Hijo derecho.
  + Valor almacenado.
* Se implementó una clase BinarySearchTree que mantiene la raíz del árbol.
* Se desarrollaron métodos para:
  + Insertar valores respetando las reglas del árbol binario.
  + Buscar valores de forma iterativa.
  + Buscar valores de forma recursiva.
* Se manejaron condiciones para árboles vacíos y valores inexistentes.

Esta implementación permitió comprender cómo se organizan los datos jerárquicamente y cómo se optimizan búsquedas.

## análisis de contenido

integra estructuras de datos fundamentales con una interfaz básica, permitiendo observar cómo la lógica interna se conecta con la visualización. Se reforzó el uso de clases, nodos y referencias, así como la interacción con el DOM y eventos. El trabajo se centra en comprender cómo estructuras abstractas pueden coexistir dentro de una aplicación funcional.

## INSIGNIA JAVASCRIPT

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

<https://developers.google.com/profile/badges/recognitions/learnings?hl=es-419>