TECNOLOGICO SUPERIOR DE LA SIERRA NEGRA DE AJALPAN

MAESTRO:JOSE ARTURO BUSTAMENTE LAZCANO
ALUMNO: JUAN CARLOS RAMOS VICARIO

ESTRUCTURA DE DATOS

PORTAFOLIO DE EVIDENCIAS

SEMANA 1

Semana 1 - Fundamentos de JavaScript y Estructuras de Arrays

📋 Información General

Lenguaje Utilizado

JavaScript (ES6+) - Lenguaje de programación interpretado ejecutado en navegadores web para crear ejemplos interactivos.

Objetivo Principal

Proporcionar una introducción fundamental a los conceptos básicos de JavaScript y las estructuras de datos tipo Array, enfocándose en:

- **Sintaxis básica de JavaScript**
- **Integración de JavaScript con HTML**
- **Operaciones fundamentales con arrays**
- **Métodos de transformación de datos (map, filter, reduce)**
- **Introducción a la programación funcional**
- **Concepto de inmutabilidad**

📁 Estructura de Archivos

```
Semana1/
├── code.js                                    # Ejemplos iniciales: funciones básicas y FizzBuzz
├── index.html                                 # Página principal de bienvenida
├── arrays/
│   └── index.js                               # Implementación personalizada de clase Array
└── array/                                     # Métodos y conceptos de arrays
    ├── intro/
    │   ├── index.html
    │   └── main.js
    ├── delete/
    │   ├── index.html
    │   └── main.js
    ├── filter/
    │   ├── index.html
    │   └── main.js
```

```
    ├── map/
    │   ├── index.html
    │   └── main.js
    ├── reduce/
    │   ├── index.html
    │   └── main.js
    ├── immutability/
    │   ├── index.html
    │   └── main.js
    └── functionalProgramiming/
        ├── index.html
        └── main.js
```

📖 Descripción Detallada del Contenido

1. Archivos Raíz

`code.js` - Fundamentos de JavaScript

Contiene tres funciones básicas que introducen conceptos fundamentales:

- `saludar()`: **Función simple que muestra un alert. Demuestra:**
  - **Declaración de función**
  - **Uso de métodos built-in como `alert()`**
  - **Ejecución desde HTML con eventos**
- `teSaludo(nombre)`: **Función con parámetros que:**
  - **Recibe argumentos**
  - **Realiza concatenación de strings**
  - **Interactúa con el usuario**
- `fizzBuzz()`: **Problema clásico de lógica que:**
  - **Implementa bucles `for`**
  - **Usa operadores condicionales (, , `ifelse ifelse`)**
  - **Aplica operador módulo (`%`)**
  - **Imprime en consola con `console.log()`**

`index.html` - Página Principal

- **Estructura básica HTML5**
- **Integración de JavaScript con atributo en botones`onclick`**
- **Incluye comentarios explicativos**
- **Demuestra la invocación de funciones desde elementos HTML**

---

2. `Carpetas arrays/` - Implementación Personalizada

`arrays/index.js` - Clase Array Personalizada

Implementa una clase que simula el comportamiento nativo de JavaScript arrays desde cero:

```
class array {
  constructor()       // Inicializa propiedades length y data
      get(index)          // Obtiene elemento por índice
 push(item)          // Añade elemento al final (incrementa length)
        pop()               // Elimina último elemento
  methodDelete(index) // Elimina elemento en posición específica
  shiftIndex(index)  // Reorganiza índices después de eliminación
}
```

Conceptos Clave:

- Uso de objetos para almacenar datos (`this.data = {}`)
- **Gestión manual de índices**
- **Operaciones de mutación (cambio del array original)**

---

3. `Matriz de carpetas/` - Métodos y Conceptos

`introducción/` - Introducción a Arrays

Enseña operaciones básicas:

- **Declaración:** `const arr = [1, 2, 3]`
- **Acceso por índice:** `arr[0]`
- **Iteración con tradicional**`for`
- **Iteración con** `for...of`
- **Método para agregar elementos**`push()`

`delete/` - Eliminación de Elementos

Demuestra cómo:

- **Remover elementos específicos por índice**
- **Reorganizar los índices restantes**
- **Mantener la integridad de la estructura**

`filter/` - Filtrado de Arrays

Ejemplo: Filtrar nombres que comienzan con una letra específica

```
// Forma estructurada
function searchFirstLetter(collection, letter) {
    const newCollection = [];
    for (const word of collection) {
        if (word[0].toUpperCase() === letter.toUpperCase())
            newCollection.push(word);
    }
    return newCollection;
}

// Usando método filter (programación funcional)
function searchFirstLetter2(collection, letter) {
    return collection.filter(word =>
        word[0].toUpperCase() === letter.toUpperCase()
    );
}
```

Conceptos:

- Predicados (funciones que retornan boolean)
- Inmutabilidad (no modifica el array original)
- Comparación entre enfoque procedural vs funcional

`map/` - Transformación de Arrays

Ejemplo: Sumar un número a todos los elementos

```
// Forma estructurada
function sumNumber(collection, num) {
    const newColletion = [];
    for (let item of collection) {
        newColletion.push(item + num);
    }
    return newColletion;
}

// Usando método map
function sumNumber2(collection, num) {
    return collection.map(item => item + num);
}
```

Conceptos:

- Transformación de elementos
- Creación de nuevos arrays
- Arrow functions para funciones anónimas`=>`

`reduce/` - Acumulación de Valores

Transforma arrays en valores únicos mediante acumulación:

- **Suma de todos los elementos**
- **Multiplicación acumulativa**
- **Conteo de elementos**
- **Creación de objetos desde arrays**

`inmutabilidad/` - Inmutabilidad

Enseña la importancia de no modificar datos originales:

- **Diferencia entre mutación e inmutabilidad**
- **Por qué la inmutabilidad es importante**
- **Métodos que no modifican el array original (mapear, filtrar, reducir)**
- **Métodos que sí modifican (push, pop, splice)**

`functionalProgramiming/` - Programación Funcional

Integra conceptos de programación funcional:

- **Funciones puras (sin efectos secundarios)**
- **Composición de funciones**
- **Uso de callbacks**
- **Encadenamiento de métodos**

🔍 Análisis de Contenido

Fortalezas del Módulo

1. Progresión Pedagógica: Comienza con lo básico (sintaxis JS) y avanza hacia conceptos avanzados (programación funcional)
2. Enfoque Dual: Muestra tanto enfoque procedural como funcional para comparación
3. Interactividad: Ejemplos integrados con HTML permiten aprendizaje práctico
4. Implementación Personalizada: La clase ayuda a entender la mecánica interna`array`
5. Casos de Uso Reales: Problemas como FizzBuzz y filtrado de nombres son relevantes

Temas Cubiertos

Tema	Concepto	Método/Ejemplo
Funciones	**Parámetros y retorno**	`saludar()`, `teSaludo()`
Control de Flujo	**Condicionales e iteración**	`fizzBuzz()`

Estructuras de Datos	Arrays y objetos	Clase personalizada
Métodos de Array	Búsqueda y filtrado	`filter()`
Transformación	Mapeo de datos	`map()`
Acumulación	Reducción de datos	`reduce()`
Paradigmas	Procedural vs Funcional	Comparaciones en ejemplos
Principios	Inmutabilidad	Ejemplos comparativos

Progresión de Dificultad

```
Bajo      → Básico (sintaxis, funciones)
          → Intermedio (arrays, métodos)
          → Avanzado (reduce, composición funcional)
Alto
```

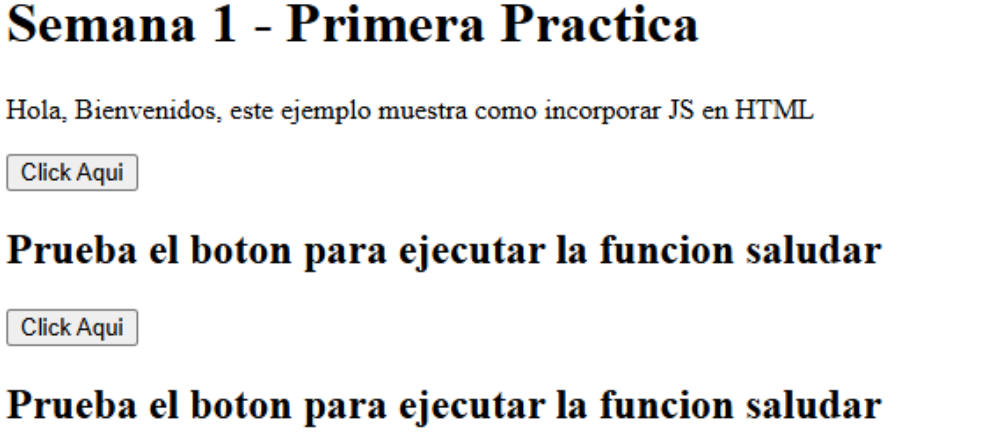
Aplicabilidad Práctica

Este módulo sienta las bases para:

- **Manipulación eficiente de datos**
- **Desarrollo web con JavaScript moderno**
- **Introducción a paradigmas de programación**
- **Comprensión de estructuras de datos para módulos posteriores**

💡 Conclusión

Semana 1 proporciona una base sólida y bien estructurada para aprender JavaScript y el trabajo con arrays. Combina teoría con práctica interactiva, y prepara a los estudiantes para conceptos más avanzados de estructuras de datos en semanas posteriores.



Análisis Detallado - Semana 1: Fundamentos de JavaScript y Estructuras de Arrays

---

📌 Información General

Lenguaje de Programación
- **Tecnología Principal**: JavaScript (ES6+)
- **Plataforma**: Navegador Web (HTML5)
- **Tipo de Ejecución**: Interpretado en el lado del cliente
- **Paradigmas**: Programación Funcional, Programación Orientada a Objetos (POO)

Objetivo Educativo
Esta semana introduce los conceptos fundamentales de JavaScript con enfoque especial en **Estructuras de Datos tipo Array**. Los objetivos específicos son:

1. **Sintaxis Básica**: Familiarizar con la sintaxis fundamental de JavaScript (variables, funciones, control de flujo)
2. **Integración Web**: Aprender cómo incorporar JavaScript en documentos HTML
3. **Estructuras de Arrays**: Dominar operaciones básicas y avanzadas con arrays
4. **Programación Funcional**: Introducir métodos funcionales (map, filter, reduce)
5. **Inmutabilidad**: Entender el concepto de datos inmutables en JavaScript
6. **Implementación Personalizada**: Crear arrays desde cero para comprender su funcionamiento interno

---

📁 Estructura de Archivos

```
Semana1/
├── code.js                    # Ejemplos iniciales: funciones y FizzBuzz
├── index.html                 # Página principal de bienvenida
```

```
    ├── DESCRIPCION.md                          # Descripción del
                            contenido
    ├── README.md                               # Documentación
                            principal
├── ANALISIS_CONTENIDO.md                       # Este archivo (análisis
                            detallado)
                                |
    ├── arrays/                                 # Implementación
                           personalizada
|   └── index.js                                # Clase Array custom con
                          métodos propios
                                |
└── array/                                      # Métodos y conceptos de
                              arrays
    ├── intro/                                  # Introducción a arrays
                           |   ├── index.html
                            |   └── main.js
        ├── delete/                                 # Operación de
                            eliminación
                           |   ├── index.html
                            |   └── main.js
      ├── filter/                                 # Filtrado de datos
                           |   ├── index.html
                            |   └── main.js
      ├── map/                                    # Transformación de
                              datos
                           |   ├── index.html
                            |   └── main.js
     ├── reduce/                                 # Reducción/Agregación
                             de datos
                           |   ├── index.html
                            |   └── main.js
        ├── immutability/                           # Concepto de
                           inmutabilidad
                           |   ├── index.html
                            |   └── main.js
        └── functionalProgramiming/                 # Paradigma de
                        programación funcional
                                ├── index.html
                                 └── main.js
                                 ```

```
```

📚 Descripción Detallada del Contenido

1. **code.js** - Programas Iniciales
Propósito: Introducir conceptos básicos de JavaScript

Conceptos Cubiertos:
- Declaración y ejecución de funciones
- Uso de `alert()` para interacción con usuario
- Concatenación de strings
- Algoritmo FizzBuzz (problema clásico de lógica de programación)
- Loops con `for`
- Operadores aritméticos y de módulo (`%`)

Ejemplos Principales:
```javascript
- function saludar()              // Función simple
- function teSaludo(nombre)       // Función con parámetros
- function fizzBuzz()             // Algoritmo clásico
```

---

2. **index.html** - Página Principal
Propósito: Página de bienvenida interactiva

Características:
- Botones interactivos que llaman funciones JavaScript
- Vinculación de código externo mediante `<script src="code.js">`
- Descripción del proyecto y objetivos de aprendizaje
- Documentación del reto FizzBuzz

---

3. **arrays/index.js** - Implementación Personalizada de Arrays
Propósito: Crear una clase Array desde cero para comprender su estructura interna

Métodos Implementados:
```javascript
class array {
  constructor()          // Inicializa length=0 y data como objeto
      get(index)             // Obtiene elemento en posición
```

```
        push(item)              // Añade elemento al final
    pop()                   // Elimina y retorna último elemento
methodDelete(index)    // Elimina elemento en posición específica
                // ... otros métodos personalizados
                                }
                                ```

                        Aprendizaje:
    - Arrays en JavaScript son objetos con índices numéricos
      - Internamente usan objetos con propiedades numéricas
       - La propiedad `length` controla el tamaño del array

          ### 4. **array/intro/** - Introducción a Arrays
                        Conceptos Cubiertos:
            - Declaración de arrays con tipos mixtos
          - Iteración con `for` clásico (usando índices)
            - Iteración con `for...of` (sobre valores)
            - Método `push()` para agregar elementos
                - Propiedad `length` del array

                        Ejemplo:
                        ```javascript
    let array = [1, 2, "pato", true];  // Array heterogéneo
    array.push(10);                        // Agregar elementos
                                ```

                ### 5. **array/map/** - Método Map
      Propósito: Transformar cada elemento de un array

                        Conceptos:
              - Programación funcional con `map()`
            - Arrow functions `(item) => item + num`
        - Comparación: forma estructurada vs forma funcional
        - Creación de nuevos arrays sin modificar el original

                        Ejemplo:
                        ```javascript
                    // Forma tradicional
            function sumNumber(collection, num) {
```

```
  const newCollection = [];
  for (let item of collection) {
    newCollection.push(item + num);
  }
  return newCollection;
}

// Forma funcional
function sumNumber2(collection, num) {
  return collection.map(item => item + num);
}
```

---

6. **array/reduce/** - Método Reduce

Propósito: Agregación/reducción de datos a un valor único

Conceptos:
- Acumuladores en `reduce()`
- Sintaxis: `reduce((acumulador, item) => acumulador + item, valorInicial)`
- Comparación: loop tradicional vs reduce funcional
- Casos de uso: sumatoria, contadores, agregaciones

Ejemplo:

```javascript
const numbers = [1, 2, 3, 4, 5, 6];

// Forma tradicional
function sumNumber(collection) {
  let sum = 0;
  for (let item of collection) {
    sum += item;
  }
  return sum;
}

// Forma funcional
function sumNumber2(collection) {
  return collection.reduce((sum, item) => sum + item, 0);
}
```

---

7. **array/filter/** - Método Filter
Propósito: Seleccionar elementos que cumplan una condición

Conceptos:
- Filtrado condicional de datos
- Predicados (funciones que retornan boolean)
- Creación de subconjuntos de arrays
- Uso de arrow functions con condicionales

---

8. **array/delete/** - Operación Delete
Propósito: Eliminar elementos de arrays

Conceptos:
- Diferencia entre `delete` y métodos como `pop()` o `splice()`
- Impacto en la estructura del array
- Manipulación de índices

---

9. **array/immutability/** - Inmutabilidad
Propósito: Introducir el concepto de datos inmutables

Conceptos:
- Diferencia entre mutación e inmutabilidad
- Métodos que no modifican el array original
- Métodos que sí modifican el array original
- Beneficios de la programación inmutable (predecibilidad, debugging)

Métodos Inmutables:
- `map()`, `filter()`, `reduce()` - crean nuevos arrays
- `concat()` - combina sin modificar original
- Spread operator `[...array]` - crea copia

Métodos Mutables:
- `push()`, `pop()`, `splice()` - modifican original

---

10. **array/functionalProgramiming/** - Programación Funcional
Propósito: Introducir el paradigma funcional en JavaScript

Conceptos Cubiertos:
- Funciones como objetos de primera clase
- Higher-order functions (funciones que reciben/retornan funciones)
- Composición de funciones
- Arrow functions `=>`
- Funciones puras (sin efectos secundarios)
- Aplicación práctica de `map()`, `filter()`, `reduce()`

🎯 Análisis de Contenido

Progresión de Aprendizaje
1. **Básico**: Funciones simples, FizzBuzz, introducción a arrays
2. **Intermedio**: Métodos de array (`map`, `filter`, `reduce`)
3. **Avanzado**: Programación funcional, inmutabilidad, implementación personalizada

Patrones Educativos
- **Comparación dual**: Cada concepto muestra forma tradicional vs forma funcional
- **Práctica interactiva**: HTML con botones para ejecutar ejemplos
- **De adentro hacia afuera**: Primero implementación personalizada, luego uso de métodos nativos

Tecnologías Utilizadas
Tecnología	Uso
JavaScript ES6+	Código principal
HTML5	Estructura de páginas
Console	Debugging y output
Arrow Functions	Programación funcional
Métodos Nativos	map, filter, reduce, push, pop

Objetivos de Aprendizaje Logrados
✅ Comprensión de sintaxis básica de JavaScript
✅ Manejo completo de arrays
✅ Introducción a programación funcional
✅ Entendimiento de mutabilidad e inmutabilidad
✅ Implementación personalizada de estructuras de datos

✅ Buenas prácticas de código limpio

💡 Recursos Pedagógicos

Tipos de Ejemplos Proporcionados
1. **Ejemplos Interactivos**: Botones HTML que disparan funciones
2. **Ejemplos en Consola**: Output mediante `console.log()`
3. **Comparativas**: Código tradicional vs funcional lado a lado
4. **Implementaciones**: Crear estructuras desde cero

Nivel de Dificultad
- 🟢 **Principiante**: code.js, array/intro
- 🟡 **Intermedio**: array/map, array/filter, array/reduce
- 🔴 **Avanzado**: arrays/index.js (implementación personalizada), functionalProgramiming

---

🚀 Cómo Utilizar Este Material

1. **Comenzar por `index.html`**: Abre en navegador para interfaz interactiva
2. **Revisar `code.js`**: Entiende conceptos básicos
3. **Explorar cada carpeta `array/`**: Sigue la estructura de adentro hacia afuera
4. **Ejecutar ejemplos**: Abre las páginas HTML en navegador y usa la consola (F12)
5. **Estudiar `arrays/index.js`**: Comprende cómo funcionan los arrays internamente

---

📖 Resumen Ejecutivo

Semana 1 proporciona una base sólida en JavaScript enfocándose en Arrays y Programación Funcional. El material combina teoría (documentación), práctica (ejemplos interactivos) e implementación (arrays personalizados). Está diseñado para estudiantes sin experiencia en programación, con progresión clara de dificultad y múltiples formas de aprendizaje (visual, interactiva, práctica).

SEMANA 2

Semana 2 - Análisis de Contenido

📋 Información General

Aspecto	Detalle
Lenguaje Principal	HTML5 + JavaScript
Tecnologías	HTML, CSS, JavaScript (Programación Orientada a Objetos)
Enfoque	Fundamentos de HTML y Estructuras de Datos (Árboles de Búsqueda Binaria)

🎯 Objetivo

Introducir a los estudiantes en:

1. **HTML5: Elementos semánticos, estructura de documentos web, navegación**
2. **Navegación Web: Enlaces internos y externos**
3. **Conceptos Avanzados: Implementación de estructura de datos (Binary Search Tree) en JavaScript**
4. **Organización de Proyectos: Estructura de sitios web con múltiples páginas**

📁 Estructura de Archivos

```
Semana2/
│
├── index.html                          # Página principal - Fundamentos de HTML
├── acerca.html                         # Página "Acerca de"
├── nosotros.html                       # Página "Nosotros"
├── HTML inicia con la etiqueta !DOCTYP.txt  # Notas sobre DOCTYPE
│
├── assets/                             # Carpeta de recursos (imágenes)
│   ├── 1.jpg
│   ├── 2.jpg
│   ├── ... (hasta 9.jpg)
│   └── 8.png, 9.png
│
```

```
├── binarySearchTree/                # Implementación de árbol de búsqueda
                                    binaria
    │   ├── index.html                    # Página HTML para demostración
│   └── main.js                         # Clase BinarySearchTree (210 líneas)
                                        │
    ├── misitio/                          # Proyecto pequeño de sitio web
│   ├── code.js                         # Código JavaScript (funciones básicas)
            │   └── index.html                    # Página HTML
                                        │
  └── tarea/                            # Carpeta con ejercicios prácticos
      ├── index.html                      # Página principal de tareas
      ├── conciertos.html                 # Ejercicio sobre conciertos
     └── fotos.html                       # Ejercicio sobre galería de fotos
```

📝 Descripción Detallada de Contenido

1. index.html - Página Principal

Archivo fundamental que enseña los elementos HTML básicos:

Temas cubiertos:

- **Encabezados: a `<h1><h6>`**
- Párrafos: y formato previo `<p><pre>`
- Listas: Ordenadas y desordenadas `<ol><ul>`
- Formato de texto:
    - Negritas: , `<b><strong>`
    - **Cursiva: , `<i><em>`**
    - **Marcado: `<mark>`**
    - **Subrayado: `<u>`**
    - **Tachado: `<del>`**
    - **Insertado: `<ins>`**
    - **Superíndice/Subíndice: , `<sup><sub>`**
- Enlaces externos: Google, YouTube, Facebook, Instagram, Twitter
- Enlaces internos: Navegación entre páginas (Inicio, Nosotros, Acerca de)
- Imágenes: 9 imágenes con atributos width y height
- Navegación: Elemento con listas de enlaces`<nav>`

2. acerca.html y nosotros.html

Páginas de navegación que demuestran:

- **Estructura básica de documentos HTML**
- **Navegación entre múltiples páginas**
- **Consistencia en la estructura de sitios web**

3. binarySearchTree/main.js - Estructura de Datos

Implementación completa de un Árbol de Búsqueda Binaria (BST):

Clases:

- `Node`: Representa un nodo con:
  - `value`: Valor almacenado
  - `left`: Referencia al hijo izquierdo
  - `right`: Referencia al hijo derecho
- `BinarySearchTree`: Árbol con métodos:
  - `insert(value)`: Inserta valores manteniendo la propiedad BST
  - `search(value)`: Busca un valor en el árbol
  - `showInOrder()`: Recorrido en orden (izquierda-raíz-derecha)
  - `showInPreOrder()`: Recorrido preorden (raíz-izquierda-derecha)
  - `showInPostOrder()`: Recorrido postorden (izquierda-derecha-raíz)

Conceptos:

- Programación Orientada a Objetos
- Recursión para recorridos
- Algoritmos de búsqueda binaria

4. misitio/ - Proyecto de Sitio Web Pequeño

Demostración de proyecto web pequeño con:

- `index.html`: Estructura HTML
- `code.js`: Código JavaScript con funciones básicas

5. tarea/ - Ejercicios Prácticos

Carpeta con tareas para practicar:

- index.html: Página principal de tareas
- conciertos.html: Ejercicio sobre eventos o conciertos
- fotos.html: Ejercicio de galería de fotos

🔍 Análisis de Contenido

Conceptos HTML Clave

✓ Semántica HTML5: Uso correcto de etiquetas semánticas

✓ Estructura de documentos: DOCTYPE, meta tags, navegación

✓ Accesibilidad: Atributos alt en imágenes, navegación clara

✓ Navegación: Enlaces internos y externos (target="_self" vs target="_blank")

Conceptos JavaScript/Programación

✓ POO: Clases y constructores

✓ Estructuras de datos: Árbol de búsqueda binaria

✓ Algoritmos: Búsqueda y recorridos (in-order, pre-order, post-order)

✓ Recursión: Implementada en métodos de recorrido

Puntos de Aprendizaje Progresivo

1. Fase Básica: Fundamentos HTML (tags, atributos, estructura)
2. Fase Media: Navegación entre páginas, gestión de activos
3. Fase Avanzada: Estructuras de datos complejas en JavaScript

Oportunidades de Mejora

- **Agregar CSS para estilización visual**
- **Implementar funcionalidad interactiva en el BST**
- **Optimizar la estructura de carpetas para proyectos más grandes**
- **Agregar validación de formularios en tareas**

🎓 Nivel de Dificultad

Tema	Nivel
Elementos HTML básicos	Principiante
Navegación entre páginas	Principiante
Formato de texto avanzado	Principiante
Árbol de Búsqueda Binaria	Intermedio/Avanzado
Recorridos recursivos	Avanzado

💡 Conclusión

Semana 2 proporciona una base sólida en HTML5 combinada con introducción a estructuras de datos avanzadas en JavaScript. El contenido avanza desde conceptos básicos de web hasta implementaciones de algoritmos complejos, permitiendo a los estudiantes entender tanto el desarrollo frontend como la ciencia de datos en JavaScript.

Semana 2 Curso de HTML 1

Semana 2 Curso de HTML 2

Semana 2 Curso de HTML 3

Semana 2 Curso de HTML 4

Semana 2 Curso de HTML 5

Semana 2 Curso de HTML 6

Hola a todos.
Bienvenidos

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Doloremque iste beatae expedita tenetur inventore iusto possimus quisquam. Eos nam autem, ducimus necessitatibus quidem sequi commodi nobis temporibus eveniet quos consequatur.

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Vitae vero quasi commodi est, quaerat natus beatae exercitationem a sit temporibus nobis. Quae eaque nisi nihil dolore cumque ut incidunt nulla!

Lorem ipsum dolor sit amet consectetur adipisicing elit. Incidunt sequi quibusdam possimus atque, placeat, eveniet quam error numquam suscipit, dolore eaque vero laboriosam fuga molestias! Mollitia odit porro officia totam.

```
Lorem ipsum dolor sit amet consectetur adipisicing elit.
Eaque illo deserunt quaerat veritatis aspernatur, dolores nobis veniam sapiente fugiat praesentium ratione error culpa.
Quis animi vitae, veniam harum esse quod.
```

Lista ordenada

Semana 2 - Descripción Completa del Contenido

🔤 Lenguajes Utilizados

Lenguaje	Descripción	Uso
HTML5	Markup Language para estructuración web	Estructuras de páginas, navegación, contenido
JavaScript (ES6+)	Lenguaje de programación orientado a objetos	Estructuras de datos, algoritmos, lógica
CSS	Stylesheets (implicado en assets)	Estilos y presentación visual

---

📁 Estructura de Archivos Detallada

```
Semana2/
│
├── 📄 index.html                          [Página principal - Fundamentos HTML]
├── 📄 acerca.html                         [Página "Acerca de"]
├── 📄 nosotros.html                       [Página "Nosotros"]
├── 📄 CONTENIDO.md                        [Análisis de contenido]
├── 📄 HTML inicia con la etiqueta !DOCTYP.txt  [Notas pedagógicas]
│
├── 📂 assets/                             [Recursos multimedia]
│   ├── 1.jpg through 9.jpg                [Imágenes JPG]
│   ├── 7.png, 8.png, 9.png                [Imágenes PNG]
│   └── [Total: 9 imágenes para galería]
│
├── 📂 binarySearchTree/                   [Implementación de estructura de datos avanzada]
│   ├── index.html                         [Interfaz de demostración]
│   └── main.js                            [Clases Node y BinarySearchTree - 210 líneas]
│
├── 📂 misitio/                            [Proyecto pequeño de sitio web]
│   ├── code.js                            [Código JavaScript básico]
```

```
│   └── index.html                          [Página HTML del sitio]
│
└── 📂 tarea/                               [Ejercicios prácticos para estudiantes]
    ├── index.html                          [Página de índice de tareas]
    ├── conciertos.html                     [Ejercicio: evento/conciertos]
    └── fotos.html                          [Ejercicio: galería de fotos]
```

🎯 Objetivo General

Semana 2 tiene como objetivo proporcionar a los estudiantes:

1. **Fundamentos web**: Comprensión de HTML5 semántico y estructura de documentos web
2. **Programación estructurada**: Introducción a JavaScript con principios de programación orientada a objetos
3. **Estructuras de datos**: Implementación práctica de una estructura de datos compleja (Árbol de Búsqueda Binaria)
4. **Arquitectura de proyectos**: Organizando múltiples páginas HTML con navegación coherente
5. **Habilidades prácticas**: Creación de sitios web funcionales con múltiples páginas

---

📋 Descripción de Contenido por Componente

1️⃣ **Página Principal (index.html)**

Propósito: Demostrar todos los elementos HTML básicos de manera educativa

Contenido Cubierto:
- ✅ **Encabezados jerárquicos**: `<h1>` a `<h6>`
- ✅ **Párrafos y preformato**: `<p>`, `<pre>`
- ✅ **Listas**: Ordenadas `<ol>` y desordenadas `<ul>`
- ✅ **Formato de texto**:

- Negritas: `<b>`, `<strong>`
- Cursiva: `<i>`, `<em>`
- Especiales: `<mark>`, `<small>`, `<del>`, `<ins>`, `<u>`
- Notación científica: `<sup>`, `<sub>`
- ✅ **Enlaces**:
- Externos a redes sociales y plataformas populares
- Internos con navegación entre páginas (`target="_self"` vs `target="_blank"`)
- ✅ **Imágenes**: Galería de 9 imágenes con dimensiones especificadas (240x240px)
- ✅ **Navegación**: Estructura `<nav>` con menú de enlaces internos

Líneas de código: 93 líneas

2️⃣ **Páginas de Navegación (acerca.html, nosotros.html)**

Propósito: Demostrar navegación entre múltiples páginas y consistencia estructural

Características:
- Estructura DOCTYPE y metadatos HTML5
- Enlaces de navegación intercalados
- Contenido específico de cada sección
- Demostración de sitios web multi-página

3️⃣ **Árbol de Búsqueda Binaria (binarySearchTree/)**

Propósito: Implementar una estructura de datos avanzada utilizada en algoritmos de búsqueda y ordenamiento

Clase Node

```javascript
class Node {
  constructor(value) {
    this.value = value;  // Valor almacenado en el nodo
    this.left = null;    // Referencia al hijo izquierdo
    this.right = null;   // Referencia al hijo derecho
  }
```

```
}
```

Clase BinarySearchTree

Métodos principales:

- `insert(value)`: Inserta un valor manteniendo la propiedad BST
  - Valores menores van a la izquierda
  - Valores mayores van a la derecha
  - Evita duplicados
- `search(value)`: Busca eficientemente un valor en O(log n)
- `showInOrder()`: Recorrido en orden (izquierda → raíz → derecha)
  - Resultado: valores en orden ascendente
- `showInPreOrder()`: Recorrido preorden (raíz → izquierda → derecha)
- `showInPostOrder()`: Recorrido postorden (izquierda → derecha → raíz)

Conceptos Enseñados:

- 🔹 Programación Orientada a Objetos (clases, constructores)
- 🔹 Recursión (en métodos de recorrido)
- 🔹 Algoritmos de búsqueda binaria
- 🔹 Complejidad temporal O(log n) para operaciones básicas
- 🔹 Estructuras de árbol y nodos

Líneas de código: 210 líneas

4️⃣ **Proyecto Pequeño (misitio/)**

Propósito: Aplicar conceptos básicos en un proyecto pequeño e integrado

Contenido:

- `index.html`: Estructura HTML simple
- `code.js`: Funciones JavaScript básicas (función `botones()`)
  - Demuestra integración de HTML con lógica JavaScript

5️⃣ **Tareas Prácticas (tarea/)**

Propósito: Ejercicios prácticos para que los estudiantes practiquen

Componentes:
- **index.html**: Página de índice con links a ejercicios
- **conciertos.html**: Ejercicio enfocado en eventos/conciertos
- **fotos.html**: Ejercicio de galería de fotos
- Oportunidad para practicar navegación y estructura HTML

🔍 Análisis Detallado de Contenido

Distribución Temática

Tema	Profundidad	Complejidad
HTML Semántico	Profunda	Básica
Navegación Web	Completa	Básica
Formato de Texto	Completa	Básica
Enlaces y Recursos	Completa	Básica
Estructuras de Datos	Profunda	Avanzada
Algoritmos de Búsqueda	Profunda	Avanzada
POO en JavaScript	Moderada	Intermedia/Avanzada

Progresión de Aprendizaje

```
FASE 1: Fundamentales HTML (Básico)
↓
Encabezados → Párrafos → Listas → Formatos

FASE 2: Navegación Web (Básico-Intermedio)
↓
Enlaces externos → Enlaces internos → Menú de navegación

FASE 3: Multimedia e Integración (Intermedio)
↓
Imágenes → Galerías → Proyectos multi-página

FASE 4: Estructuras de Datos (Avanzado)
↓
Clases → Constructores → Árboles Binarios → Algoritmos
```

```

### Conexión entre Componentes

1. **HTML ↔ JavaScript**: Los archivos HTML pueden importar y utilizar el código JavaScript del BST
2. **Educación Progresiva**: Comienza con lo simple (HTML básico) y avanza a conceptos complejos (estructuras de datos)
3. **Práctica Aplicada**: Las tareas permiten aplicar lo aprendido en proyectos reales

---

## 💡 Análisis de Fortalezas y Oportunidades

### ✅ Fortalezas

1. **Cobertura completa de HTML5**: Todos los elementos fundamentales están presentes
2. **Ejemplo avanzado de estructura de datos**: BST es una estructura compleja que demuestra profundidad
3. **Navegación práctica**: Enseña cómo funcionan los sitios web reales con múltiples páginas
4. **Escalabilidad**: La estructura permite agregar más proyectos y complejidad

### 🚀 Oportunidades de Mejora

1. **CSS/Estilización**: Agregar hoja de estilos para que el sitio sea visualmente atractivo
2. **Interactividad HTML-JavaScript**: Conectar el BST con una interfaz HTML interactiva
3. **Validación de formularios**: En las tareas de conciertos y fotos
4. **DOM Manipulation**: Mostrar cómo JavaScript modifica el HTML dinámicamente
5. **Responsive Design**: Hacer que los sitios sean adaptables a dispositivos móviles
6. **Comentarios detallados**: Documentación de código en los archivos JavaScript

---
```

📊 Estadísticas del Contenido

Métrica	Cantidad
Archivos HTML	6
Archivos JavaScript	2
Imágenes	9
Páginas de navegación	3
Carpetas de proyecto	4
Líneas de código (BST)	210
Total de directorios	5

🎓 Niveles de Dificultad por Tema

🏆 Conclusión

Semana 2 proporciona una transición efectiva de conceptos fundamentales de HTML a principios avanzados de programación. El contenido está estratégicamente diseñado para:

1. 🌱 **Construir bases sólidas** en tecnologías web estándar
2. 📈 **Escalar en complejidad** de manera progresiva y lógica

3. 🔗 **Conectar teoría con práctica** mediante proyectos aplicados
4. 💪 **Desafiar a los estudiantes** con estructuras de datos complejas

Esta semana es crucial porque establece patrones de pensamiento en programación orientada a objetos que serán reutilizados en semanas posteriores.

📚 Recursos Relacionados

- **CONTENIDO.md**: Análisis detallado adicional
- **HTML inicia con la etiqueta !DOCTYP.txt**: Notas sobre estructura HTML
- **Carpeta assets/**: Recursos multimedia para demostración
- **binarySearchTree/main.js**: Implementación completa de estructura de datos

---

Última actualización: Diciembre 2025
Nivel educativo: Principiante a Intermedio/Avanzado

SEMANA 3

Descripción de la Carpeta Semana3

Lenguaje

- Director: JavaScript
- Secundario: HTML5 y CSS3
- Enfoque: Educativo/Didáctico

Estructura de Archivos

```
Semana3/
├── CONTENIDO.md                # Descripción del contenido de la semana
├── RESUMEN_SEMANA3.md          # Resumen vacío (placeholder)
├── index.html                  # Página principal de prácticas
├── mistyle.css                 # Estilos globales
│
├── doublyLinkedList/           # Implementación de lista doblemente enlazada
│   ├── index.html              # Interfaz interactiva
│   ├── main.js                 # Código de la estructura
│   └── images/                 # Recursos gráficos
│
└── graphs/                     # Implementaciones de grafos
    ├── graphs.js               # Métodos y utilidades de grafos
    └── buildGraph.js           # Construcción y representación de grafos
```

Objetivo

Proporcionar ejemplos prácticos e implementaciones de estructuras de datos fundamentales en ciencias de la computación, específicamente:

- **Listas doblemente enlazadas: Estructura con navegación bidireccional**
- **Grafos: Estructura para representar relaciones complejas entre nodos**

El propósito educativo es permitir a los estudiantes comprender cómo funcionan estas estructuras a través de código ejecutable y demostraciones visuales.

Descripción General

DoublelyLinkedList (Lista doblemente enlazada)

La carpeta contiene una implementación completa de una lista enlazada donde cada nodo tiene dos punteros:`doublyLinkedList/`

- **`Puntero a continuación`**: Apunta al siguiente nodo
- `Puntero prev`: Apunta al nodo anterior

Métodos principales:

- **`add(value)`: Agrega un elemento al final de la lista**
- **`show()`: Muestra la lista en orden hacia adelante**
- **`reverse()`: Muestra la lista en orden inverso**
- **`delete(value)`: Elimina un elemento específico**
- **`clear()`: Limpia toda la lista**

Grafos (Grafos)

La carpeta contiene utilidades para trabajar con grafos, explorando diferentes representaciones:`graphs/`

- Lista de Edges: Lista de aristas (conexiones)
- Lista adyacente: Lista de adyacencia (nodos vecinos)
- Matriz adyacente: Matriz de adyacencia (tabla de conexiones)

Análisis de Contenido

Fortalezas

✅ Código didáctico: Implementaciones limpias y fáciles de entender

✅ Cobertura dual: Aborda dos estructuras fundamentales

✅ Demostraciones prácticas: Incluye interfaces HTML para visualizar las estructuras

✅ Variedad de representaciones: En grafos muestra múltiples enfoques

Áreas de Mejora

⚠️ Documentación limitada: Los archivos JavaScript podrían tener más comentarios explicativos

⚠️ Funcionalidad incompleta: La implementación de en DoublyLinkedList se corta abruptamente

⚠️ Sin pruebas unitarias: No hay tests para validar el funcionamiento

⚠️ buildGraph.js: Necesita ejemplos prácticos de cómo construir grafos

`delete()`

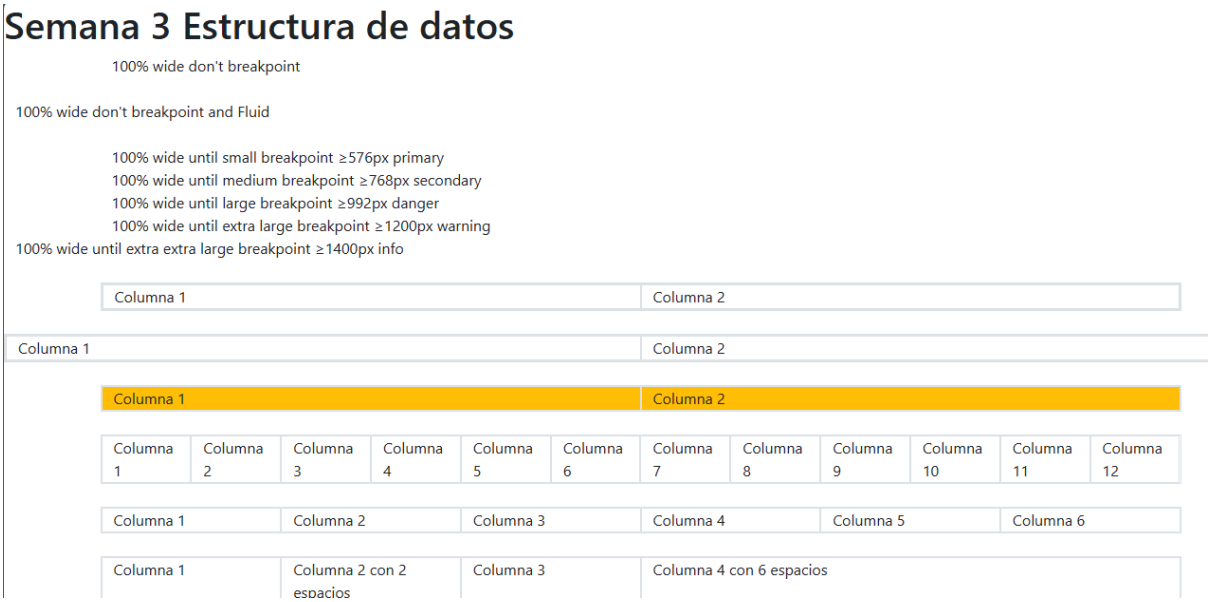
Recomendaciones de Uso

1. Para aprender: Lee y seguido de pruebas en la consola`main.js` `graphs.js`
2. Para practicar: Abre en el navegador para ver las demostraciones`index.html`
3. Para mejorar:
    - Completa la implementación de en DoublyLinkedList`delete()`
    - **Agrega métodos de búsqueda (DFS, BFS) en la clase de grafos**
    - **Implementa casos de prueba (pruebas unitarias)**
    - **Documenta cada método con JSDoc**

Conceptos Clave Cubiertos

- Navegación bidireccional en listas enlazadas
- Representaciones de grafos (matriz, lista, aristas)
- Operaciones básicas (inserción, eliminación, recorrido)
- Estructuras dinámicas de datos

Documento generado como análisis de la carpeta Semana3 del curso de Estructuras de Datos (2025)



Descripción de la Carpeta Semana3

Lenguaje
- **Principal:** JavaScript
- **Secundario:** HTML5 y CSS3
- **Enfoque:** Educativo/Didáctico

Estructura de Archivos

```
Semana3/
├── CONTENIDO.md                # Descripción del contenido de la semana
├── RESUMEN_SEMANA3.md          # Resumen vacío (placeholder)
├── index.html                  # Página principal de prácticas
├── mistyle.css                 # Estilos globales
│
├── doublyLinkedList/           # Implementación de lista doblemente enlazada
│   ├── index.html              # Interfaz interactiva
│   ├── main.js                 # Código de la estructura
│   └── images/                 # Recursos gráficos
│
└── graphs/                     # Implementaciones de grafos
    ├── graphs.js               # Métodos y utilidades de grafos
    └── buildGraph.js           # Construcción y representación de grafos
```

Objetivo
Proporcionar ejemplos prácticos e implementaciones de **estructuras de datos** fundamentales en ciencias de la computación, específicamente:
- **Listas Doblemente Enlazadas:** Estructura con navegación bidireccional
- **Grafos:** Estructura para representar relaciones complejas entre nodos

El propósito educativo es permitir a los estudiantes comprender cómo funcionan estas estructuras a través de código ejecutable y demostraciones visuales.

Descripción General

DoublyLinkedList (Lista Doblemente Enlazada)

La carpeta `doublyLinkedList/` contiene una implementación completa de una lista enlazada donde cada nodo tiene dos punteros:

- **Puntero `next`:** Apunta al siguiente nodo
- **Puntero `prev`:** Apunta al nodo anterior

Métodos principales:

- `add(value)`: Agrega un elemento al final de la lista
- `show()`: Muestra la lista en orden hacia adelante
- `reverse()`: Muestra la lista en orden inverso
- `delete(value)`: Elimina un elemento específico
- `clear()`: Limpia toda la lista

Graphs (Grafos)

La carpeta `graphs/` contiene utilidades para trabajar con grafos, explorando diferentes representaciones:

- **Edge List:** Lista de aristas (conexiones)
- **Adjacent List:** Lista de adyacencia (nodos vecinos)
- **Adjacent Matrix:** Matriz de adyacencia (tabla de conexiones)

Análisis de Contenido

Fortalezas

✅ **Código didáctico:** Implementaciones limpias y fáciles de entender
✅ **Cobertura dual:** Aborda dos estructuras fundamentales
✅ **Demostraciones prácticas:** Incluye interfaces HTML para visualizar las estructuras
✅ **Variedad de representaciones:** En grafos muestra múltiples enfoques

Áreas de Mejora

⚠️ **Documentación limitada:** Los archivos JavaScript podrían tener más comentarios explicativos
⚠️ **Funcionalidad incompleta:** La implementación de `delete()` en DoublyLinkedList se corta abruptamente
⚠️ **Sin pruebas unitarias:** No hay tests para validar el funcionamiento
⚠️ **buildGraph.js:** Necesita ejemplos prácticos de cómo construir grafos

Recomendaciones de Uso

1. **Para aprender:** Lee `main.js` y `graphs.js` seguido de pruebas en la consola

2. **Para practicar:** Abre `index.html` en el navegador para ver las demostraciones
3. **Para mejorar:**
- Completa la implementación de `delete()` en DoublyLinkedList
- Agrega métodos de búsqueda (DFS, BFS) en la clase de grafos
- Implementa casos de prueba (unit tests)
- Documenta cada método con JSDoc

Conceptos Clave Cubiertos
- **Navegación bidireccional** en listas enlazadas
- **Representaciones de grafos** (matriz, lista, aristas)
- **Operaciones básicas** (inserción, eliminación, recorrido)
- **Estructuras dinámicas** de datos

Documento generado como análisis de la carpeta Semana3 del curso de Estructuras de Datos (2025)

SEMANA 4

Semana 4 - Estructuras de Datos: Introducción a Gráfos

Información General

Aspecto	Descripción
Lenguaje	JavaScript (ES6+)
Tema Principal	Gráfos (Graphs) - Estructura de Datos Fundamental
Nivel	Intermedio
Requisitos	Conocimiento básico de JavaScript y POO

Estructura de Archivos

```
Semana4/
├── code.js                    # Ejemplos básicos de variables y funciones numéricas
├── index.html                 # Página HTML sobre tipos de datos primitivos
└── graph/
    ├── index.html             # Página HTML para el ejemplo de gráfos
    └── main.js                # Implementación de la estructura Graph
```

Objetivo

Introducir a los estudiantes a la estructura de datos de Gráfos, una de las estructuras más importantes en ciencias de la computación. Se busca comprender:

- **Qué es un gráfo y sus componentes (nodos y aristas)**
- **Cómo representar gráfos mediante código orientado a objetos**
- **La relación entre nodos a través de aristas**
- **Métodos básicos para manipular gráfos (agregar nodos, crear conexiones, visualizar)**

Descripción General

Componentes Principales

1. code.js

Contiene ejemplos introductorios de operaciones matemáticas y numéricas en JavaScript:

- **Declaración de variables numéricas (enteros, decimales, números con separadores)**
- **Funciones de iteración y operaciones matemáticas**
- **Base conceptual para trabajar con datos antes de estructuras complejas**

2. index.html

Página educativa que explica:

- **Tipos de datos primitivos en JavaScript**
- **Cómo declarar variables numéricas**
- **Diferencia entre tipos de datos básicos y objetos**
- **Tabla comparativa de constructores para números**

3. gráfico/main.js

Implementación completa de una estructura de Gráfo utilizando Programación Orientada a Objetos:

Clases implementadas:

- `Graph`**: Estructura principal que gestiona nodos**
- `Node`**: Representa un nodo individual del gráfo**

Métodos principales:

- `addNode(value)`**: Añade un nodo al gráfo**
- `addEdge(startValueNode, endValueNode)`**: Crea una conexión dirigida entre dos nodos**
- `show()`**: Visualiza la estructura del gráfo en consola**

4. graph/index.html

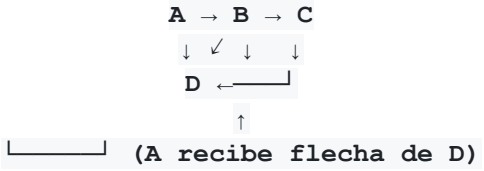
Página HTML básica para incorporar el script de gráfos en un navegador.

Análisis de Contenido

Análisis del Gráfo (grafo/main.js)

El ejemplo muestra un gráfo dirigido con las siguientes características:

Estructura Implementada:



Desglose de conexiones:

Nodo	Conexiones
Un	B, D
B	C, D
C	D
D	Un

Características Técnicas:

1. Estructura del Nodo:
   - `value`**: Almacena el identificador del nodo**
   - `edges`**: Array que contiene referencias a nodos conectados**
2. **Uso de Map:**
   - `this.nodes = new Map()` **permite acceso eficiente O(1) a nodos**
3. **Relaciones Direccionales:**
   - **Las aristas son dirigidas (A→B es diferente de B→A)**
   - **Cada nodo mantiene lista de sus conexiones salientes**
4. **Visualización:**
   - **El método imprime cada nodo con sus conexiones**`show()`
   - **Salida esperada:**

```
A -> B,D
B -> C,D
C -> D
D -> A
```

5.

Conceptos Aprendidos

1. Tipos de Datos en JavaScript:
   - Números primitivos vs objetos Número

    - Separadores numéricos para legibilidad (5_000_000)
2. Gráfos:
    - Definición: Colección de nodos conectados por aristas
    - Tipos: Dirigidos y no dirigidos
    - Aplicaciones: Redes, mapas, relaciones sociales, etc.
3. Programación Orientada a Objetos:
    - Uso de clases para modelar entidades
    - Encapsulación de datos y métodos
    - Relaciones entre objetos
4. Estructuras de Datos Asociativas:
    - Uso de Map para almacenamiento eficiente
    - Búsqueda rápida de nodos por identificador

Conclusión

Semana 4 introduce una estructura fundamental en programación mediante un enfoque progresivo:

1. **Repaso de conceptos básicos (variables numéricas)**
2. **Introducción a una estructura de datos compleja (Gráfos)**
3. **Implementación práctica con POO**

Este es un punto de inflexión importante donde los estudiantes transicionan de estructuras simples a estructuras más complejas que tienen aplicaciones reales en algoritmos de búsqueda, análisis de redes y muchísimas áreas de la informática.

Tipos de Datos primitivos

En Javascript, crear variables numéricas es muy sencillo, pero hay muchísimos matices que se deben conocer y que necesitamos dominar para trabajar correctamente con números y anticiparnos a posibles situaciones.

¿Qué es una variable numérica?

```
En Javascript, los números son uno de los tipos de datos básicos (tipos primitivos),
que, para crearlos, simplemente basta con escribirlos literalmente.
No obstante, como en Javascript todo se puede representar con objetos (como veremos más adelante)
también se pueden declarar mediante la palabra clave new:
```

Constructor	Descripcion
new Number(number)	Crea un objeto numérico a partir del número number pasado por parámetro.
number	Simplemente, el número en cuestión. Notación preferida.

Semana 4 - Estructuras de Datos: Introducción a Gráfos

Información General

Aspecto	Descripción
Lenguaje	JavaScript (ES6+)
Tema Principal	Gráfos (Graphs) - Estructura de Datos Fundamental
Nivel	Intermedio
Requisitos	Conocimiento básico de JavaScript y POO

Estructura de Archivos

```
Semana4/
├── code.js                 # Ejemplos básicos de variables y funciones numéricas
├── index.html              # Página HTML sobre tipos de datos primitivos
└── graph/
    ├── index.html          # Página HTML para el ejemplo de gráfos
    └── main.js             # Implementación de la estructura Graph
```

Objetivo

Introducir a los estudiantes a la estructura de datos de **Gráfos**, una de las estructuras más importantes en ciencias de la computación. Se busca comprender:

- Qué es un gráfo y sus componentes (nodos y aristas)
- Cómo representar gráfos mediante código orientado a objetos
- La relación entre nodos a través de aristas
- Métodos básicos para manipular gráfos (agregar nodos, crear conexiones, visualizar)

Descripción General

Componentes Principales

1. **code.js**
Contiene ejemplos introductorios de operaciones matemáticas y numéricas en JavaScript:
- Declaración de variables numéricas (enteros, decimales, números con separadores)
- Funciones de iteración y operaciones matemáticas
- Base conceptual para trabajar con datos antes de estructuras complejas

2. **index.html**
Página educativa que explica:
- Tipos de datos primitivos en JavaScript
- Cómo declarar variables numéricas
- Diferencia entre tipos de datos básicos y objetos
- Tabla comparativa de constructores para números

3. **graph/main.js**
Implementación completa de una estructura de **Gráfo** utilizando Programación Orientada a Objetos:

Clases implementadas:
- `Graph`: Estructura principal que gestiona nodos
- `Node`: Representa un nodo individual del gráfo

Métodos principales:
- `addNode(value)`: Añade un nodo al gráfo
- `addEdge(startValueNode, endValueNode)`: Crea una conexión dirigida entre dos nodos
- `show()`: Visualiza la estructura del gráfo en consola

4. **graph/index.html**
Página HTML básica para incorporar el script de gráfos en un navegador.

Análisis de Contenido

Análisis del Gráfo (graph/main.js)

El ejemplo muestra un **gráfo dirigido** con las siguientes características:

Estructura Implementada:

```
A → B → C
↓ ↙ ↓   ↓
D ←——┘
↑
└——┘ (A recibe flecha de D)
```

Desglose de conexiones:

Nodo	Conexiones
A	B, D
B	C, D
C	D
D	A

Características Técnicas:

1. **Estructura del Nodo:**
   - `value`: Almacena el identificador del nodo
   - `edges`: Array que contiene referencias a nodos conectados

2. **Uso de Map:**
   - `this.nodes = new Map()` permite acceso eficiente O(1) a nodos

3. **Relaciones Direccionales:**
   - Las aristas son **dirigidas** (A→B es diferente de B→A)
   - Cada nodo mantiene lista de sus conexiones salientes

4. **Visualización:**
   - El método `show()` imprime cada nodo con sus conexiones
   - Salida esperada:

```
A -> B,D
B -> C,D
C -> D
D -> A
```

Conceptos Aprendidos

1. **Tipos de Datos en JavaScript:**
- Números primitivos vs objetos Number
- Separadores numéricos para legibilidad (5_000_000)

2. **Gráfos:**
- Definición: Colección de nodos conectados por aristas
- Tipos: Dirigidos y no dirigidos
- Aplicaciones: Redes, mapas, relaciones sociales, etc.

3. **Programación Orientada a Objetos:**
- Uso de clases para modelar entidades
- Encapsulación de datos y métodos
- Relaciones entre objetos

4. **Estructuras de Datos Asociativas:**
- Uso de Map para almacenamiento eficiente
- Búsqueda rápida de nodos por identificador

Conclusión

Semana 4 introduce una estructura fundamental en programación mediante un enfoque progresivo:
1. Repaso de conceptos básicos (variables numéricas)
2. Introducción a una estructura de datos compleja (Gráfos)
3. Implementación práctica con POO

Este es un punto de inflexión importante donde los estudiantes transicionan de estructuras simples a estructuras más complejas que tienen aplicaciones reales en algoritmos de búsqueda, análisis de redes y muchísimas áreas de la informática.

SEMANA 5

Descripción Completa - Semana 5

🌐 Lenguajes Utilizados

- JavaScript - Lógica de programación y funcionalidad interactiva
- HTML - Estructura y visualización de ejemplos
- CSS - Estilos básicos en módulos específicos
- JSON - Formato de datos para ejemplos

📁 Estructura de Archivos

```
Semana5/
├── code.js                          # Script principal con ejemplos de manipulación DOM
├── index.html                       # Página raíz de la semana
├── inner.html                       # Página auxiliar/interna
├── CONTENIDO.md                     # Resumen de contenido
│
├── introduccion/                    # Módulo introductorio
│   ├── funtions/                    # Tema: Funciones
│   │   ├── index.html
│   │   └── main.js
│   │
│   └── variable/                    # Tema: Variables
│       ├── index.html
│       ├── main.css
│       └── main.js
│
├── json/                            # Módulo: Manejo de JSON
│   ├── index.html
│   ├── main.js                      # Fetch API, Promises, async/await
│   └── people.json                  # Datos de ejemplo
│
└── linkedList/                      # Módulo: Estructuras de Datos
    ├── index.html
    ├── main.js                      # Implementación de Lista Enlazada
    └── images/                      # Recursos gráficos
```

🎯 Objetivo General

Proporcionar una introducción práctica a conceptos fundamentales de JavaScript y estructuras de datos básicas, mediante ejemplos interactivos organizados en módulos temáticos progresivos.

📝 Descripción Detallada

Contenido Principal

1. code.js - Ejemplos de Manipulación DOM

Implementa funciones básicas para interactuar con elementos HTML:

- `llenarLista()` **- Actualiza contenido dinámicamente usando** `innerHTML`
- `saludar()` **- Captura entrada de usuario y manipula el DOM**
- `mostrarParrafos()` **- Transforma elementos en encabezados** `<li><h4>`
- Gestión de arrays - Almacena nombres de alumnos en variable global

2. Módulo: introducción/funtions

Tema: Funciones en JavaScript

Conceptos cubiertos:

- Declaración de funciones
- Parámetros y valores por defecto
- Valores de retorno
- Cálculos matemáticos (perímetro de círculo)

Ejemplo clave:

```
function circlePerimeter(PI, diameter) {
    return PI * diameter;
}
```

3. Módulo: introducción/variable

Tema: Declaración y ámbito de variables

Conceptos cubiertos:

- Variables con y `letconst`
- **Tipos de datos básicos**
- **Constantes matemáticas**
- **Operaciones aritméticas**

4. Módulo: json

Tema: Consumo y manejo de datos JSON

Tecnologías abordadas:

- Fetch API - Obtener datos remotos
- Promises - Manejo asincrónico
- async/await - Sintaxis moderna para código asincrónico
- IIFE (Expresión de Función Invocada Inmediatamente)
- Parsing de JSON - Conversión de texto a objetos JavaScript

Ejemplo avanzado:

```
(async function () {
    const dataResult = await
fetch("https://jsonplaceholder.typicode.com/users");
    const jsonResult = await dataResult.json();
})();
```

Datos de ejemplo: contiene información de personas con atributos como nombre, email y habilidades.`people.json`

5. Módulo: linkedList

Tema: Estructuras de datos - Lista Enlazada

Implementación completa con:

- Clase `Node` **- Nodo individual con valor y referencia al siguiente**
- **Clase** `LinkedList` **- Estructura completa con:**
  - `add(value)` **- Elementos insertares**
  - `delete(value)` **- Eliminar elementos**
  - `show()` **- Mostrar contenido en consola**
  - `size()` **- Contar elementos**

Esta es la primera introducción a estructuras de datos complejas en el curso.

🔍 Análisis de Contenido

Fortalezas

✅ Progresión clara - Comienza con lo básico (variables, funciones) y avanza a estructuras de datos

✅ Enfoque práctico - Cada tema tiene un ejemplo HTML para visualizar resultados

✅ Cobertura de paradigmas - Incluye programación imperativa y

orientada a objetos (clases)

✅ Tecnologías modernas - Introduce Fetch API, Promises y async/await

✅ Modularización - Temas separados en carpetas para facilitar la navegación

Áreas de mejora

⚠️ Falta documentación detallada dentro de los scripts

⚠️ No hay instrucciones claras sobre cómo ejecutar o visualizar cada demo

⚠️ El archivo podría beneficiarse de comentarios explicativos

⚠️ Faltan pruebas unitarias o ejemplos de verificación `code.js`

Nivel de dificultad

- Introducción: Muy accesible para principiantes
- JSON: Requiere comprensión de conceptos asincróno
- LinkedList: Demanda pensamiento abstracto sobre referencias y punteros

Temas cubiertos

Tema	Nivel	Módulo	Estado
Variables	**Básico**	**introducción/variable**	✓
Funciones	**Básico**	**Introducción/Funciones**	✓
Manipulación del DOM	**Intermedio**	**code.js, *.html**	✓
JSON	**Intermedio**	**json/**	✓
Asincronía	**Avanzado**	**JSON/main.js**	✓
Estructuras de Datos	**Avanzado**	**linkedList/**	✓

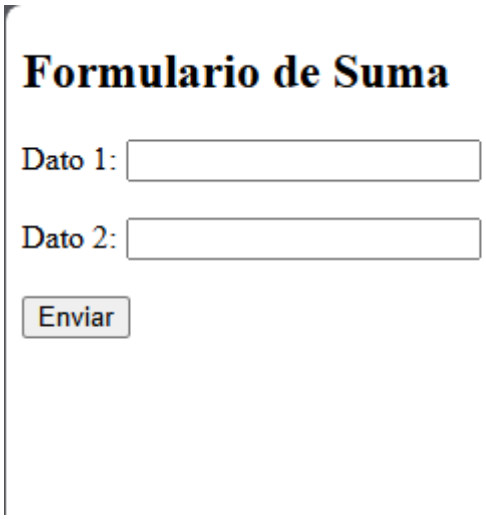
💡 Recomendaciones de Uso

1. Para estudiantes nuevos: Comenzar por y `introduccion/variableintroduccion/funtions`
2. Para visualizar ejemplos: Abrir los archivos en un navegador `.html`

3. Para entender la lógica: Revisar los scripts con la consola abierta `.js`
4. Para avanzados: Estudiar y experimentar con métodos adicionales`linkedList/main.js`
5. Para práctica: Modificar para consumir diferentes APIs`json/main.js`

📌 Conclusión

Semana 5 representa un punto de transición en el aprendizaje de estructuras de datos, combinando fundamentos de JavaScript (variables, funciones) con introducción a conceptos avanzados (asincronía, estructuras de datos). Es ideal para estudiantes que ya comprenden lo básico y buscan fortalecer su entendimiento de patrones más complejos.



Descripción Completa - Semana 5

🌐 Lenguajes Utilizados

- **JavaScript** - Lógica de programación y funcionalidad interactiva
- **HTML** - Estructura y visualización de ejemplos
- **CSS** - Estilos básicos en módulos específicos
- **JSON** - Formato de datos para ejemplos

📁 Estructura de Archivos

```
Semana5/
├── code.js                    # Script principal con ejemplos de manipulación DOM
├── index.html                 # Página raíz de la semana
├── inner.html                 # Página auxiliar/interna
├── CONTENIDO.md               # Resumen de contenido
│
├── introduccion/              # Módulo introductorio
│   ├── funtions/              # Tema: Funciones
│   │   ├── index.html
│   │   └── main.js
│   │
│   └── variable/              # Tema: Variables
│       ├── index.html
│       ├── main.css
│       └── main.js
│
├── json/                      # Módulo: Manejo de JSON
│   ├── index.html
│   ├── main.js                # Fetch API, Promises, async/await
│   └── people.json            # Datos de ejemplo
│
└── linkedList/                # Módulo: Estructuras de Datos
    ├── index.html
    ├── main.js                # Implementación de Lista Enlazada
    └── images/                # Recursos gráficos
```

🎯 Objetivo General

Proporcionar una **introducción práctica** a conceptos fundamentales de JavaScript y estructuras de datos básicas, mediante ejemplos interactivos organizados en módulos temáticos progresivos.

📝 Descripción Detallada

Contenido Principal

1. **code.js** - Ejemplos de Manipulación DOM
Implementa funciones básicas para interactuar con elementos HTML:
- **`llenarLista()`** - Actualiza contenido dinámicamente usando `innerHTML`
- **`saludar()`** - Captura entrada de usuario y manipula el DOM
- **`mostrarParrafos()`** - Transforma elementos `<li>` en encabezados `<h4>`
- **Gestión de arrays** - Almacena nombres de alumnos en variable global

2. **Módulo: introduccion/funtions**
Tema: Funciones en JavaScript

Conceptos cubiertos:
- Declaración de funciones
- Parámetros y valores por defecto
- Valores de retorno
- Cálculos matemáticos (perímetro de círculo)

Ejemplo clave:
```javascript
function circlePerimeter(PI, diameter) {
    return PI * diameter;
}
```

3. **Módulo: introduccion/variable**
Tema: Declaración y ámbito de variables

Conceptos cubiertos:
- Variables con `let` y `const`

- Tipos de datos básicos
- Constantes matemáticas
- Operaciones aritméticas

4. **Módulo: json**
Tema: Consumo y manejo de datos JSON

Tecnologías abordadas:
- **Fetch API** - Obtener datos remotos
- **Promises** - Manejo asincrónico
- **async/await** - Sintaxis moderna para código asincrónico
- **IIFE** (Immediately Invoked Function Expression)
- **Parsing de JSON** - Conversión de texto a objetos JavaScript

Ejemplo avanzado:
```javascript
(async function () {
const dataResult = await
fetch("https://jsonplaceholder.typicode.com/users");
const jsonResult = await dataResult.json();
})();
```

Datos de ejemplo: `people.json` contiene información de personas con atributos como nombre, email y habilidades.

5. **Módulo: linkedList**
Tema: Estructuras de datos - Lista Enlazada

Implementación completa con:
- **Clase `Node`** - Nodo individual con valor y referencia al siguiente
- **Clase `LinkedList`** - Estructura completa con:
  - `add(value)` - Insertar elementos
  - `delete(value)` - Eliminar elementos
  - `show()` - Mostrar contenido en consola
  - `size()` - Contar elementos

Esta es la primera introducción a estructuras de datos complejas en el curso.

🔍 Análisis de Contenido

Fortalezas

✅ **Progresión clara** - Comienza con lo básico (variables, funciones) y avanza a estructuras de datos

✅ **Enfoque práctico** - Cada tema tiene un ejemplo HTML para visualizar resultados

✅ **Cobertura de paradigmas** - Incluye programación imperativa y orientada a objetos (clases)

✅ **Tecnologías modernas** - Introduce Fetch API, Promises y async/await

✅ **Modularización** - Temas separados en carpetas para facilitar navegación

Áreas de mejora

⚠️ Falta documentación detallada dentro de los scripts

⚠️ No hay instrucciones claras sobre cómo ejecutar o visualizar cada demo

⚠️ El archivo `code.js` podría beneficiarse de comentarios explicativos

⚠️ Faltan pruebas unitarias o ejemplos de verificación

Nivel de dificultad

- **Introducción:** Muy accesible para principiantes
- **JSON:** Requiere comprensión de conceptos asincróno
- **LinkedList:** Demanda pensamiento abstracto sobre referencias y punteros

Temas cubiertos

Tema	Nivel	Módulo	Estado
Variables	Básico	introduccion/variable	✓
Funciones	Básico	introduccion/funtions	✓
DOM Manipulation	Intermedio	code.js, *.html	✓
JSON	Intermedio	json/	✓
Asincronía	Avanzado	json/main.js	✓
Estructuras de Datos	Avanzado	linkedList/	✓

💡 Recomendaciones de Uso

1. **Para estudiantes nuevos:** Comenzar por `introduccion/variable` y `introduccion/funtions`
2. **Para visualizar ejemplos:** Abrir los archivos `.html` en un navegador
3. **Para entender la lógica:** Revisar los scripts `.js` con la consola abierta
4. **Para avanzados:** Estudiar `linkedList/main.js` y experimentar con métodos adicionales
5. **Para práctica:** Modificar `json/main.js` para consumir diferentes APIs

---

📌 Conclusión

Semana 5 representa un punto de transición en el aprendizaje de estructuras de datos, combinando fundamentos de JavaScript (variables, funciones) con introducción a conceptos avanzados (asincronía, estructuras de datos). Es ideal para estudiantes que ya comprenden lo básico y buscan fortalecer su entendimiento de patrones más complejos.

Documento generado: 14 de Diciembre de 2025

SEMANA 6

Contenido de la carpeta Semana6

Lenguajes

- HTML
- JavaScript
- CSS

Estructura de archivos (resumen)

- `code.js` — código principal de ejemplos/prácticas.
- `CONTENIDO.md` — índice/guía (existente en la carpeta).
- `index.html` — página de entrada para la semana.
- `README.md` — información general del directorio.
- `RESUMEN_SEMANA6.md` — resumen de la semana.
- `SEMANA6_DESCRIPCION.md` — descripción detallada (si existe).
- `map/`
  - `index.html`
  - `main.js`
- `poo/`
  - `clases/`
    - `indesx.html` (posible error tipográfico)
    - `main.css`
    - `main.js`
  - `reference/`
    - `index.html`
    - `main.js`
- `queue/`
  - `index.html`
  - `main.js`

Nota: la carpeta contiene demos y ejemplos organizados por tema (map, poo, queue).

Objetivo

Recopilar y organizar ejercicios y ejemplos prácticos relacionados con estructuras y técnicas de programación (uso de mapas, programación orientada a objetos y colas) para la semana 6 del curso.

Descripción

La carpeta agrupa archivos de demostración y materiales didácticos:

- Páginas HTML para ver ejemplos en el navegador.
- Archivos JavaScript con implementaciones y ejercicios (, , , etc.).`code.jsmap/main.jsqueue/main.js`
- Recursos CSS mínimos para estilos de las demos.
- Subcarpetas temáticas (, ,) que contienen ejemplos específicos y recursos asociados.`mappooqueue`

Breve análisis de contenido

- Calidad: La carpeta está bien segmentada por temas; facilita navegar entre demos.
- Complejidad: Predominan ejemplos de nivel introductorio/medio (ejercicios y pequeños demos).
- Recomendaciones: Unificar nombres (por ejemplo corregir a) y centralizar un con enlaces directos a cada demo para mejorar accesibilidad.`indesx.htmlindex.htmlREADME.md`

¿De qué trata la Semana 6?

Durante esta semana se estudian y practican:

- **Estructuras de datos**
    - Uso de **Map** para almacenar y gestionar pares clave-valor.
    - Implementación de **colas (Queue)** para manejar datos en orden FIFO.

- **Programación Orientada a Objetos (POO)**
    - Creación y uso de **clases**.
    - Organización del código con métodos y atributos.
    - Ejemplos prácticos para entender la POO en JavaScript.
- **Integración Web**
    - Uso de **HTML** para mostrar los ejemplos en el navegador.
    - **JavaScript** como lenguaje principal para la lógica.
    - **CSS** básico para dar estilo a las demostraciones.

Propósito de la carpeta Semana 6

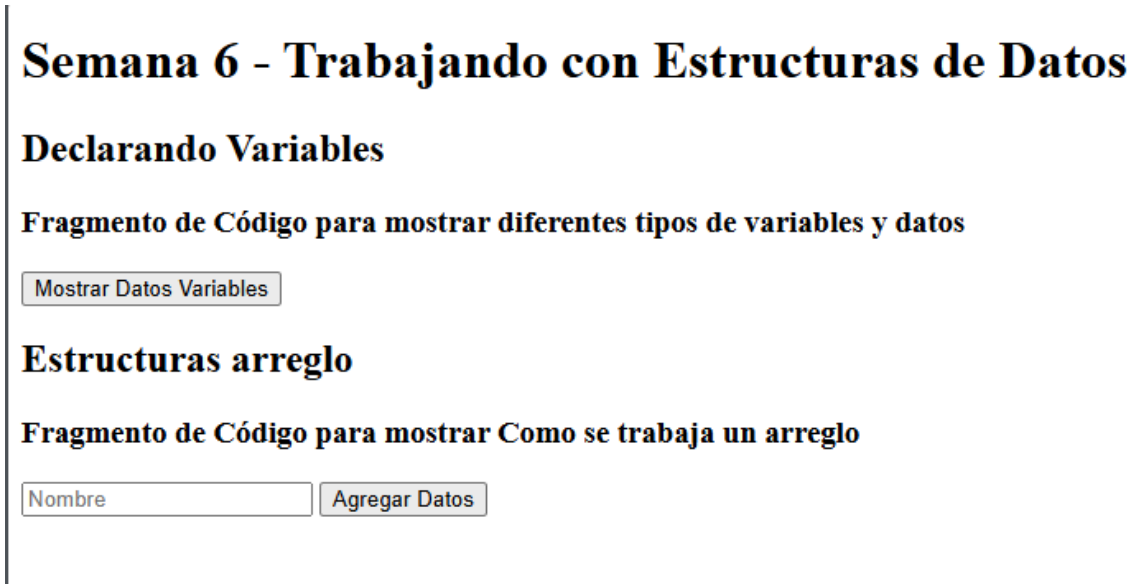
El objetivo es **reunir ejercicios, demos y material didáctico** que ayuden a:

- Comprender cómo funcionan las estructuras de datos en JavaScript.
- Aplicar la POO en proyectos sencillos.
- Visualizar el funcionamiento del código directamente en el navegador.

Organización del contenido

- **Archivos principales**
    - `index.html`: punto de entrada de la semana.
    - `code.js`: ejemplos y prácticas generales.
    - `README.md`, `CONTENIDO.md`, `RESUMEN_SEMANA6.md`: documentación y guía.

- **Subcarpetas temáticas**
    - `map/`: ejemplos del uso de Map.
    - `poo/`: prácticas de programación orientada a objetos.
    - `queue/`: implementación de colas.



Contenido de la carpeta Semana6

Lenguajes

- HTML
- JavaScript
- CSS

Estructura de archivos (resumen)

- `code.js` — código principal de ejemplos/prácticas.
- `CONTENIDO.md` — índice/guía (existente en la carpeta).
- `index.html` — página de entrada para la semana.
- `README.md` — información general del directorio.
- `RESUMEN_SEMANA6.md` — resumen de la semana.
- `SEMANA6_DESCRIPCION.md` — descripción detallada (si existe).
- `map/`
  - `index.html`
  - `main.js`
- `poo/`
  - `clases/`
    - `indesx.html` (posible typo)
    - `main.css`
    - `main.js`
  - `reference/`
    - `index.html`
    - `main.js`

- `queue/`
  - `index.html`
  - `main.js`

> Nota: la carpeta contiene demos y ejemplos organizados por tema (map, poo, queue).

Objetivo

Recopilar y organizar ejercicios y ejemplos prácticos relacionados con estructuras y técnicas de programación (uso de mapas, programación orientada a objetos y colas) para la semana 6 del curso.

Descripción

La carpeta agrupa archivos de demostración y materiales didácticos:

- Páginas HTML para ver ejemplos en el navegador.
- Archivos JavaScript con implementaciones y ejercicios (`code.js`, `map/main.js`, `queue/main.js`, etc.).
- Recursos CSS mínimos para estilos de las demos.
- Subcarpetas temáticas (`map`, `poo`, `queue`) que contienen ejemplos específicos y recursos asociados.

Breve análisis de contenido

- **Calidad:** La carpeta está bien segmentada por temas; facilita navegar entre demos.
- **Complejidad:** Predominan ejemplos de nivel introductorio/medio (ejercicios y pequeños demos).

```
- **Recomendaciones:** Unificar nombres (por ejemplo corregir `indesx.html` a `index.html`) y centralizar un `README.md` con enlaces directos a cada demo para mejorar accesibilidad.



---

Generado automáticamente: resumen de la carpeta `Semana6`.
```

Semana 7 — Análisis Detallado de Contenidos

📋 Lenguaje

- Principal: JavaScript (ES6+)
- Complementario: HTML5 y CSS para interfaz gráfica de demostración
- Enfoque: Implementación de estructuras de datos con ejemplos ejecutables en el navegador

---

📁 Estructura de Archivos

```
Semana7/
├── CONTENIDO.md                    # Resumen oficial de contenidos
├── index.html                      # Página principal de demostración
├── code.js                         # Ejemplos generales de variables y estructuras
├── hashTables/
│   └── hashTables.js               # Implementación de tabla hash con manejo de colisiones
├── set/
│   ├── index.html                  # Interfaz de demostración para Set
│   └── main.js                     # Ejemplos prácticos de conjuntos (Set)
└── stack/
    ├── index.html                  # Interfaz de demostración para Stack
```

```
    └── main.js                    # Implementación de pila (Stack LIFO)
```

🎯 Objetivo

Proporcionar una comprensión práctica de tres estructuras de datos fundamentales:

1. **Hash Table - Almacenamiento clave-valor con resolución de colisiones**
2. **Set (Conjunto) - Colección de valores únicos**
3. **Stack (Pila) - Estructura LIFO (Último en entrar, primero en salir)**

Cada estructura incluye:

- **Implementación en JavaScript**
- **Ejemplos de uso en consola**
- **Interfaz HTML para visualización interactiva**
- **Casos de uso prácticos y comunes**

📖 Descripción Detallada

Archivo Principal: `index.html` y `code.js`

Punto de entrada de la semana con tres secciones demostrativas:

1. **Declaración de Variables**
    - **Tipos primitivos: String, Number, Boolean, Number (decimal)**
    - **Función que despliega tipos con `mostrarVariables()` `typeof`**
2. Estructuras de Arreglos (Arrays)
    - Operaciones CRUD básicas: agregar, editar, eliminar elementos
    - Uso de y para iteración`push()` `forEach()`
    - **Interfaz interactiva con botones**
3. **Estructuras de Datos Personalizadas**
    - **Objetos como estructuras:** `{nombre, edad, telefono, soltero, estatura}`

- **Arrays de objetos para almacenar múltiples registros**
- **Manipulación de datos con validación de tipos**

Subcarpeta: `hashTables/`

Archivo: `hashTables.js`

Implementación de una clase Hash Table con:

- **Constructor: Inicializa un array de tamaño especificado**
- **hashMethod(key): Función hash que convierte claves en índices**
  - **Algoritmo: suma ponderada de códigos de caracteres módulo tamaño**
  - **Propósito: distribuir claves uniformemente**
- **set(key, value): Agrega pares clave-valor**
  - **Maneja colisiones usando "encadenamiento separado" (arrays anidados)**
- **get(key): Recupera valores por clave**
  - **Busca en el bucket correspondiente**
  - **Retorna si no existe`undefined`**
- delete(key): Elimina pares clave-valor
  - Limpia el bucket después de eliminar

Caso de uso: Búsquedas rápidas O(1) promedio, diccionarios, mapeos

---

Subcarpeta: `set/`

Archivo: `main.js`

Utiliza el Set nativo de JavaScript con ejemplos de:

- **add(): Agregar elementos (automáticamente evita duplicados)**
  - **Demuestra que no se duplica aunque se agregue dos veces`1`**
  - **Los objetos se comparan por referencia, no por valor**
- **delete(): Eliminar elementos específicos**
  - **Elimina por referencia exacta**
- **has(): Verificar existencia**

- **Muestra diferencia entre búsqueda por valor vs. por referencia**
- **`{name: "Juan"}` (nuevo objeto) no es igual a uno agregado previamente**

Concepto clave: Los Set garantizan unicidad pero comparan objetos por identidad (===), no por igualdad de contenido

Caso de uso: Eliminar duplicados, búsquedas rápidas, relaciones de pertenencia

---

Subcarpeta: `acumular/`

Archivo: **`main.js`**

Implementación personalizada de Stack (Pila) con:

- **Propiedad privada (encapsulación ES2022)`#items = []`**
- push(item): Agrega elemento al tope
- pop(): Extrae y retorna el elemento del tope (LIFO)
- Peek(): Visualiza el tope sin remover
- size(): Retorna cantidad de elementos
- isEmpty(): Verifica si está vacía
- getItems(): Copia del array (operador de dispersión `[...]`)

Ejemplo de ejecución:

`Stack vacío → isEmpty: true`

`push("Pedro", "Juan", "Héctor")`

`peek() → "Héctor"`

`pop() → retorna "Héctor" (se remueve)`

Caso de uso: Navegación (undo/redo), evaluación de expresiones, búsqueda en profundidad (DFS)

🔍 Breve Análisis de Contenido

Fortalezas

✅ Didáctico: Las implementaciones son simples y fáciles de entender

✅ Práctico: Cada estructura tiene ejemplos ejecutables en consola y en navegador

✅ Modular: Cada estructura en su propia carpeta facilita el aprendizaje progresivo

✅ Variado: Combina estructuras nativas de JS (Set) con implementaciones propias (Hash Table, Stack)

Contenido Técnico

- Hash Table: Demuestra colisiones y resolución mediante chaining
- Set: Aprovecha la colección nativa con énfasis en unicidad y búsqueda
- Stack: Implementación desde cero con encapsulación moderna (campos privados)

Áreas de Mejora

⚠️ Faltan comentarios explicativos en el código

⚠️ No hay casos de prueba (testing) formales

⚠️ Las páginas HTML son mínimas (sin estilos CSS)

⚠️ Sería útil incluir análisis de complejidad (O(1), O(n), etc.)

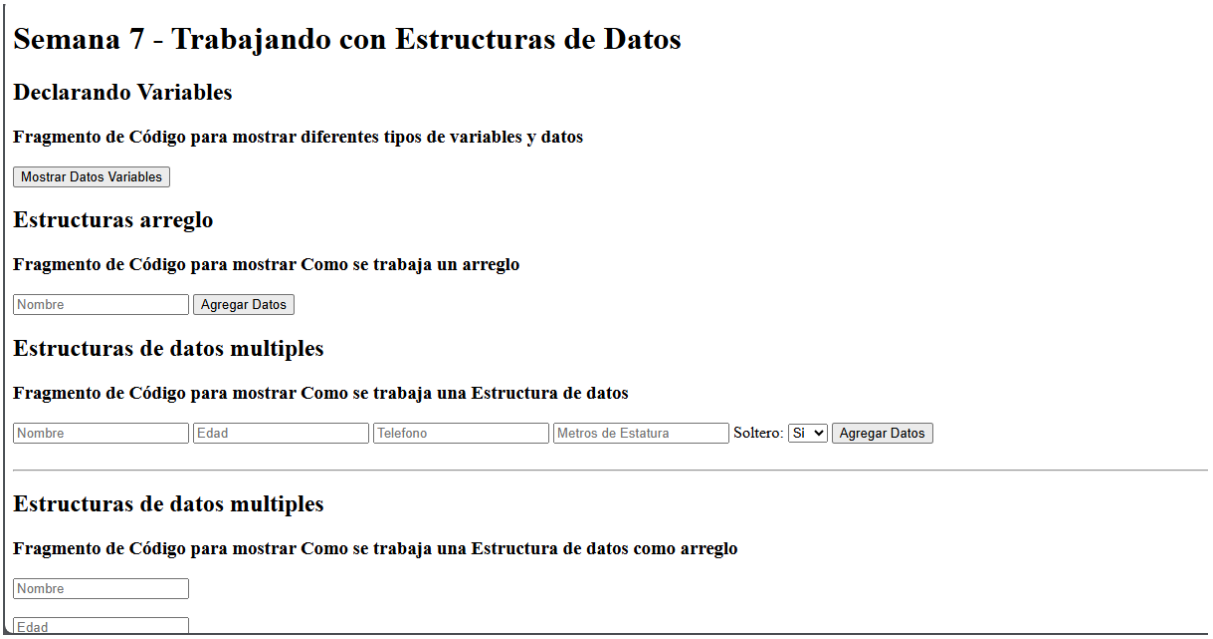
Nivel Educativo

- **Dirigido a estudiantes principiantes-intermedios**
- **Asume familiaridad con: arrays, objetos, bucles, funciones**
- **Introduce conceptos fundamentales de estructuras sin excesiva complejidad**

🚀 Flujo de Aprendizaje Recomendado

1. Comienza por - comprende variables básicas y arrays`index.html`
2. Continúa con - estructura simple con operaciones LIFO`stack/main.js`
3. Luego - uso de colecciones con garantía de unicidad`set/main.js`

4. Finalmente - implementación compleja con resolución de colisiones`hashTables/hashTables.js`



Semana 7 — Análisis Detallado de Contenidos

📋 Lenguaje

- **Principal**: JavaScript (ES6+)
- **Complementario**: HTML5 y CSS para interfaz gráfica de demostración
- **Enfoque**: Implementación de estructuras de datos con ejemplos ejecutables en el navegador

---

📁 Estructura de Archivos

```
Semana7/
├── CONTENIDO.md                    # Resumen oficial de contenidos
├── index.html                      # Página principal de demostración
├── code.js                         # Ejemplos generales de variables y estructuras
├── hashTables/
│   └── hashTables.js               # Implementación de tabla hash con manejo de colisiones
├── set/
│   ├── index.html                  # Interfaz de demostración para Set
│   └── main.js                     # Ejemplos prácticos de conjuntos (Set)
└── stack/
    ├── index.html                  # Interfaz de demostración para Stack
    └── main.js                     # Implementación de pila (Stack LIFO)
```

🎯 Objetivo

Proporcionar una comprensión práctica de tres **estructuras de datos fundamentales**:

1. **Hash Table** - Almacenamiento clave-valor con resolución de colisiones
2. **Set (Conjunto)** - Colección de valores únicos

3. **Stack (Pila)** - Estructura LIFO (Last In, First Out)

Cada estructura incluye:
- Implementación en JavaScript
- Ejemplos de uso en consola
- Interfaz HTML para visualización interactiva
- Casos de uso prácticos y comunes

📖 Descripción Detallada

Archivo Principal: `index.html` y `code.js`
Punto de entrada de la semana con tres secciones demostrativas:

1. **Declaración de Variables**
- Tipos primitivos: String, Number, Boolean, Number (decimal)
- Función `mostrarVariables()` que despliega tipos con `typeof`

2. **Estructuras de Arreglos (Arrays)**
- Operaciones CRUD básicas: agregar, editar, eliminar elementos
- Uso de `push()` y `forEach()` para iteración
- Interfaz interactiva con botones

3. **Estructuras de Datos Personalizadas**
- Objetos como estructuras: `{nombre, edad, telefono, soltero, estatura}`
- Arrays de objetos para almacenar múltiples registros
- Manipulación de datos con validación de tipos

Subcarpeta: `hashTables/`
Archivo: `hashTables.js`

Implementación de una clase **Hash Table** con:
- **Constructor**: Inicializa un array de tamaño especificado
- **hashMethod(key)**: Función hash que convierte claves en índices
- Algoritmo: suma ponderada de códigos de caracteres módulo tamaño
- Propósito: distribuir claves uniformemente
- **set(key, value)**: Agrega pares clave-valor
- Maneja colisiones usando "separate chaining" (arrays anidados)
- **get(key)**: Recupera valores por clave
- Busca en el bucket correspondiente
- Retorna `undefined` si no existe

- **delete(key)**: Elimina pares clave-valor
- Limpia el bucket después de eliminar

Caso de uso: Búsquedas rápidas O(1) promedio, diccionarios, mapeos

---

Subcarpeta: `set/`
Archivo: `main.js`

Utiliza el **Set nativo de JavaScript** con ejemplos de:
- **add()**: Agregar elementos (automáticamente evita duplicados)
- Demuestra que `1` no se duplica aunque se agregue dos veces
- Los objetos se comparan por referencia, no por valor
- **delete()**: Eliminar elementos específicos
- Elimina por referencia exacta
- **has()**: Verificar existencia
- Muestra diferencia entre búsqueda por valor vs. por referencia
- `{name: "Juan"}` (nuevo objeto) no es igual a uno agregado previamente

Concepto clave: Los Set garantizan unicidad pero comparan objetos por identidad (===), no por igualdad de contenido

Caso de uso: Eliminar duplicados, búsquedas rápidas, relaciones de pertenencia

---

Subcarpeta: `stack/`
Archivo: `main.js`

Implementación personalizada de **Stack (Pila)** con:
- **Propiedad privada** `#items = []` (encapsulación ES2022)
- **push(item)**: Agrega elemento al tope
- **pop()**: Extrae y retorna el elemento del tope (LIFO)
- **peek()**: Visualiza el tope sin remover
- **size()**: Retorna cantidad de elementos
- **isEmpty()**: Verifica si está vacía
- **getItems()**: Copia del array (spread operator `[...]`)

Ejemplo de ejecución:
```
```

Stack vacío → isEmpty: true
push("Pedro", "Juan", "Héctor")
peek() → "Héctor"
pop() → retorna "Héctor" (se remueve)
```

**Caso de uso**: Navegación (undo/redo), evaluación de expresiones, búsqueda en profundidad (DFS)

---

## 🔍 Breve Análisis de Contenido

### **Fortalezas**
✅ **Didáctico**: Las implementaciones son simples y fáciles de entender
✅ **Práctico**: Cada estructura tiene ejemplos ejecutables en consola y en navegador
✅ **Modular**: Cada estructura en su propia carpeta facilita el aprendizaje progresivo
✅ **Variado**: Combina estructuras nativas de JS (Set) con implementaciones propias (Hash Table, Stack)

### **Contenido Técnico**
- **Hash Table**: Demuestra colisiones y resolución mediante chaining
- **Set**: Aprovecha la colección nativa con énfasis en unicidad y búsqueda
- **Stack**: Implementación desde cero con encapsulación moderna (campos privados)

### **Áreas de Mejora**
⚠️ Faltan comentarios explicativos en el código
⚠️ No hay casos de prueba (testing) formales
⚠️ Las páginas HTML son mínimas (sin estilos CSS)
⚠️ Sería útil incluir análisis de complejidad (O(1), O(n), etc.)

### **Nivel Educativo**
- Dirigido a estudiantes **principiantes-intermedios**
- Asume familiaridad con: arrays, objetos, bucles, funciones
- Introduce conceptos fundamentales de estructuras sin excesiva complejidad

---
```

🚀 Flujo de Aprendizaje Recomendado

1. **Comienza por** `index.html` - comprende variables básicas y arrays
2. **Continúa con** `stack/main.js` - estructura simple con operaciones LIFO
3. **Luego** `set/main.js` - uso de colecciones con garantía de unicidad
4. **Finalmente** `hashTables/hashTables.js` - implementación compleja con resolución de colisiones

---

Generado: Análisis automático de estructura de datos - Semana 7

SEMANA 8

Semana 8 - Estructuras de Datos: Listas Enlazadas

Lenguaje

JavaScript (ES6+)

Estructura de Archivos

```
Semana8/
└── linkedlist/
    ├── addNode.js        (Ejemplos básicos de nodos)
    ├── singly.js         (Implementación de Singly Linked List)
    └── doublyList.js     (Implementación de Doubly Linked List)
```

Objetivo

Comprender e implementar dos de las estructuras de datos más fundamentales en programación:

- Listas Enlazadas Simples (Listas Enlazadas Simples): Nodos conectados en una sola dirección
- Listas Doblemente Enlazadas (Listas Enlazadas Dobles): Nodos conectados en ambas direcciones con referencias previas y siguientes

---

Descripción General

Esta semana se enfoca en el estudio de las listas enlazadas, estructuras de datos dinámicas donde cada elemento (nodo) contiene un valor y referencias a otros nodos. A diferencia de los arrays, las listas enlazadas no requieren memoria contigua y permiten inserciones y eliminaciones eficientes.

Conceptos Clave:

- Nodo: Unidad básica que contiene un valor y referencias a otros nodos
- Cabeza (Cabeza): El primer nodo de la lista
- Cola (Tail): El último nodo de la lista
- Siguiente (Next): Referencia al próximo nodo
- Anterior (Prev): Referencia al nodo anterior (solo en listas dobles)

Análisis de Contenido

1. addNode.js

```
// Ejemplo básico de estructura de nodos
let singlyLinked = {
  head: {
    value: 1,
    next: { /* ... */ }
  }
};
```

Contenido:

- Demostración básica de cómo representar una lista enlazada simple usando objetos
- Estructura de un nodo con y `valuenext`
- Función para iterar sobre la lista`recorrer()`
- Referencia a para marcar el final de la lista`null`

Enfoque: Conceptos fundamentales e introducción a la estructura de datos

---

2. singly.js

```
class Nodes {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class myDoublyLinkedList {
```

```
  constructor(value) {
    this.head = { value: value, next: null };
    this.tail = this.head;
    this.length = 1;
  }
}
```

Contenido:

- Implementación de clases para crear nodos
- Estructura básica de una lista enlazada con cabeza y cola
- Propiedades: , , `headtaillength`

Enfoque: Programación orientada a objetos (POO) con clases de ES6

---

3. doublyList.js ⭐ (Implementación Completa)

```
class myDoublyLinkedList {
  constructor(value) { /* ... */ }

  add(value)            // Agregar nodo al final
  insert(index, value) // Insertar nodo en posición específica
  getTheIndex(index)    // Obtener nodo por índice
  remove(index)         // Eliminar nodo por índice
  search()              // Recorrer e imprimir la lista
}
```

Métodos Implementados:

Método	Descripción	Complejidad
`add(value)`	Agrega un nodo al final de la lista	O(1)
`insert(index, value)`	Inserta un nodo en una posición específica	O(n)
`getTheIndex(index)`	Obtiene el nodo en una posición determinada	O(n)

`remove(index)`	Elimina un nodo de la lista	O(n)
`search()`	Recorre la lista imprimiendo valores	O(n)

Características:

- Nodos con referencias bidireccionales (y `next` `prev`)
- Mantenimiento de cabeza y cola para acceso rápido
- Contador de longitud dinámica
- Ejemplo práctico de uso al final del archivo

Resumen de Aprendizajes

✅ Listas Enlazadas Individuales: Estructuras unidireccionales, eficientes en memoria

✅ Doblemente Listas Enlazadas: Estructuras bidireccionales, flexibles para recorridos

✅ Operaciones Fundamentales: Add, Insert, Remove, Search

✅ Ventajas sobre Arrays: Inserción/eliminación O(1) sin reorganización

✅ Desventajas: Acceso aleatorio O(n) y mayor consumo de memoria para referencias

---

Uso

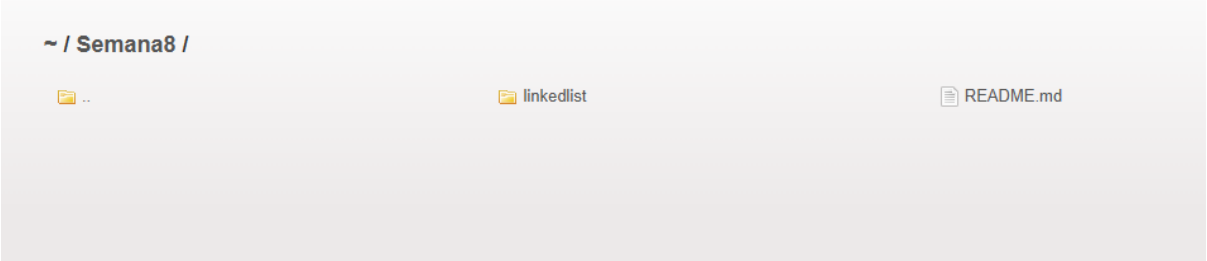
Para ejecutar cualquiera de los archivos:

```
node linkedlist/doublyList.js
```

Salida esperada:

```
Nodo numero cero
Primer nodo
Segundo nodo
Tercer nodo
Cuarto nodo
```

Nivel: Intermedio | Tema: Estructuras de Datos | Enfoque: Implementación práctica en JavaScript



Semana 8 - Estructuras de Datos: Listas Enlazadas

Lenguaje

JavaScript (ES6+)

Estructura de Archivos

```
Semana8/
└── linkedlist/
    ├── addNode.js       (Ejemplos básicos de nodos)
    ├── singly.js        (Implementación de Singly Linked List)
    └── doublyList.js    (Implementación de Doubly Linked List)
```

Objetivo

Comprender e implementar dos de las estructuras de datos más fundamentales en programación:

- **Singly Linked Lists (Listas Enlazadas Simples)**: Nodos conectados en una sola dirección
- **Doubly Linked Lists (Listas Enlazadas Dobles)**: Nodos conectados en ambas direcciones con referencias previas y siguientes

---

Descripción General

Esta semana se enfoca en el estudio de las **listas enlazadas**, estructuras de datos dinámicas donde cada elemento (nodo) contiene un valor y referencias a otros nodos. A diferencia de los arrays, las listas enlazadas no requieren memoria contigua y permiten inserciones y eliminaciones eficientes.

Conceptos Clave:
- **Nodo**: Unidad básica que contiene un valor y referencias a otros nodos
- **Cabeza (Head)**: El primer nodo de la lista
- **Cola (Tail)**: El último nodo de la lista
- **Siguiente (Next)**: Referencia al próximo nodo
- **Anterior (Prev)**: Referencia al nodo anterior (solo en listas dobles)

Análisis de Contenido

1. **addNode.js**
```javascript
// Ejemplo básico de estructura de nodos
let singlyLinked = {
    head: {
    value: 1,
  next: { /* ... */ }
    }
  };
```

Contenido:
- Demostración básica de cómo representar una lista enlazada simple usando objetos
- Estructura de un nodo con `value` y `next`
- Función `recorrer()` para iterar sobre la lista
- Referencia a `null` para marcar el final de la lista

Enfoque: Conceptos fundamentales e introducción a la estructura de datos

---

2. **singly.js**

```javascript
class Nodes {
constructor(value) {
this.value = value;
this.next = null;
}
}

class myDoublyLinkedList {
constructor(value) {
this.head = { value: value, next: null };
this.tail = this.head;
this.length = 1;
}
}
```

Contenido:

- Implementación de clases para crear nodos
- Estructura básica de una lista enlazada con cabeza y cola
- Propiedades: `head`, `tail`, `length`

Enfoque: Programación orientada a objetos (POO) con clases de ES6

---

3. **doublyList.js** ⭐ (Implementación Completa)

```javascript
class myDoublyLinkedList {
constructor(value) { /* ... */ }

add(value)           // Agregar nodo al final
insert(index, value) // Insertar nodo en posición específica
getTheIndex(index)   // Obtener nodo por índice
remove(index)        // Eliminar nodo por índice
search()             // Recorrer e imprimir la lista
}
```

Métodos Implementados:

Método	Descripción	Complejidad

```
| `add(value)` | Agrega un nodo al final de la lista | O(1) |
| `insert(index, value)` | Inserta un nodo en una posición específica | O(n) |
| `getTheIndex(index)` | Obtiene el nodo en una posición determinada | O(n) |
| `remove(index)` | Elimina un nodo de la lista | O(n) |
| `search()` | Recorre la lista imprimiendo valores | O(n) |

**Características:**
- Nodos con referencias bidireccionales (`next` y `prev`)
- Mantenimiento de cabeza y cola para acceso rápido
- Contador de longitud dinámico
- Ejemplo práctico de uso al final del archivo

---

## Resumen de Aprendizajes

✅ **Singly Linked Lists**: Estructuras unidireccionales, eficientes en memoria
✅ **Doubly Linked Lists**: Estructuras bidireccionales, flexibles para recorridos
✅ **Operaciones Fundamentales**: Add, Insert, Remove, Search
✅ **Ventajas sobre Arrays**: Inserción/eliminación O(1) sin reorganización
✅ **Desventajas**: Acceso aleatorio O(n) y mayor consumo de memoria para referencias

---

## Uso

Para ejecutar cualquiera de los archivos:

```bash
node linkedlist/doublyList.js
```

Output esperado:
```
Nodo numero cero
Primer nodo
Segundo nodo
```
```

```
                                Tercer nodo
                                Cuarto nodo
                                    ```

Nivel: Intermedio | **Tema:** Estructuras de Datos | **Enfoque:**
                Implementación práctica en JavaScript
```
```

SEMANA 9

Semana 9 - Estructuras de Datos en JavaScript

📋 Lenguaje Utilizado

JavaScript (ES6+) - Implementación de estructuras de datos clásicas utilizando clases modernas de JavaScript.

🗂️ Estructura de Archivos

```
Semana9/
├── arrays.js                    # Implementación personalizada de Array con clase MyArray
│   ├── stack.js                 # Pila (Stack) - estructura LIFO
│   ├── queue.js                 # Cola (Queue) - estructura FIFO
├── singly_linkedList.js         # Lista enlazada simple (Singly Linked List)
├── doubly_linkedList.js         # Lista enlazada doble (Doubly Linked List)
├── tree.js                      # Árbol binario de búsqueda (Binary Search Tree)
│   ├── hash_table.js            # Tabla hash (Hash Table)
│   ├── graph.js                 # Grafo (Graph)
│   ├── queues/
│   │   └── queues.js            # Implementación adicional de colas
├── README.md                    # Documentación básica del proyecto
└── LICENSE                      # Licencia del proyecto
```

🎯 Objetivo General

El objetivo de Semana 9 es proporcionar implementaciones funcionales y educativas de las estructuras de datos más fundamentales en Ciencias de la Computación, utilizando JavaScript moderno con clases (ES6+). Esta semana consolida el conocimiento de estructuras que ya fueron introducidas en semanas anteriores, ofreciendo código limpio, reutilizable y fácil de entender.

📝 Descripción Detallada

Contenido Principal

Este módulo implementa las siguientes estructuras de datos de forma educativa:

1. Matrices (`arrays.js`)
    - Clase personalizada que replica la funcionalidad de un arreglo`MyArray`
    - Métodos: , , , `get()``push()``pop()``delete()`
    - Demuestra cómo los arrays se construyen internamente usando objetos
2. Pila - Pila (`stack.js`)
    - Estructura LIFO (Último en entrar, primero en salir)
    - Clase con métodos: , , `Stack``push()``pop()``peek()`
    - Implementada con nodos vinculados para mejor rendimiento
3. Cola - Cola (`queue.js`)
    - Estructura FIFO (Primero en entrar, primero en salir)
    - Clase con métodos: , , `Queue``enqueue()``dequeue()``peek()`
    - Nodos con referencias para mantener el orden`next`
4. Lista Enlazada Individual (`singly_linkedList.js`)
    - Lista enlázada unidireccional
    - Clase con métodos de manipulación`MySinglyLinkedList`
    - Cada nodo apunta solo al siguiente (`next`)
    - Operaciones: , , búsqueda, eliminación`append()``prepend()`
5. Lista Doblemente Enlazada (`doubly_linkedList.js`)
    - Lista enlázada bidireccional
    - Clase con navegación en ambas direcciones`MyDoubleLinkedList`
    - Cada nodo tiene referencias y `next``prev`
    - Permite traversal en ambas direcciones
6. Árbol de búsqueda binario (`tree.js`)
    - Árbol binario de búsqueda balanceado
    - Clase para operaciones de búsqueda eficiente`BinarySearchTree`
    - Métodos: , , travesía`insert()``search()`
    - Mantiene orden y permite búsquedas en O(log n)

7. Tabla hash (`hash_table.js`)
    - Tabla hash para almacenamiento clave-valor
    - Implementación con manejo de colisiones (encadenada)
    - Método hash personalizado para distribuir datos
    - Métodos: , `set()` `get()`
8. Gráfico (`graph.js`)
    - Estructura de grafo para representar redes complejas
    - Implementación de grafos dirigidos o no dirigidos
    - Operaciones de travesía y búsqueda
9. Colas Adicionales (`colas/queues.js`)
    - Implementación alternativa o extendida de colas

🔍 Análisis Breve de Contenido

Patrones de Diseño Utilizados

- Clases ES6: Todas las estructuras están implementadas como clases para encapsulación y reutilización
- Nodos: Estructuras que utilizan nodos (class) para enlazar elementos`Node`
- Gestión de Punteros: Uso de referencias para conectar elementos

Características Clave

Estructura	Complejidad de Tiempo	Caso de Uso
Array Personalizado	O(1) acceso, O(n) inserción	Acceso rápido por índice
Stack	O(1) empuje/pop	Deshacer/rehacer, recursión
Cola	O(1) enqueuear/dequeuear	Procesamiento FIFO, colas
Lista Enlazada Individual	O(n) búsqueda, O(1) inserción	Listas dinámicas

Lista Doblemente Enlazada	O(n) búsqueda, O(1) inserción	Navegación bidireccional
Árbol de búsqueda binario	O(log n) búsqueda media	Búsqueda y ordenamiento
Tabla hash	O(1) promedio	Búsqueda de clave-valor rápida
Graph	Varía según implementación	Redes, mapas, recomendaciones

Nivel Educativo

- Dificultad: Intermedia
- Prerequisitos: Conceptos básicos de JavaScript, POO, clases
- Enfoque: Implementación desde cero, sin librerías externas
- Propósito: Entender cómo funcionan internamente estas estructuras

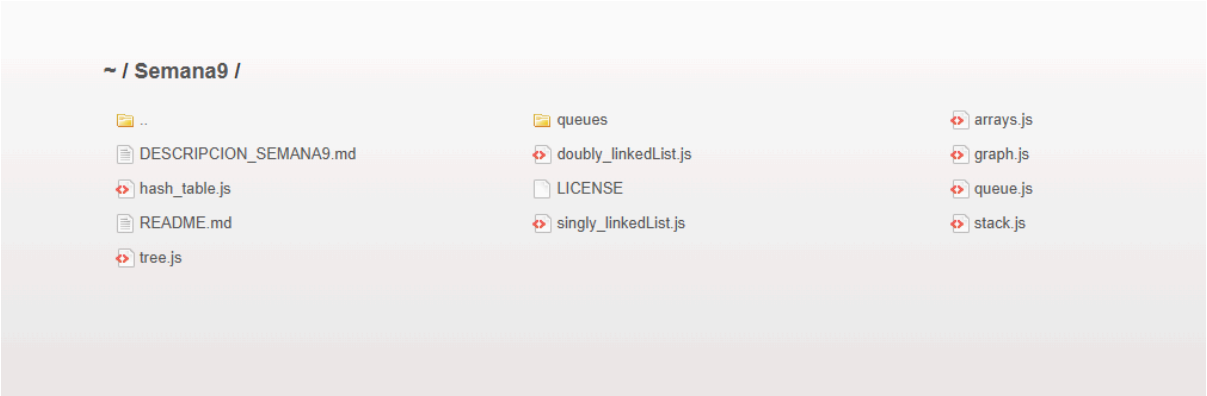
Aplicaciones Prácticas

- Stack: Undo/Redo en aplicaciones, evaluación de expresiones
- Queue: Sistemas de procesamiento, colas de espera, BFS
- Listas enlazadas: Memoria dinámica, listas de reproducción
- BST: Índices de bases de datos, búsquedas eficientes
- Hash Tables: Diccionarios, cachés, búsquedas por clave
- Graphs: Redes sociales, sistemas de navegación, recomendadores

---

💡 Conclusión

Semana 9 presenta una colección completa y bien organizada de implementaciones de estructuras de datos clásicas. Es un recurso excelente para consolidar conocimientos teóricos y ver cómo estas estructuras se implementan en la práctica. Cada implementación está enfocada en claridad y educación, haciendo que sea fácil entender los mecanismos internos de cada estructura.



Semana 9 - Estructuras de Datos en JavaScript

📋 Lenguaje Utilizado

JavaScript (ES6+) - Implementación de estructuras de datos clásicas utilizando clases modernas de JavaScript.

🗂️ Estructura de Archivos

```
Semana9/
├── arrays.js                # Implementación personalizada de Array con clase MyArray
├── stack.js                 # Pila (Stack) - estructura LIFO
├── queue.js                 # Cola (Queue) - estructura FIFO
├── singly_linkedList.js     # Lista enlazada simple (Singly Linked List)
├── doubly_linkedList.js     # Lista enlazada doble (Doubly Linked List)
├── tree.js                  # Árbol binario de búsqueda (Binary Search Tree)
├── hash_table.js            # Tabla hash (Hash Table)
├── graph.js                 # Grafo (Graph)
├── queues/
│   └── queues.js            # Implementación adicional de colas
├── README.md                # Documentación básica del proyecto
└── LICENSE                  # Licencia del proyecto
```

🎯 Objetivo General

El objetivo de Semana 9 es proporcionar implementaciones funcionales y educativas de las **estructuras de datos más fundamentales** en Ciencias de la Computación, utilizando **JavaScript moderno con clases (ES6+)**. Esta semana consolida el conocimiento de estructuras que ya fueron introducidas en semanas anteriores, ofreciendo código limpio, reutilizable y fácil de entender.

📝 Descripción Detallada

Contenido Principal

Este módulo implementa las siguientes estructuras de datos de forma educativa:

1. **Arrays (`arrays.js`)**
   - Clase personalizada `MyArray` que replica la funcionalidad de un arreglo
   - Métodos: `get()`, `push()`, `pop()`, `delete()`
   - Demuestra cómo los arrays se construyen internamente usando objetos

2. **Stack - Pila (`stack.js`)**
   - Estructura LIFO (Last In, First Out)
   - Clase `Stack` con métodos: `push()`, `pop()`, `peek()`
   - Implementada con nodos vinculados para mejor rendimiento

3. **Queue - Cola (`queue.js`)**
   - Estructura FIFO (First In, First Out)
   - Clase `Queue` con métodos: `enqueue()`, `dequeue()`, `peek()`
   - Nodos con referencias `next` para mantener el orden

4. **Singly Linked List (`singly_linkedList.js`)**
   - Lista enlazada unidireccional
   - Clase `MySinglyLinkedList` con métodos de manipulación
   - Cada nodo apunta solo al siguiente (`next`)
   - Operaciones: `append()`, `prepend()`, búsqueda, eliminación

5. **Doubly Linked List (`doubly_linkedList.js`)**
   - Lista enlazada bidireccional
   - Clase `MyDoubleLinkedList` con navegación en ambas direcciones

- Cada nodo tiene referencias `next` y `prev`
- Permite traversal en ambas direcciones

6. **Binary Search Tree (`tree.js`)**
- Árbol binario de búsqueda balanceado
- Clase `BinarySearchTree` para operaciones de búsqueda eficiente
- Métodos: `insert()`, `search()`, traversal
- Mantiene orden y permite búsquedas en O(log n)

7. **Hash Table (`hash_table.js`)**
- Tabla hash para almacenamiento clave-valor
- Implementación con manejo de colisiones (chaining)
- Método hash personalizado para distribuir datos
- Métodos: `set()`, `get()`

8. **Graph (`graph.js`)**
- Estructura de grafo para representar redes complejas
- Implementación de grafos dirigidos o no dirigidos
- Operaciones de traversal y búsqueda

9. **Queues Adicional (`queues/queues.js`)**
- Implementación alternativa o extendida de colas

🔍 Análisis Breve de Contenido

Patrones de Diseño Utilizados

- **Clases ES6**: Todas las estructuras están implementadas como clases para encapsulación y reutilización
- **Nodos**: Estructuras que utilizan nodos (`Node` class) para enlazar elementos
- **Gestión de Punteros**: Uso de referencias para conectar elementos

Características Clave

Estructura	Complejidad de Tiempo	Caso de Uso
Array Personalizado	O(1) acceso, O(n) inserción	Acceso rápido por índice
Stack	O(1) push/pop	Deshacer/rehacer, recursión
Queue	O(1) enqueue/dequeue	Procesamiento FIFO, colas

Singly Linked List	O(n) búsqueda, O(1) inserción	Listas dinámicas
Doubly Linked List	O(n) búsqueda, O(1) inserción	Navegación bidireccional
Binary Search Tree	O(log n) búsqueda promedio	Búsqueda y ordenamiento
Hash Table	O(1) promedio	Búsqueda de clave-valor rápida
Graph	Varía según implementación	Redes, mapas, recomendaciones

Nivel Educativo

- **Dificultad**: Intermedia
- **Prerequisitos**: Conceptos básicos de JavaScript, POO, clases
- **Enfoque**: Implementación desde cero, sin librerías externas
- **Propósito**: Entender cómo funcionan internamente estas estructuras

Aplicaciones Prácticas

- **Stack**: Undo/Redo en aplicaciones, evaluación de expresiones
- **Queue**: Sistemas de procesamiento, colas de espera, BFS
- **Linked Lists**: Memoria dinámica, listas de reproducción
- **BST**: Índices de bases de datos, búsquedas eficientes
- **Hash Tables**: Diccionarios, cachés, búsquedas por clave
- **Graphs**: Redes sociales, sistemas de navegación, recomendadores

---

💡 Conclusión

Semana 9 presenta una colección completa y bien organizada de implementaciones de estructuras de datos clásicas. Es un recurso excelente para consolidar conocimientos teóricos y ver cómo estas estructuras se implementan en la práctica. Cada implementación está enfocada en claridad y educación, haciendo que sea fácil entender los mecanismos internos de cada estructura.

SEMANA 10

Semana 10 - Estructuras de Datos Avanzadas

Lenguaje

- JavaScript (ES6+)
- HTML5
- CSS3

Estructura de Archivos

```
Semana10/
├── code.js                    # Demo interactiva: consumo API Rick and Morty
├── index.html                 # Interfaz web para visualizar datos
├── style.css                  # Estilos de la aplicación web
├── linkedList.js              # Implementación de Lista Enlazada (Linked List)
├── binaryTree.js              # Implementación de Árbol Binario (Binary Tree)
├── dictionary.js              # Implementación de Diccionario (Dictionary/Hash Map)
├── hastTable.js               # Implementación de Tabla Hash (Hash Table)
├── Set.js                     # Operaciones con Conjuntos (Sets)
├── alternative code.txt       # Código alternativo/referencia
├── stack/
│   └── stack.js               # Implementación de Pila (Stack)
└── tree/
    └── tree.js                # Implementación de Árbol Binario de Búsqueda (BST)
```

Objetivo

Semana 10 tiene como objetivo proporcionar una comprensión profunda de las estructuras de datos avanzadas más utilizadas en ciencias de la computación. La carpeta combina implementaciones detalladas de estructuras de datos

clásicas con una aplicación práctica que demuestra cómo consumir APIs externas y manipular datos en una interfaz web interactiva.

Descripción General

Esta semana cubre:

1. Estructuras de Datos Lineales

- Linked List (linkedList.js): Lista enlazada simple con métodos push(), pop(), get() e isEmpty()
- Stack (stack/stack.js): Pila implementada con nodos enlazados, operaciones LIFO (Last In First Out)

2. Estructuras de Datos Jerárquicas

- Binary Tree (binaryTree.js): Árbol binario con métodos de recorrido (in-order, pre-order, post-order)
- Binary Search Tree (tree/tree.js): Árbol binario de búsqueda con operaciones de inserción

3. Estructuras de Datos Asociativas

- Diccionario (dictionary.js): Estructura clave-valor con métodos has(), set(), get(), delete()
- Hash Table (hastTable.js): Tabla hash con manejo de colisiones mediante direccionamiento abierto
- Set (Set.js): Conjunto con operaciones de unión e intersección

4. Aplicación Práctica Web

- code.js: Consumo de la API pública "Rick and Morty" usando fetch
- index.html: Interfaz responsiva para mostrar personajes
- style.css: Estilos modernos para la presentación

Análisis de Contenido

Características Principales

A. Estructuras Lineales

- LinkedList: Implementación completa con gestión de cabeza (head) y cola (tail)
    - Soporte para push y pop en O(1) y O(n) respectivamente
    - Manejo eficiente de memoria sin necesidad de asignación contigua
- Stack: Estructura LIFO basada en nodos enlazados
    - Operaciones push() y pop() en O(1)
    - Caso de uso: evaluación de expresiones, backtracking

B. Estructuras Jerárquicas

- BinaryTree: Árbol binario completo con tres tipos de recorrido
    - In-order, Pre-order, Post-order para diferentes necesidades de procesamiento
    - Método addChild() con validación de duplicados
    - Altura y profundidad calculables
- BST (Binary Search Tree): Árbol binario optimizado para búsqueda
    - Inserción manteniendo propiedad de orden
    - Complejidad de búsqueda O(log n) en promedio
    - Base para estructuras más complejas

C. Estructuras Asociativas

- Diccionario: Mapeo clave-valor simple usando objetos de JavaScript
    - Interfaz intuitiva similar a Map
    - O(1) en búsqueda, inserción y eliminación
- Hash Table: Implementación de tabla hash con sondeo lineal
    - Manejo de colisiones mediante direccionamiento abierto
    - Función hash con doble hash para evitar clustering
    - Validación de capacidad
- Set: Operaciones de teoría de conjuntos
    - Unión (unión), Intersección (intersección), Diferencia
    - Implementación usando Set nativo de JavaScript
    - Ejemplos prácticos de uso

D. Aplicación Práctica

- Rick & Morty Demo: Aplicación web funcional que:
    - Consume datos de API externa en tiempo real

- Renderiza tarjetas responsivas con información de personajes
- Incluye manejo de errores y estados de carga
- Interfaz moderna con branding inspirado en la API oficial

Patrones de Implementación

1. Clases ES6: Uso de sintaxis moderna de JavaScript para definir estructuras
2. Métodos Encapsulados: Cada estructura expone métodos públicos limpios
3. Manejo de Errores: Validaciones en operaciones críticas
4. Async/Await: Manejo moderno de promesas en el consumo de API
5. DOM Manipulation: Creación dinámica de elementos HTML

Complejidad Algoritmica

Estructura	Inserción	Búsqueda	Eliminación	Espacio
Lista enlazada	O(1)*	O(n)	O(n)	O(n)
Stack	O(1)	O(n)	O(1)	O(n)
Árbol binario	O(log n)**	O(log n)**	O(log n)**	O(n)
Tabla hash	O(1)	O(1)	O(1)	O(n)
Diccionario	O(1)	O(1)	O(1)	O(n)

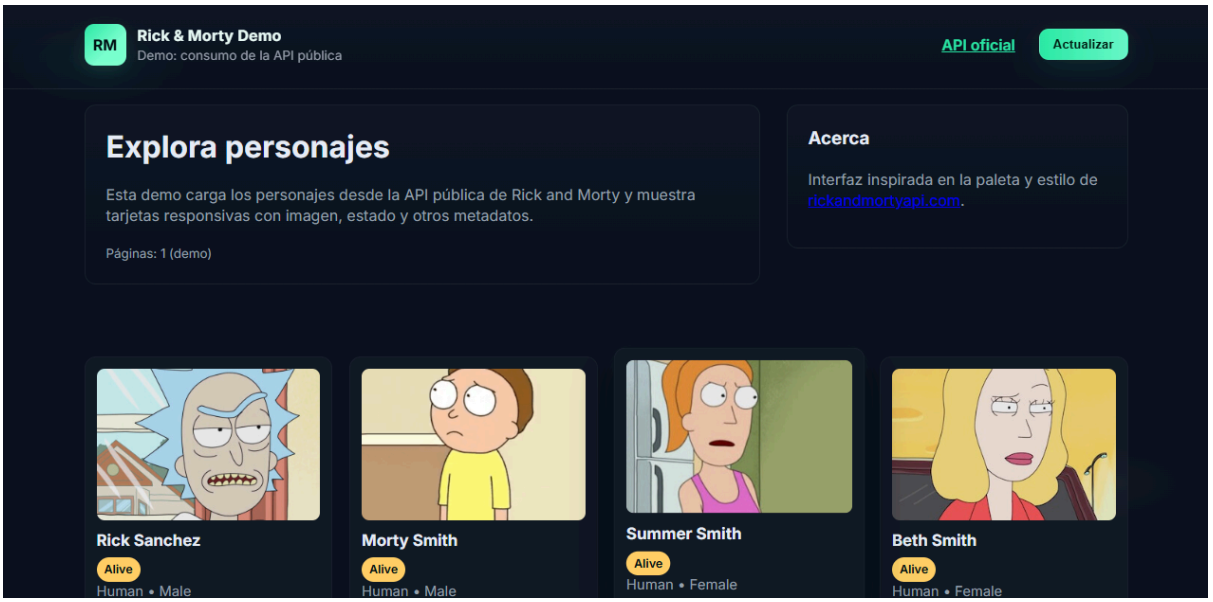
*Al final de la lista. **En promedio para BST balanceado

Casos de Uso

- Lista enlazada: Sistemas de gestión de memoria, implementación de colas
- Stack: Undo/Redo, evaluación de expresiones postfijas, DFS
- Binary Tree: Indexación de bases de datos, árbol de expresiones
- Hash Table: Caché, indexación rápida, detección de duplicados
- Set: Operaciones matemáticas, eliminación de duplicados

Conclusión

Semana 10 proporciona una base sólida en estructuras de datos fundamentales, combinando teoría con práctica. Los estudiantes pueden ver cómo estas estructuras se implementan desde cero y cómo se aplican en aplicaciones reales como el consumo de APIs web.



Semana 10 - Estructuras de Datos Avanzadas

Lenguaje

- **JavaScript (ES6+)**
- **HTML5**
- **CSS3**

Estructura de Archivos

```
Semana10/
├── code.js                    # Demo interactiva: consumo API Rick and Morty
├── index.html                 # Interfaz web para visualizar datos
├── style.css                  # Estilos de la aplicación web
├── linkedList.js              # Implementación de Lista Enlazada (Linked List)
├── binaryTree.js              # Implementación de Árbol Binario (Binary Tree)
├── dictionary.js              # Implementación de Diccionario (Dictionary/Hash Map)
├── hastTable.js               # Implementación de Tabla Hash (Hash Table)
├── Set.js                     # Operaciones con Conjuntos (Sets)
├── alternative code.txt       # Código alternativo/referencia
├── stack/
│   └── stack.js               # Implementación de Pila (Stack)
└── tree/
    └── tree.js                # Implementación de Árbol Binario de Búsqueda (BST)
```

Objetivo

Semana 10 tiene como objetivo proporcionar una comprensión profunda de las **estructuras de datos avanzadas** más utilizadas en ciencias de la computación. La carpeta combina implementaciones detalladas de estructuras de datos clásicas con una aplicación práctica que demuestra cómo consumir APIs externas y manipular datos en una interfaz web interactiva.

Descripción General

Esta semana cubre:

1. **Estructuras de Datos Lineales**
- **Linked List** (linkedList.js): Lista enlazada simple con métodos push(), pop(), get() e isEmpty()
- **Stack** (stack/stack.js): Pila implementada con nodos enlazados, operaciones LIFO (Last In First Out)

2. **Estructuras de Datos Jerárquicas**
- **Binary Tree** (binaryTree.js): Árbol binario con métodos de recorrido (in-order, pre-order, post-order)
- **Binary Search Tree** (tree/tree.js): Árbol binario de búsqueda con operaciones de inserción

3. **Estructuras de Datos Asociativas**
- **Dictionary** (dictionary.js): Estructura clave-valor con métodos has(), set(), get(), delete()
- **Hash Table** (hastTable.js): Tabla hash con manejo de colisiones mediante direccionamiento abierto
- **Set** (Set.js): Conjunto con operaciones de unión e intersección

4. **Aplicación Práctica Web**
- **code.js**: Consumo de la API pública "Rick and Morty" usando fetch
- **index.html**: Interfaz responsiva para mostrar personajes
- **style.css**: Estilos modernos para la presentación

Análisis de Contenido

Características Principales

A. Estructuras Lineales
- **LinkedList**: Implementación completa con gestión de cabeza (head) y cola (tail)
  - Soporte para push y pop en O(1) y O(n) respectivamente
  - Manejo eficiente de memoria sin necesidad de asignación contigua

- **Stack**: Estructura LIFO basada en nodos enlazados
  - Operaciones push() y pop() en O(1)
  - Caso de uso: evaluación de expresiones, backtracking

B. Estructuras Jerárquicas

- **BinaryTree**: Árbol binario completo con tres tipos de recorrido
  - In-order, Pre-order, Post-order para diferentes necesidades de procesamiento
  - Método addChild() con validación de duplicados
  - Altura y profundidad calculables

- **BST (Binary Search Tree)**: Árbol binario optimizado para búsqueda
  - Inserción manteniendo propiedad de orden
  - Complejidad de búsqueda O(log n) en promedio
  - Base para estructuras más complejas

C. Estructuras Asociativas

- **Dictionary**: Mapeo clave-valor simple usando objetos de JavaScript
  - Interfaz intuitiva similar a Map
  - O(1) en búsqueda, inserción y eliminación

- **Hash Table**: Implementación de tabla hash con sondeo lineal
  - Manejo de colisiones mediante direccionamiento abierto
  - Función hash con doble hash para evitar clustering
  - Validación de capacidad

- **Set**: Operaciones de teoría de conjuntos
  - Unión (union), Intersección (intersection), Diferencia
  - Implementación usando Set nativo de JavaScript
  - Ejemplos prácticos de uso

D. Aplicación Práctica

- **Rick & Morty Demo**: Aplicación web funcional que:
  - Consume datos de API externa en tiempo real
  - Renderiza tarjetas responsivas con información de personajes
  - Incluye manejo de errores y estados de carga
  - Interfaz moderna con branding inspirado en la API oficial

Patrones de Implementación

1. **Clases ES6**: Uso de sintaxis moderna de JavaScript para definir estructuras
2. **Métodos Encapsulados**: Cada estructura expone métodos públicos limpios
3. **Manejo de Errores**: Validaciones en operaciones críticas
4. **Async/Await**: Manejo moderno de promesas en el consumo de API
5. **DOM Manipulation**: Creación dinámica de elementos HTML

Complejidad Algoritmica

Estructura	Inserción	Búsqueda	Eliminación	Espacio
Linked List	O(1)*	O(n)	O(n)	O(n)
Stack	O(1)	O(n)	O(1)	O(n)
Binary Tree	O(log n)**	O(log n)**	O(log n)**	O(n)
Hash Table	O(1)	O(1)	O(1)	O(n)
Dictionary	O(1)	O(1)	O(1)	O(n)

*Al final de la lista. **En promedio para BST balanceado

Casos de Uso

- **Linked List**: Sistemas de gestión de memoria, implementación de colas
- **Stack**: Undo/Redo, evaluación de expresiones postfijas, DFS
- **Binary Tree**: Indexación de bases de datos, árbol de expresiones
- **Hash Table**: Caché, indexación rápida, detección de duplicados
- **Set**: Operaciones matemáticas, eliminación de duplicados

Conclusión

Semana 10 proporciona una base sólida en estructuras de datos fundamentales, combinando teoría con práctica. Los estudiantes pueden ver cómo estas estructuras se implementan desde cero y cómo se aplican en aplicaciones reales como el consumo de APIs web.

SEMANA 11

Semana 11 - Proyecto E-commerce con Carrito de Compras

📋 Descripción General

Este proyecto implementa una tienda virtual interactiva (e-commerce) con funcionalidad de carrito de compras dinámico. Es una aplicación web de una sola página que permite a los usuarios ver un catálogo de productos, agregarlos a un carrito de compras y gestionar su compra.

🔧 Lenguaje y Tecnologías Utilizadas

Tecnología	Versión	Uso
HTML5	-	Estructura y marcado del documento
CSS3	-	Estilos y diseño responsivo
JavaScript (ES6+)	-	Lógica interactiva y gestión de estado
JSON	-	Almacenamiento de datos de productos
LocalStorage API	-	Persistencia de datos del carrito
API de obtención	-	Carga dinámica de productos

📁 Estructura de Archivos

```
Semana 11/
├── index.html          # Archivo HTML principal con estructura del sitio
├── app.js              # Lógica principal de la aplicación
├── style.css           # Estilos y diseño visual
├── products.json       # Base de datos de productos en formato JSON
└── assets/             # Carpeta con imágenes de productos
```

```
    ├── 1.png-20.png   # Imágenes de los productos
    └── 1.jpg-20.jpg   # Imágenes adicionales
```

🎯 Objetivo Principal

Crear una aplicación de comercio electrónico funcional que demuestre:

1. Manipulación del DOM: Creación dinámica de elementos HTML desde JavaScript
2. Gestión de datos: Uso de estructuras de datos (arrays, objetos) para gestionar productos y carrito
3. Eventos en JavaScript: Listeners para interacciones del usuario
4. Persistencia de datos: Almacenamiento en LocalStorage
5. Consumo de APIs: Carga de datos JSON mediante Fetch API
6. Responsive Design: Interfaz adaptable a diferentes dispositivos

---

📖 Descripción Detallada

1. Estructura HTML (index.html)

```
- Contenedor principal (.container)
    ├── Header
    |   ├── Título "PRODUCT LIST"
    |   └── Ícono de carrito con badge de cantidad
    ├── Sección de productos (.listProduct)
    |   └── Items dinámicos generados por JavaScript
    └── Panel de carrito (.cartTab)
        ├── Título "Shopping Cart"
        ├── Lista de items del carrito
        └── Botones (CLOSE, CHECK OUT)
```

Características:

- Estructura semántica HTML5
- Ícono SVG descargado de Flowbite Icons
- Divisiones principales para productos y carrito
- Script vinculado al final para mejor rendimiento

2. Estilos CSS (style.css)

Características principales:

- Fuentes importadas de Google Fonts (Poppins, Kablammo)
- Flexbox para alineación y distribución
- Diseño responsivo con media queries
- Contenedor con ancho máximo (900px, 90vw)
- Estilos para:
    - Header con ícono de carrito posicionado
    - Galería de productos
    - Panel de carrito interactivo
    - Botones y controles

Animaciones:

- Transiciones suaves (0,5s)
- Toggle del carrito al hacer clic

3. Lógica JavaScript (app.js)

Variables globales:

```
let listProduct = [];       // Array de productos cargados
let carts = [];            // Array del carrito de compras
```

Funciones principales:

`addDataToHTML()`

- Limpia la lista de productos HTML
- Itera sobre el array `listProducts`
- Crea dinámicamente elementos con:`<div>`
    - Imagen del producto
    - Nombre
    - Precio
    - Botón "Añadir al carrito"

`addToCart(product_id)`

- Busca si el producto ya existe en el carrito

- Si es nuevo, lo agrega con cantidad 1
- Si existe, incrementa su cantidad
- Actualiza el HTML y el almacenamiento local

`addCartToHTML()`

- Renderiza los items del carrito
- Calcula la cantidad total y el precio total
- Crea botones de incremento/decremento (-, +)
- Actualiza el badge del ícono de carrito

`addCartToMemory()`

- Guarda el estado del carrito en `localStorage`
- Convierte el array a JSON para persistencia

`initApp()`

- Se ejecuta al cargar la página
- Realiza un fetch del archivo `products.json`
- Carga los productos y genera el HTML
- Restaura el carrito desde localStorage si existe

Oyentes del evento:

- Click en ícono de carrito: toggle de visibilidad
- Click en botón cerrar: oculta el carrito
- Click en productos: agrega items al carrito
- Click en controles de cantidad: incrementa/decrementa (en desarrollo)

4. Base de Datos (products.json)

Estructura de datos:

```
[
  {
    "id": número,
    "name": "Nombre del producto",
    "price": número decimal,
    "image": "ruta/a/imagen.png"
  }
]
```

Productos incluidos (20 artículos):

- Silla Moderna (75 $)
- Sillón Clásico (120,50 $)
- Silla de Oficina (95 $)
- Sillón Relax (200 $)
- Silla Gamer (150,25 $)
- Sillón Vintage (180,75 $)
- Silla de Comedor (45,99 $)
- Sillón Ejecutivo (210 dólares)
- Y más variantes...

🔍 Análisis de Contenido

Flujo de Ejecución

1. Inicialización: se ejecuta al cargar la página`initApp()`
2. Carga de datos: Fetch obtiene products.json
3. Renderizado: genera la galería de productos`addDataToHTML()`
4. Interacción usuario:
    - Clic en producto → `addToCart()`
    - Clic en carrito → toggle de visibilidad
5. Persistencia: guarda estado en localStorage`addCartToMemory()`
6. Actualización UI: refleja cambios en tiempo real`addCartToHTML()`

Patrones de Diseño Utilizados

Patrón	Implementación
MVC	Datos (JSON), Vista (HTML), Control (JS)
Observador	Event Listeners para interacciones
Fábrica	Creación dinámica de elementos DOM

Repositorio	localStorage como almacenamiento

Conceptos Clave Implementados

1. Manipulación del DOM: , , `createElement()` `appendChild()` `classList`
2. Plantillas Strings: Uso de backticks para HTML dinámico
3. Métodos de Array: , , `forEach()` `findIndex()` `push()`
4. Desestructuración: Acceso a propiedades de objetos JSON
5. Closure: Funciones que acceden a variables externas
6. Asincronía: Fetch API con promesas
7. Almacenamiento del lado del cliente: LocalStorage

Funcionalidades Completadas

✅ Carga de productos desde JSON ✅ Visualización de catálogo ✅ Agregar productos al carrito ✅ Mostrar/ocultar panel de carrito ✅ Actualizar cantidad de items en carrito ✅ Guardar carrito en localStorage ✅ Restaurar carrito al recargar página

Funcionalidades Incompletas

⚠️ Botones de incremento/decremento (-, +) no implementados completamente ⚠️ Cálculo de total de compra (estructura preparada, lógica pendiente) ⚠️ Funcionalidad de checkout ⚠️ Sistema de pago ⚠️ Validación de datos

---

💡 Conceptos de Estructuras de Datos

Matrices

- `listProducts`: almacena todos los productos disponibles
- `carts`: almacena items seleccionados por el usuario

Objetos

- Cada producto es un objeto con: , , , `idnamepriceimage`
- Cada ítem del carrito contiene: , `product_idquantity`

JSON

- Formato de intercambio de datos entre servidor y cliente

- Serialización y deserialización con y `JSON.stringify()` `JSON.parse()`

🚀 Cómo Ejecutar

1. Abre en un navegador web moderno `index.html`
2. Los productos se cargarán automáticamente desde `products.json`
3. Haz clic en "Add To Cart" para agregar productos
4. Haz clic en el ícono de carrito para ver tu compra
5. Los datos se guardan automáticamente en localStorage

Requisitos:

- Navegador con soporte para ES6, Fetch API, LocalStorage
- Servidor HTTP simple (puede usar Live Server en VS Code)

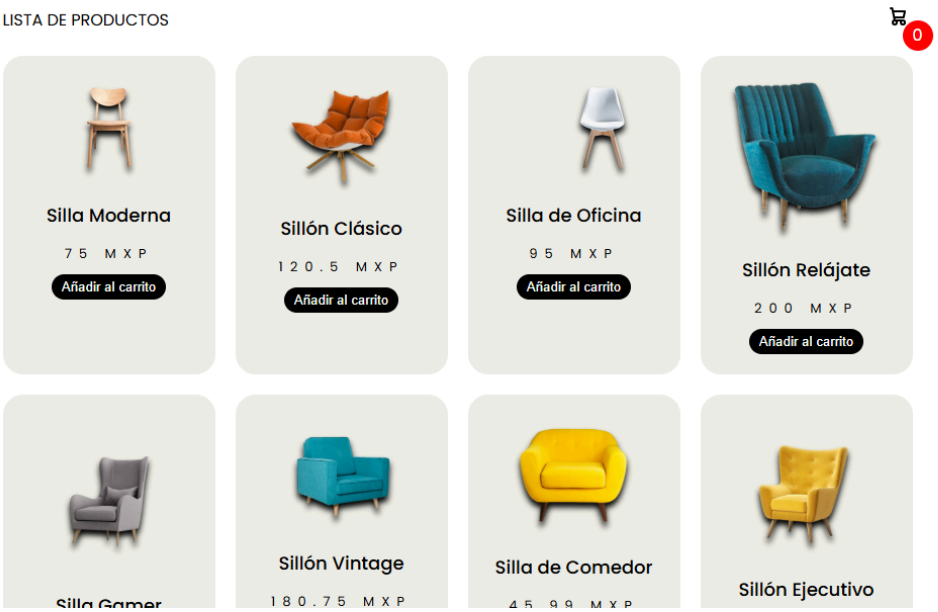
📝 Notas Importantes

- El proyecto utiliza para cargar , por lo que requiere un servidor HTTP `fetch()` `products.json`
- Los datos del carrito persisten entre sesiones gracias a localStorage
- El diseño es responsivo y se adapta a dispositivos móviles
- Las imágenes están organizadas en la carpeta `assets/`

📊 Resumén Técnico

Aspecto	Detalles
Líneas de código (aprox.)	HTML: 50, CSS: 200, JS: 130
Productos en catálogo	20

Métodos principales	5 (addDataToHTML, addToCart, addCartToHTML, addCartToMemory, initApp)
Oyentes del evento	3 principales
Complejidad	Medios - Intermedio
Propósito educativo	Aplicación práctica de DOM, eventos, fetch y almacenamiento



Semana 11 - Proyecto E-commerce con Carrito de Compras

📋 Descripción General

Este proyecto implementa una tienda virtual interactiva (e-commerce) con funcionalidad de carrito de compras dinámico. Es una aplicación web de una sola página que permite a los usuarios ver un catálogo de productos, agregarlos a un carrito de compras y gestionar su compra.

🔧 Lenguaje y Tecnologías Utilizadas

Tecnología	Versión	Uso
HTML5	-	Estructura y marcado del documento
CSS3	-	Estilos y diseño responsivo
JavaScript (ES6+)	-	Lógica interactiva y gestión de estado
JSON	-	Almacenamiento de datos de productos
LocalStorage API	-	Persistencia de datos del carrito
Fetch API	-	Carga dinámica de productos

📁 Estructura de Archivos

```
Semana 11/
├── index.html          # Archivo HTML principal con estructura del sitio
├── app.js              # Lógica principal de la aplicación
├── style.css           # Estilos y diseño visual
├── products.json       # Base de datos de productos en formato JSON
└── assets/             # Carpeta con imágenes de productos
    ├── 1.png-20.png    # Imágenes de los productos
    └── 1.jpg-20.jpg    # Imágenes adicionales
```

🎯 Objetivo Principal

Crear una aplicación de comercio electrónico funcional que demuestre:

1. **Manipulación del DOM**: Creación dinámica de elementos HTML desde JavaScript
2. **Gestión de datos**: Uso de estructuras de datos (arrays, objetos) para gestionar productos y carrito
3. **Eventos en JavaScript**: Listeners para interacciones del usuario
4. **Persistencia de datos**: Almacenamiento en LocalStorage
5. **Consumo de APIs**: Carga de datos JSON mediante Fetch API
6. **Responsive Design**: Interfaz adaptable a diferentes dispositivos

---

📖 Descripción Detallada

1. Estructura HTML (index.html)

```
- Contenedor principal (.container)
├── Header
│   ├── Título "PRODUCT LIST"
│   └── Ícono de carrito con badge de cantidad
├── Sección de productos (.listProduct)
│   └── Items dinámicos generados por JavaScript
└── Panel de carrito (.cartTab)
    ├── Título "Shopping Cart"
    ├── Lista de items del carrito
    └── Botones (CLOSE, CHECK OUT)
```

Características:
- Estructura semántica HTML5
- Ícono SVG descargado de Flowbite Icons
- Divisiones principales para productos y carrito
- Script vinculado al final para mejor rendimiento

2. Estilos CSS (style.css)

Características principales:
- Fuentes importadas de Google Fonts (Poppins, Kablammo)
- Flexbox para alineación y distribución
- Diseño responsivo con media queries
- Contenedor con ancho máximo (900px, 90vw)
- Estilos para:

- Header con ícono de carrito posicionado
- Galería de productos
- Panel de carrito interactivo
- Botones y controles

Animaciones:
- Transiciones suaves (0.5s)
- Toggle del carrito al hacer clic

3. Lógica JavaScript (app.js)

Variables globales:
```javascript
let listProduct = [];      // Array de productos cargados
let carts = [];            // Array del carrito de compras
```

Funciones principales:

`addDataToHTML()`
- Limpia la lista de productos HTML
- Itera sobre el array `listProducts`
- Crea dinámicamente elementos `<div>` con:
  - Imagen del producto
  - Nombre
  - Precio
  - Botón "Add To Cart"

`addToCart(product_id)`
- Busca si el producto ya existe en el carrito
- Si es nuevo, lo agrega con cantidad 1
- Si existe, incrementa su cantidad
- Actualiza el HTML y el almacenamiento local

`addCartToHTML()`
- Renderiza los items del carrito
- Calcula la cantidad total y el precio total
- Crea botones de incremento/decremento (-, +)
- Actualiza el badge del ícono de carrito

`addCartToMemory()`
- Guarda el estado del carrito en `localStorage`
- Convierte el array a JSON para persistencia

`initApp()`
- Se ejecuta al cargar la página
- Realiza un fetch del archivo `products.json`
- Carga los productos y genera el HTML
- Restaura el carrito desde localStorage si existe

Event Listeners:
- Click en ícono de carrito: toggle de visibilidad
- Click en botón cerrar: oculta el carrito
- Click en productos: agrega items al carrito
- Click en controles de cantidad: incrementa/decrementa (en desarrollo)

4. Base de Datos (products.json)

Estructura de datos:
```json
[
  {
    "id": número,
    "name": "Nombre del producto",
    "price": número decimal,
    "image": "ruta/a/imagen.png"
  }
]
```

Productos incluidos (20 items):
- Silla Moderna ($75)
- Sillón Clásico ($120.50)
- Silla de Oficina ($95)
- Sillón Relax ($200)
- Silla Gamer ($150.25)
- Sillón Vintage ($180.75)
- Silla de Comedor ($45.99)
- Sillón Ejecutivo ($210)
- Y más variantes...

🔍 Análisis de Contenido

Flujo de Ejecución

1. **Inicialización**: `initApp()` se ejecuta al cargar la página
2. **Carga de datos**: Fetch obtiene products.json
3. **Renderizado**: `addDataToHTML()` genera la galería de productos
4. **Interacción usuario**:
   - Clic en producto → `addToCart()`
   - Clic en carrito → toggle de visibilidad
5. **Persistencia**: `addCartToMemory()` guarda estado en localStorage
6. **Actualización UI**: `addCartToHTML()` refleja cambios en tiempo real

Patrones de Diseño Utilizados

Patrón	Implementación
MVC	Datos (JSON), Vista (HTML), Control (JS)
Observer	Event Listeners para interacciones
Factory	Creación dinámica de elementos DOM
Repository	localStorage como almacenamiento

Conceptos Clave Implementados

1. **Manipulación del DOM**: `createElement()`, `appendChild()`, `classList`
2. **Template Strings**: Uso de backticks para HTML dinámico
3. **Métodos de Array**: `forEach()`, `findIndex()`, `push()`
4. **Desestructuración**: Acceso a propiedades de objetos JSON
5. **Closure**: Funciones que acceden a variables externas
6. **Asincronía**: Fetch API con promesas
7. **Almacenamiento del lado del cliente**: LocalStorage

Funcionalidades Completadas

✅ Carga de productos desde JSON
✅ Visualización de catálogo
✅ Agregar productos al carrito
✅ Mostrar/ocultar panel de carrito
✅ Actualizar cantidad de items en carrito
✅ Guardar carrito en localStorage
✅ Restaurar carrito al recargar página

Funcionalidades Incompletas

⚠️ Botones de incremento/decremento (-, +) no implementados completamente
⚠️ Cálculo de total de compra (estructura preparada, lógica pendiente)
⚠️ Funcionalidad de checkout
⚠️ Sistema de pago
⚠️ Validación de datos

💡 Conceptos de Estructuras de Datos

Arrays
- `listProducts`: almacena todos los productos disponibles
- `carts`: almacena items seleccionados por el usuario

Objetos
- Cada producto es un objeto con: `id`, `name`, `price`, `image`
- Cada item del carrito contiene: `product_id`, `quantity`

JSON
- Formato de intercambio de datos entre servidor y cliente
- Serialización y deserialización con `JSON.stringify()` y `JSON.parse()`

🚀 Cómo Ejecutar

1. Abre `index.html` en un navegador web moderno
2. Los productos se cargarán automáticamente desde `products.json`
3. Haz clic en "Add To Cart" para agregar productos
4. Haz clic en el ícono de carrito para ver tu compra
5. Los datos se guardan automáticamente en localStorage

Requisitos:
- Navegador con soporte para ES6, Fetch API, LocalStorage
- Servidor HTTP simple (puede usar Live Server en VS Code)

📝 Notas Importantes

```
- El proyecto utiliza `fetch()` para cargar `products.json`, por lo que requiere un servidor HTTP
- Los datos del carrito persisten entre sesiones gracias a localStorage
- El diseño es responsivo y se adapta a dispositivos móviles
- Las imágenes están organizadas en la carpeta `assets/`

---

## 📊 Resumén Técnico

| Aspecto | Detalles |
|---|---|
| **Líneas de código (aprox.)** | HTML: 50, CSS: 200, JS: 130 |
| **Productos en catálogo** | 20 |
| **Métodos principales** | 5 (addDataToHTML, addToCart, addCartToHTML, addCartToMemory, initApp) |
| **Event Listeners** | 3 principales |
| **Complejidad** | Media - Intermedio |
| **Propósito educativo** | Aplicación práctica de DOM, eventos, fetch y almacenamiento |
```

SEMANA 12

Análisis Semana 12: Pokédex Interactiva

📋 Información General

Aspecto	Descripción
Lenguaje	HTML5, CSS3, JavaScript (ES6+)
Tipo de Proyecto	Aplicación web interactiva
Objetivo	Desarrollar una Pokédex dinámica que consume una API externa (PokeAPI)
API Utilizada	PokeAPI

📁 Estructura de Archivos

```
Semana 12/
├── archivos-iniciales/
│   ├── index.html          # Estructura HTML principal
│   ├── css/
│   │   └── main.css        # Estilos y diseño responsivo
│   ├── img/                # Carpeta para logos e imágenes
│   └── js/
│       └── main.js         # Lógica JavaScript
```

🎯 Objetivo Principal

Crear una Pokédex interactiva que permita a los usuarios:

- Visualizar todos los Pokémon disponibles en la API (1025 registros)
- Pokémon Filtrar por tipo (Normal, Fuego, Agua, Planta, Eléctrico, etc.)
- Ver información detallada de cada Pokémon (ID, nombre, imagen, tipo, altura y peso)

📝 Descripción Detallada

Frontend - HTML (index.html)

- Estructura semántica con header, nav y main
- Navegación: Botones para filtrar por tipo de Pokémon
    - "Ver todos" para mostrar la lista completa
    - 18 botones para filtrar por cada tipo específico
- Contenedor dinámico con ID donde se insertan los Pokémon`listaPokemon`

Estilismo - CSS (main.css)

- Define variables CSS para colores según el tipo de Pokémon
- Cada tipo tiene su propio color distintivo (Normal, Fuego, Agua, etc.)
- Fuente personalizada: Rubik (Google Fonts)
- Diseño limpio y moderno con paleta de colores Pokémon oficial

Lógica - JavaScript (main.js)

- Consumo de API: Realiza fetch requests a `https://pokeapi.co/api/v2/pokemon/`
- Carga inicial: Itera del 1 al 1025 para obtener todos los Pokémon
- Función `mostrarPokemon()`:
    - Extrae tipos de cada Pokémon
    - Formatea IDs con padding de ceros (001, 002, etc.)
    - Crea dinámicamente elementos HTML con tarjetas de Pokémon
    - Incluye imagen oficial del Pokémon desde GitHub
- Sistema de filtrado:
    - Event listeners en botones de header
    - Al hacer clic, filtra Pokémon que contengan el tipo seleccionado
    - Limpia y recarga la lista dinámicamente

🔍 Análisis de Contenido

Conceptos Clave Implementados

Concepto	Detalles
API de obtención	Comunicación asincrónica con servidor externo
Promesas	Manejo de para respuestas de API`.then()`
JSON	Parseo y manipulación de datos JSON
Manipulación del DOM	Creación y modificación dinámica de elementos
Oyentes del evento	Interactividad con botones
Métodos de Array	`map()`, , para procesamiento de datos`join()` `some()`
Literales de plantilla	Interpolación de datos en HTML
CSS Variables	Dinamismo y mantenibilidad del diseño

Flujo de Ejecución

1. Inicialización: Al cargar la página, se hacen 1025 solicitudes fetch simultáneas
2. Renderizado: Cada respuesta se procesa y se añade a la vista
3. Interactividad: El usuario puede filtrar haciendo clic en botones
4. Re-render: La página se limpia y se vuelven a cargar solo los Pokémon que coinciden

Características Principales

✅ Carga masiva de datos: Gestión de 1025 registros desde API

✅ Filtrado dinámico: 19 opciones de filtrado (todos + 18 tipos)

✅ Diseño responsivo: CSS preparado para múltiples dispositivos

✅ Información visual: Imágenes oficiales de Pokémon desde repositorio externo

✅ Información detallada: ID, nombre, tipo(s), altura y peso

💡 Aprendizajes Aplicados

Este proyecto pone en práctica:

- Asincronía en JavaScript: Manejo de múltiples promises simultáneamente
- Integración de APIs externas: Consumo de datos en tiempo real
- Manipulación del DOM: Creación y actualización de elementos
- Estructuras de datos: Procesamiento de arrays y objetos JSON
- CSS avanzado: Variables, grid/flexbox, diseño moderno
- UX/UI: Interfaz intuitiva y bien diseñada

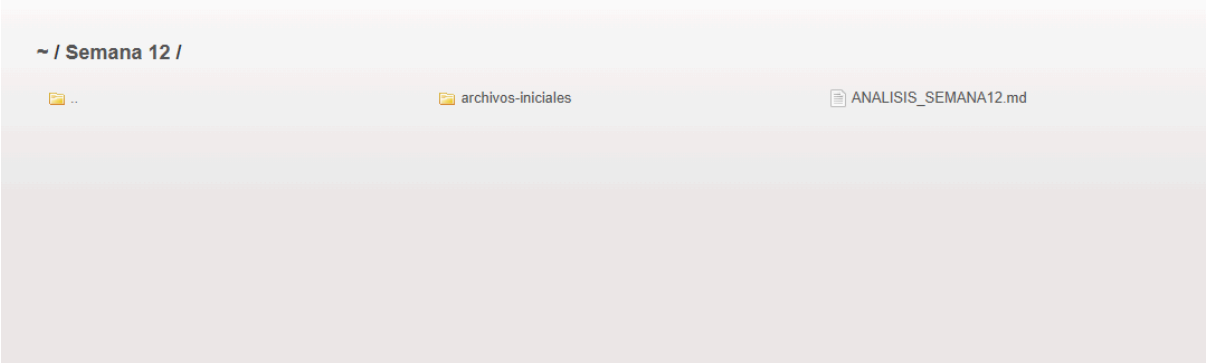
🚀 Tecnologías Utilizadas

- HTML5: Estructura semántica moderna
- CSS3: Diseño responsivo con variables y flexbox
- JavaScript ES6+: Sintaxis moderna (funciones de flecha, literales de plantilla)
- PokeAPI v2: API REST gratuita de datos de Pokémon
- GitHub Raw Content: Para servir imágenes de sprites oficiales

---

📌 Notas Adicionales

- El proyecto realiza muchas solicitudes simultáneas, lo que puede afectar el rendimiento
- Sería mejora futura: Paginación o infinite scroll para optimizar
- La API puede ser lenta la primera vez que se carga debido a las 1025 solicitudes
-



Análisis Semana 12: Pokédex Interactiva

📋 Información General

Aspecto	Descripción
Lenguaje	HTML5, CSS3, JavaScript (ES6+)
Tipo de Proyecto	Aplicación web interactiva
Objetivo	Desarrollar una Pokédex dinámica que consume una API externa (PokeAPI)
API Utilizada	[PokeAPI](https://pokeapi.co/)

📁 Estructura de Archivos

```
Semana 12/
├── archivos-iniciales/
│   ├── index.html          # Estructura HTML principal
│   ├── css/
│   │   └── main.css        # Estilos y diseño responsivo
│   ├── img/                # Carpeta para logos e imágenes
│   └── js/
│       └── main.js         # Lógica JavaScript
```

🎯 Objetivo Principal

Crear una **Pokédex interactiva** que permita a los usuarios:
- Visualizar todos los Pokémon disponibles en la API (1025 registros)
- Filtrar Pokémon por tipo (Normal, Fire, Water, Grass, Electric, etc.)
- Ver información detallada de cada Pokémon (ID, nombre, imagen, tipo, altura y peso)

📝 Descripción Detallada

Frontend - HTML (index.html)
- Estructura semántica con header, nav y main

- **Navegación**: Botones para filtrar por tipo de Pokémon
  - "Ver todos" para mostrar la lista completa
  - 18 botones para filtrar por cada tipo específico
- **Contenedor dinámico** con ID `listaPokemon` donde se insertan los Pokémon

Styling - CSS (main.css)
- Define variables CSS para colores según el tipo de Pokémon
- Cada tipo tiene su propio color distintivo (Normal, Fire, Water, etc.)
- Fuente personalizada: Rubik (Google Fonts)
- Diseño limpio y moderno con paleta de colores Pokémon oficial

Lógica - JavaScript (main.js)
- **Consumo de API**: Realiza fetch requests a `https://pokeapi.co/api/v2/pokemon/`
- **Carga inicial**: Itera del 1 al 1025 para obtener todos los Pokémon
- **Función `mostrarPokemon()`**:
  - Extrae tipos de cada Pokémon
  - Formatea IDs con padding de ceros (001, 002, etc.)
  - Crea dinámicamente elementos HTML con tarjetas de Pokémon
  - Incluye imagen oficial del Pokémon desde GitHub
- **Sistema de filtrado**:
  - Event listeners en botones de header
  - Al hacer clic, filtra Pokémon que contengan el tipo seleccionado
  - Limpia y recarga la lista dinámicamente

🔍 Análisis de Contenido

Conceptos Clave Implementados

Concepto	Detalles
Fetch API	Comunicación asincrónica con servidor externo
Promises	Manejo de `.then()` para respuestas de API
JSON	Parseo y manipulación de datos JSON
DOM Manipulation	Creación y modificación dinámica de elementos
Event Listeners	Interactividad con botones
Array Methods	`map()`, `join()`, `some()` para procesamiento de datos

| **Template Literals** | Interpolación de datos en HTML |
| **CSS Variables** | Dinamismo y mantenibilidad del diseño |

Flujo de Ejecución

1. **Inicialización**: Al cargar la página, se hacen 1025 solicitudes fetch simultáneas
2. **Renderizado**: Cada respuesta se procesa y se añade a la vista
3. **Interactividad**: El usuario puede filtrar haciendo clic en botones
4. **Re-render**: La página se limpia y se vuelven a cargar solo los Pokémon que coinciden

Características Principales

✅ **Carga masiva de datos**: Gestión de 1025 registros desde API
✅ **Filtrado dinámico**: 19 opciones de filtrado (todos + 18 tipos)
✅ **Diseño responsivo**: CSS preparado para múltiples dispositivos
✅ **Información visual**: Imágenes oficiales de Pokémon desde repositorio externo
✅ **Información detallada**: ID, nombre, tipo(s), altura y peso

---

💡 Aprendizajes Aplicados

Este proyecto pone en práctica:
- **Asincronía en JavaScript**: Manejo de múltiples promises simultáneamente
- **Integración de APIs externas**: Consumo de datos en tiempo real
- **Manipulación del DOM**: Creación y actualización de elementos
- **Estructuras de datos**: Procesamiento de arrays y objetos JSON
- **CSS avanzado**: Variables, grid/flexbox, diseño moderno
- **UX/UI**: Interfaz intuitiva y bien diseñada

🚀 Tecnologías Utilizadas

- **HTML5**: Estructura semántica moderna
- **CSS3**: Diseño responsivo con variables y flexbox
- **JavaScript ES6+**: Sintaxis moderna (arrow functions, template literals)

- **PokeAPI v2**: API REST gratuita de datos de Pokémon
- **GitHub Raw Content**: Para servir imágenes de sprites oficiales

---

📌 Notas Adicionales

- El proyecto realiza muchas solicitudes simultáneas, lo que puede afectar el rendimiento
- Seria mejora futura: Paginación o infinite scroll para optimizar
- La API puede ser lenta la primera vez que se carga debido a las 1025 solicitudes

SEMANA 13

Semana 13 - Sistema de Asistencia y Análisis de Algoritmos

📋 Resumen Ejecutivo

Semana 13 es un módulo híbrido que combina un sistema completo de gestión educativa (EduTrack) con ejemplos prácticos de algoritmos avanzados (Recursión y Ordenamiento). Proporciona una aplicación web funcional para instituciones educativas junto con implementaciones fundamentales de conceptos algorítmicos.

🔤 Lenguaje y Tecnologías

Tecnología	Versión	Propósito
JavaScript	ES6+	Lógica principal y funcionalidad
HTML5	Estándar	Estructura y marcado
CSS3	Con Tailwind	Estilos y responsividad
LocalStorage	Nativo	Base de datos del cliente
Anime.js	-	Animaciones fluidas

📁 Estructura de Archivos

```
Semana13/
├── index.html                  # Interfaz web principal
├── main.js                     # Lógica central del sistema EduTrack (983 líneas)
├── README.md                   # Documentación completa del proyecto
├── database.md                 # Esquema y estructura de base de datos
└── design.md                   # Guía de diseño visual y UX
```

```
├── processes.md                # Procesos y flujos del sistema
│
├── Recursion/                  # Ejemplos de algoritmos recursivos
│   ├── callStackExample.js     # Demostración de la pila de llamadas
│   ├── countToZero.js          # Recursión simple (contar hacia atrás)
│   ├── fibonacciIterative.js   # Fibonacci versión iterativa O(n)
│   ├── fibonacciRecursive.js   # Fibonacci versión recursiva O(2^n)
│   └── fibonacciRecursiveBetter.js # Fibonacci optimizada (memoización)
│
└── Sorting/                    # Algoritmos de ordenamiento avanzados
    ├── mergesort.js            # Merge Sort O(n log n) - Divide y conquista
    ├── quicksort.js            # Quick Sort O(n log n) promedio
    └── selectionSort.js        # Selection Sort O(n²) - Comparación
```

🎯 Objetivo General

Desarrollar una solución integral educativa que integre:

1. Sistema de Gestión (EduTrack): Aplicación web para docentes que permite registrar asistencia, participación y generar reportes.
2. Fundamentos Algorítmicos: Implementaciones prácticas de conceptos clave en Ciencias de la Computación.

📌 Descripción Detallada

1. Sistema EduTrack - Aplicación Principal

Propósito

Proporcionar una herramienta moderna para que docentes gestionen:

- ✅ Asistencia de estudiantes
- 💬 Participación en clase
- 📊 Análisis y reportes
- 📈 Dashboard con estadísticas

Componentes Principales

Clases Principales (main.js)

- `EduTrackSystem`: Clase central que gestiona toda la funcionalidad
    - Gestión de estudiantes y clases
    - Registro de asistencia y participación
    - Generación de reportes
    - Manejo de eventos de interfaz

Estructura de Datos (database.md)

- estudiantes: Lista de estudiantes con perfil completo
- clases: Clases con horarios y docentes
- Récords de asistencia: Registro temporal de asistencia
- participationRecords: Evaluación de participación (escala 1-10)
- claseSessions: Sesiones específicas de clase

Interfaz Visual (index.html + design.md)

- Dashboard responsivo con estadísticas en tiempo real
- Formularios para registro de asistencia
- Panel de control para docentes
- Notificaciones y micro-interacciones
- Diseño minimalista con paleta profesional

Características Clave

- 📊 Dashboard: Visualización de estadísticas generales
- ➕ Marcación Masiva: Registrar asistencia de todos rápidamente
- 🔍 Seguimiento Individual: Análisis por estudiante
- 📥 Exportación JSON: Descarga de datos
- 📱 Diseño Responsivo: Funciona en desktop y móvil

---

2. Módulo de recursión

Conceptos Cubiertos

Nivel Básico

- `callStackExample.js`: Demostración visual de cómo funciona la pila de llamadas
- `countToZero.js`: Función recursiva simple que cuenta regresivamente

Nivel Intermedio

`fibonacciIterative.js`: Solución iterativa O(n)

```
// Ejemplo: genera secuencia de Fibonacci
// Complejidad: O(n) - lineal
```

-

Nivel Avanzado

`fibonacciRecursive.js`: Solución recursiva pura O(2^n)

```
function getNthFibo(n) {
  if (n <= 1) return n;
  return getNthFibo(n - 1) + getNthFibo(n - 2);
}
// Problema: exponencial, muy lenta para n > 40
```

-
- `fibonacciRecursiveBetter.js`: Recursión optimizada con memoización
    - Reduce complejidad de O(2^n) a O(n)
    - Demuestra técnicas de optimización

Objetivo Pedagógico
Comprender las ventajas y desventajas de la recursión frente a iteración, y técnicas de optimización como memoización.

---

3. Clasificación de módulos

Algoritmos Implementados

Algoritmo	Complejidad	Tipo	Archivo
Clasificación de selección	O(n²)	Comparación	`selectionSort.js`
Ordenación por fusión	O(n log n)	Divide y Conquista	`mergesort.js`

Ordenación rápida	O(n log n)*	Divide y Conquista	`quicksort.js`

Ordenación por fusión - Ejemplo

```
function merge(leftA, rightA) {
  // Combina dos arrays ordenados
  var results = [], leftIndex = 0, rightIndex = 0;
  while (leftIndex < leftA.length && rightIndex < rightA.length) {
    if (leftA[leftIndex] < rightA[rightIndex]) {
      results.push(leftA[leftIndex++]);
    } else {
      results.push(rightA[rightIndex++]);
    }
  }
  return
results.concat(leftA.slice(leftIndex)).concat(rightA.slice(rightIndex));
}

function mergeSort(array) {
  if (array.length < 2) return array;
  var midpoint = Math.floor(array.length / 2);
  var leftArray = array.slice(0, midpoint);
  var rightArray = array.slice(midpoint);
  return merge(mergeSort(leftArray), mergeSort(rightArray));
}
```

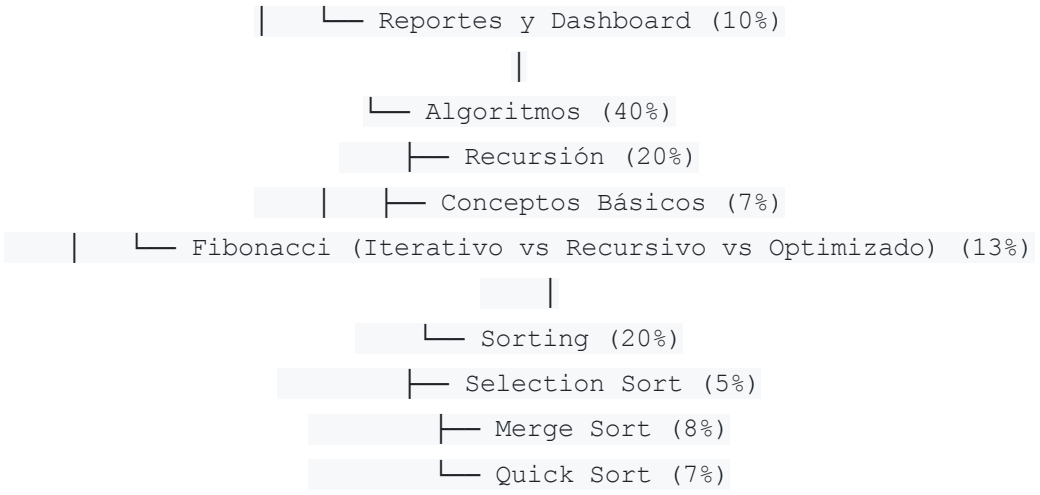
Objetivo Pedagógico

Comparar estrategias de ordenamiento, entender trade-offs entre complejidad y implementación, y aplicar paradigmas como "divide y conquista".

🔍 Análisis de Contenido

Distribución de Temas

```
Semana 13 (100%)
├── Sistema EduTrack (60%)
│   ├── Gestión de Estudiantes y Clases (20%)
│   ├── Funcionalidad de Asistencia (15%)
│   ├── Funcionalidad de Participación (15%)
```



Progresión de Dificultad

1. Introducción (EduTrack): Arquitectura de sistemas completos
2. Fundamental (Recursión Básica): Conceptos de pila y llamadas recursivas
3. Intermedio (Fibonacci): Comparación de enfoques (iterativo vs recursivo)
4. Avanzado (Optimización): Memoización y técnicas de mejora de performance
5. Avanzado (Ordenación): Algoritmos O(n²) vs O(n log n), divide y conquista

Habilidades Desarrolladas

✅ Backend/Lógica

- Diseño de clases y sistemas orientados a objetos
- Estructuras de datos complejas
- Implementación de algoritmos eficientes
- Memoización y técnicas de optimización

✅ Frontend/UI

- Desarrollo de interfaces responsivas
- Manipulación del DOM
- Manejo de eventos y formularios
- Integración con animaciones

✅ Bases de Datos

- Modelado de entidades
- Esquemas de almacenamiento

- Relaciones entre tablas

✅ Análisis

- Complejidad computacional (Big O)
- Compensaciones entre estrategias
- Optimización de código

🎓 Conexiones Pedagógicas

Relación con Semanas Anteriores

- Semana 1-9: Estructuras de datos (arrays, listas enlazadas, árboles, gráficos, tablas hash)
- Semana 10: Árboles binarios y sets
- Semana 11-12: Aplicaciones web interactivas
- Semana 13: ⭐ Integración: Usa estructuras de datos en una aplicación real + algoritmos fundamentales

Relación con Semanas Posteriores

- Proporciona base para algoritmos más complejos
- Demuestra cómo aplicar estructuras de datos en sistemas reales
- Introduce optimización (importante para escalabilidad)

📚 Puntos Clave para Aprender

1. EduTrack: Cómo estructurar una aplicación educativa completa
2. Recursión: Cuándo usar recursión vs iteración y cómo optimizar
3. Sorting: Comparación de algoritmos y sus características
4. Performance: Cómo analizar y mejorar la complejidad computacional
5. Integración: Usar múltiples conceptos en un proyecto cohesivo

---

🚀 Cómo Usar Este Módulo

Para Ejecutar EduTrack

1. Abrir en el navegador`index.html`
2. Sistema carga datos de ejemplo automáticamente
3. Interactuar con la interfaz para registrar asistencia y participación
4. Ver reportes en tiempo real

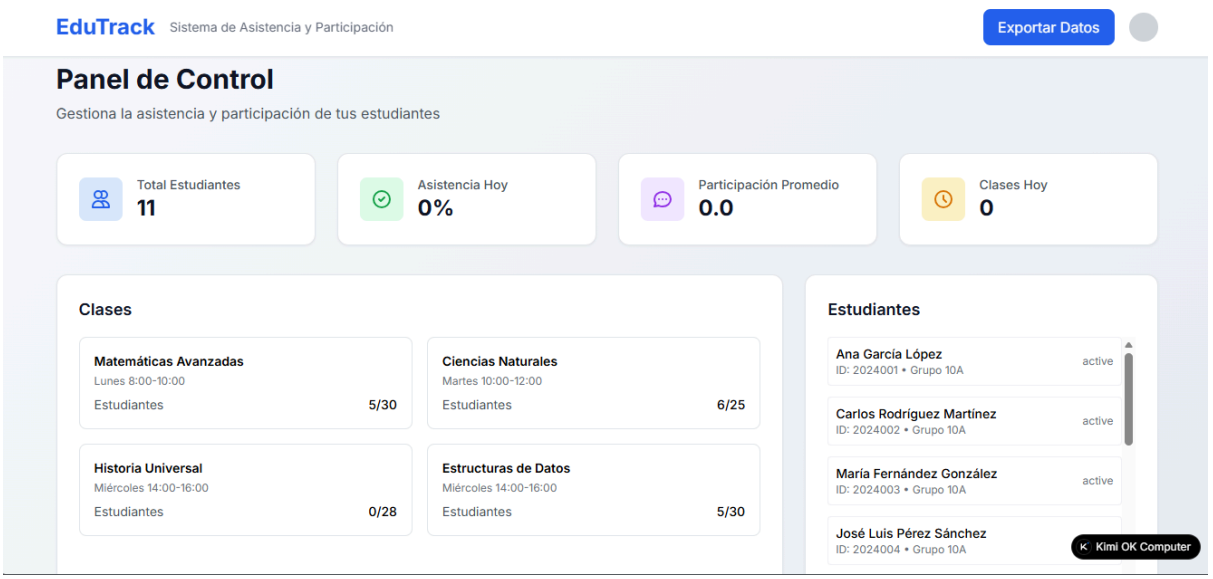
Para Estudiar Algoritmos

1. Revisar comentarios en cada archivo de Recursion/ y Sorting/
2. Comparar complejidad: ejecutar con diferentes tamaños de entrada
3. Modificar código para experimentar con optimizaciones
4. Usar herramientas del navegador (DevTools) para medir performance

---

📖 Archivos de Documentación Relacionados

- README.md - Descripción completa de EduTrack
- database.md - Estructura de base de datos
- design.md - Guía visual y UX
- processes.md - Flujos de procesos



Semana 13 - Sistema de Asistencia y Análisis de Algoritmos

📋 Resumen Ejecutivo

Semana 13 es un módulo híbrido que combina un **sistema completo de gestión educativa (EduTrack)** con ejemplos prácticos de **algoritmos avanzados** (Recursión y Ordenamiento). Proporciona una aplicación web funcional para instituciones educativas junto con implementaciones fundamentales de conceptos algorítmicos.

🔤 Lenguaje y Tecnologías

Tecnología	Versión	Propósito
JavaScript	ES6+	Lógica principal y funcionalidad
HTML5	Estándar	Estructura y marcado
CSS3	Con Tailwind	Estilos y responsividad
LocalStorage	Nativo	Base de datos del cliente
Anime.js	-	Animaciones fluidas

📁 Estructura de Archivos

```
Semana13/
├── index.html                    # Interfaz web principal
├── main.js                       # Lógica central del sistema EduTrack (983 líneas)
├── README.md                     # Documentación completa del proyecto
├── database.md                   # Esquema y estructura de base de datos
├── design.md                     # Guía de diseño visual y UX
├── processes.md                  # Procesos y flujos del sistema
│
├── Recursion/                    # Ejemplos de algoritmos recursivos
│   ├── callStackExample.js       # Demostración de la pila de llamadas
│   ├── countToZero.js            # Recursión simple (contar hacia atrás)
│   ├── fibonacciIterative.js     # Fibonacci versión iterativa O(n)
│   ├── fibonacciRecursive.js     # Fibonacci versión recursiva O(2^n)
│   └── fibonacciRecursiveBetter.js # Fibonacci optimizada (memoización)
```

```
└── Sorting/                    # Algoritmos de ordenamiento avanzados
    ├── mergesort.js            # Merge Sort O(n log n) - Divide y conquista
    ├── quicksort.js            # Quick Sort O(n log n) promedio
    └── selectionSort.js        # Selection Sort O(n²) - Comparación
```

🎯 Objetivo General

Desarrollar una **solución integral educativa** que integre:

1. **Sistema de Gestión (EduTrack)**: Aplicación web para docentes que permite registrar asistencia, participación y generar reportes.
2. **Fundamentos Algorítmicos**: Implementaciones prácticas de conceptos clave en Ciencias de la Computación.

📌 Descripción Detallada

1. **Sistema EduTrack - Aplicación Principal**

Propósito
Proporcionar una herramienta moderna para que docentes gestionen:
- ✅ Asistencia de estudiantes
- 💬 Participación en clase
- 📊 Análisis y reportes
- 📈 Dashboard con estadísticas

Componentes Principales

Clases Principales (main.js)
- `EduTrackSystem`: Clase central que gestiona toda la funcionalidad
  - Gestión de estudiantes y clases
  - Registro de asistencia y participación
  - Generación de reportes
  - Manejo de eventos de interfaz

Estructura de Datos (database.md)
- **students**: Lista de estudiantes con perfil completo

- **classes**: Clases con horarios y docentes
- **attendanceRecords**: Registro temporal de asistencia
- **participationRecords**: Evaluación de participación (escala 1-10)
- **classSessions**: Sesiones específicas de clase

Interfaz Visual (index.html + design.md)
- Dashboard responsivo con estadísticas en tiempo real
- Formularios para registro de asistencia
- Panel de control para docentes
- Notificaciones y micro-interacciones
- Diseño minimalista con paleta profesional

Características Clave
- 📊 **Dashboard**: Visualización de estadísticas generales
- ➕ **Marcación Masiva**: Registrar asistencia de todos rápidamente
- 🔍 **Seguimiento Individual**: Análisis por estudiante
- 📥 **Exportación JSON**: Descarga de datos
- 📱 **Diseño Responsivo**: Funciona en desktop y móvil

---

2. **Módulo Recursión**

Conceptos Cubiertos

Nivel Básico
- `callStackExample.js`: Demostración visual de cómo funciona la pila de llamadas
- `countToZero.js`: Función recursiva simple que cuenta regresivamente

Nivel Intermedio
- `fibonacciIterative.js`: Solución iterativa O(n)
```javascript
// Ejemplo: genera secuencia de Fibonacci
// Complejidad: O(n) - lineal
```

Nivel Avanzado
- `fibonacciRecursive.js`: Solución recursiva pura O(2^n)
```javascript
function getNthFibo(n) {
  if (n <= 1) return n;
return getNthFibo(n - 1) + getNthFibo(n - 2);
```

```
}
// Problema: exponencial, muy lenta para n > 40
```

- `fibonacciRecursiveBetter.js`: Recursión optimizada con memoización
  - Reduce complejidad de O(2^n) a O(n)
  - Demuestra técnicas de optimización

Objetivo Pedagógico

Comprender las ventajas y desventajas de la recursión frente a iteración, y técnicas de optimización como memoización.

---

3. **Módulo Sorting**

Algoritmos Implementados

Algoritmo	Complejidad	Tipo	Archivo
Selection Sort	O(n²)	Comparación	`selectionSort.js`
Merge Sort	O(n log n)	Divide y Conquista	`mergesort.js`
Quick Sort	O(n log n)*	Divide y Conquista	`quicksort.js`

Merge Sort - Ejemplo

```javascript
function merge(leftA, rightA) {
// Combina dos arrays ordenados
var results = [], leftIndex = 0, rightIndex = 0;
while (leftIndex < leftA.length && rightIndex < rightA.length) {
if (leftA[leftIndex] < rightA[rightIndex]) {
results.push(leftA[leftIndex++]);
} else {
results.push(rightA[rightIndex++]);
}
}
return
results.concat(leftA.slice(leftIndex)).concat(rightA.slice(rightIndex))
;
}

function mergeSort(array) {
if (array.length < 2) return array;
```

```
    var midpoint = Math.floor(array.length / 2);
    var leftArray = array.slice(0, midpoint);
    var rightArray = array.slice(midpoint);
    return merge(mergeSort(leftArray), mergeSort(rightArray));
}
```

Objetivo Pedagógico
Comparar estrategias de ordenamiento, entender trade-offs entre complejidad y implementación, y aplicar paradigmas como "divide y conquista".

🔍 Análisis de Contenido

Distribución de Temas

```
Semana 13 (100%)
├── Sistema EduTrack (60%)
│   ├── Gestión de Estudiantes y Clases (20%)
│   ├── Funcionalidad de Asistencia (15%)
│   ├── Funcionalidad de Participación (15%)
│   └── Reportes y Dashboard (10%)
│
└── Algoritmos (40%)
    ├── Recursión (20%)
    │   ├── Conceptos Básicos (7%)
    │   └── Fibonacci (Iterativo vs Recursivo vs Optimizado) (13%)
    │
    └── Sorting (20%)
        ├── Selection Sort (5%)
        ├── Merge Sort (8%)
        └── Quick Sort (7%)
```

Progresión de Dificultad

1. **Introducción (EduTrack)**: Arquitectura de sistemas completos
2. **Fundamental (Recursión Básica)**: Conceptos de pila y llamadas recursivas

3. **Intermedio (Fibonacci)**: Comparación de enfoques (iterativo vs recursivo)
4. **Avanzado (Optimización)**: Memoización y técnicas de mejora de performance
5. **Avanzado (Sorting)**: Algoritmos O(n²) vs O(n log n), divide y conquista

Habilidades Desarrolladas

✅ **Backend/Lógica**
- Diseño de clases y sistemas orientados a objetos
- Estructuras de datos complejas
- Implementación de algoritmos eficientes
- Memoización y técnicas de optimización

✅ **Frontend/UI**
- Desarrollo de interfaces responsivas
- Manipulación del DOM
- Manejo de eventos y formularios
- Integración con animaciones

✅ **Bases de Datos**
- Modelado de entidades
- Esquemas de almacenamiento
- Relaciones entre tablas

✅ **Análisis**
- Complejidad computacional (Big O)
- Trade-offs entre estrategias
- Optimización de código

🎓 Conexiones Pedagógicas

Relación con Semanas Anteriores
- **Semana 1-9**: Estructuras de datos (arrays, linked lists, trees, graphs, hash tables)
- **Semana 10**: Árboles binarios y sets
- **Semana 11-12**: Aplicaciones web interactivas
- **Semana 13**: ⭐ **Integración**: Usa estructuras de datos en una aplicación real + algoritmos fundamentales

Relación con Semanas Posteriores

- Proporciona base para algoritmos más complejos
- Demuestra cómo aplicar estructuras de datos en sistemas reales
- Introduce optimización (importante para escalabilidad)

📚 Puntos Clave para Aprender

1. **EduTrack**: Cómo estructurar una aplicación educativa completa
2. **Recursión**: Cuándo usar recursión vs iteración y cómo optimizar
3. **Sorting**: Comparación de algoritmos y sus características
4. **Performance**: Cómo analizar y mejorar la complejidad computacional
5. **Integración**: Usar múltiples conceptos en un proyecto cohesivo

---

🚀 Cómo Usar Este Módulo

Para Ejecutar EduTrack

1. Abrir `index.html` en el navegador
2. Sistema carga datos de ejemplo automáticamente
3. Interactuar con la interfaz para registrar asistencia y participación
4. Ver reportes en tiempo real

Para Estudiar Algoritmos

1. Revisar comentarios en cada archivo de Recursion/ y Sorting/
2. Comparar complejidad: ejecutar con diferentes tamaños de entrada
3. Modificar código para experimentar con optimizaciones
4. Usar herramientas del navegador (DevTools) para medir performance

---

📖 Archivos de Documentación Relacionados

- README.md - Descripción completa de EduTrack
- database.md - Estructura de base de datos
- design.md - Guía visual y UX
- processes.md - Flujos de procesos

Última actualización: Semana 13, 2025

Nivel: Intermedio-Avanzado

Duración estimada: 10-15 horas de estudio y práctica

SEMANA 14

Análisis - Semana 14: Aplicación ToDo con CRUD

📋 Información General

Semana: 14

Tema: Operaciones CRUD (Crear, Leer, Actualizar, Eliminar) con almacenamiento local

Lenguajes: JavaScript (ES6+), HTML5, CSS3

Objetivo: Desarrollar una aplicación web interactiva para gestionar tareas con funcionalidad completa de CRUD utilizando localStorage

---

📁 Estructura de Archivos

```
Semana14/
├── app.js                # Lógica de la aplicación (CRUD y eventos)
├── index.html            # Estructura HTML de la interfaz
├── styles.css            # Estilos CSS de la aplicación
├── notas.md              # Notas iniciales del proyecto
├── .vscode/
│   └── settings.json     # Configuración del editor
└── demo/
    └── objetos.js        # Ejemplo básico de objetos en JavaScript
```

🎯 Objetivo Principal

Crear una aplicación web funcional de tareas (ToDo) que permita:

- Crear (C): Registrar nuevas tareas con título y descripción
- Riso (Derecha): Consultar tareas con búsqueda y filtrado
- Actualizar (U): Editar tareas existentes mediante modal
- Eliminar (D): Borrar tareas de forma permanente

Implementar persistencia de datos usando localStorage del navegador para que las tareas se mantengan entre sesiones.

📝 Descripción Detallada

app.js - Módulo Principal (188 líneas)

Implementa la funcionalidad CRUD en un patrón IIFE (Immediate Invoked Function Expression) para encapsular la lógica:

Variables y Configuración:

- `STORAGE_KEY`: Clave única para localStorage (`'todo_semana14_tasks_v1'`)

`tasks`: Array que almacena objetos de tareas con estructura:

```
{
  id: string (timestamp),
  title: string,
  description: string,
  done: boolean,
  createdAt: string (ISO date)
}
```

-

Funciones Principales:

Función	Descripción
`loadTasks()`	Carga tareas desde localStorage al iniciar
`saveTasks()`	Guarda tareas en localStorage (persiste datos)
`bindEvents()`	Vincula eventos a elementos del DOM
`onAddTask()`	Crea una nueva tarea
`renderTasks()`	Renderiza lista de tareas con filtrado y búsqueda

`openEdit()` / `closeEdit()`	Abre y cierra modal de edición
`onSaveEdit()`	Guarda cambios de tareas editadas
`removeTask()`	Elimina una tarea con confirmación
`toggleDone()`	Marca/desmarca una tarea como completada

index.html - Interfaz (73 líneas)

Estructura HTML semántica con dos paneles principales:

Sección 1 - Registro de Registro:

- Formulario con campos para título y descripción
- Botones para registrar y limpiar

Sección 2 - Consultar Tareas:

- Campo de búsqueda en tiempo real
- Dropdown de filtro (Todas/Pendientes/Completadas)
- Lista de tareas dinámica con acciones (Editar/Marcar/Eliminar)

Modal de Edición:

- Diálogo para editar título, descripción y estado de completado
- Cierra con botón cancelar o tecla Escape

styles.css - Estilos (50 líneas)

Diseño responsivo y moderno:

- Paleta de colores: Variables CSS personalizadas (--bg, --acento, --peligro)
- Distribución: Grid responsivo (1 columna en móvil, 2 columnas en desktop)
- Componentes: Paneles, formularios, botones, modal con overlay
- Tipografía: Segoe UI, Roboto, fuentes fallback

demo/objetos.js - Ejemplo Educativo

Breve demostración del concepto de objetos en JavaScript como estructura de datos para almacenar propiedades relacionadas.

🔍 Análisis de Contenido

Patrones y Conceptos Implementados

1. IIFE (Cierre):
    - Encapsulación de variables globales para evitar contaminación del scope global
    - `tasks` Array es privado dentro del módulo
2. Manipulación del DOM:
    - Selección de elementos con `getElementById`
    - Creación dinámica de elementos (`createElement`)
    - Eventos: , , , , `submitinputchangeclickkeydown`
3. API de LocalStorage:
    - Persistencia de datos en JSON
    - Lectura/escritura con manejo de excepciones
    - Sincronización bidireccional: aplicación ↔ navegador
4. Operaciones de Array:
    - `.push()` para agregar tareas
    - `.filter()` para búsqueda y filtrado
    - `.find()` para localizar tareas por ID
    - `.findIndex()` para actualizar índices
5. Búsqueda y Filtrado:
    - Búsqueda por substring (insensible a mayúsculas y mayúsculas)
    - Filtrado por estado (completada/pendiente)
    - Renderizado condicional
6. Mejoras en UX:
    - Modal accesible con confirmación de eliminación
    - Emoji visual (✅) para tareas completadas
    - Mensaje "No hay tareas" cuando está vacío
    - Soporte para cerrar modal con tecla Escape
    - Atributos ARIA para accesibilidad

Flujo de Datos

`Carga (init)`

```
    ↓
loadTasks() → localStorage → tasks array
    ↓
renderTasks() → DOM
    ↓
Usuario interactúa (agregar/editar/eliminar)
    ↓
Modificar array + saveTasks()
    ↓
renderTasks() actualiza vista
```

Características Destacadas

✅ CRUD Completo - Todas las operaciones de datos implementadas

✅ Persistencia - Datos se mantienen entre sesiones

✅ Búsqueda en Tiempo Real - Filtrado instantáneo mientras escribe

✅ Filtrados Múltiples - Por estado (todo/pendiente/completado)

✅ Modal de Edición - Interfaz dedicada para actualizar tareas

✅ Confirmación de Eliminación - Previene borrados accidentales

✅ Responsivo - Funciona en dispositivos móviles y desktop

✅ Accesible - Atributos ARIA y navegación por teclado

🚀 Cómo Usar

1. Abrir en un navegador`index.html`
2. Ingresar título (requerido) y descripción (opcional)
3. Hacer clic en "Registrar" para agregar tarea
4. Usar búsqueda/filtros para consultar tareas
5. Hacer clic en "Editar" para modificar una tarea
6. Marcar como completada o eliminar según sea necesario

💡 Concepto Educativo

Esta semana enseña el ciclo completo de desarrollo de una aplicación web funcional, consolidando:

- Manipulación del DOM

- Gestión de estado con arrays y objetos
- Persistencia de datos (localStorage)
- Patrones de encapsulación (IIFE)
- Interfaz de usuario interactiva y responsiva

Es un ejemplo práctico de cómo las estructuras de datos (arrays, objetos) se utilizan en aplicaciones reales para almacenar y gestionar información.

Análisis - Semana 14: Aplicación ToDo con CRUD

📋 Información General

Semana: 14
Tema: Operaciones CRUD (Create, Read, Update, Delete) con almacenamiento local
Lenguajes: JavaScript (ES6+), HTML5, CSS3
Objetivo: Desarrollar una aplicación web interactiva para gestionar tareas con funcionalidad completa de CRUD utilizando localStorage

---

📁 Estructura de Archivos

```
Semana14/
├── app.js              # Lógica de la aplicación (CRUD y eventos)
├── index.html          # Estructura HTML de la interfaz
├── styles.css          # Estilos CSS de la aplicación
├── notas.md            # Notas iniciales del proyecto
├── .vscode/
│   └── settings.json   # Configuración del editor
└── demo/
    └── objetos.js      # Ejemplo básico de objetos en JavaScript
```

🎯 Objetivo Principal

Crear una aplicación web funcional de tareas (ToDo) que permita:
- **Crear (C):** Registrar nuevas tareas con título y descripción
- **Leer (R):** Consultar tareas con búsqueda y filtrado
- **Actualizar (U):** Editar tareas existentes mediante modal
- **Eliminar (D):** Borrar tareas de forma permanente

Implementar persistencia de datos usando **localStorage** del navegador para que las tareas se mantengan entre sesiones.

📝 Descripción Detallada

app.js - Módulo Principal (188 líneas)

Implementa la funcionalidad CRUD en un patrón IIFE (Immediately Invoked Function Expression) para encapsular la lógica:

Variables y Configuración:
- `STORAGE_KEY`: Clave única para localStorage (`'todo_semana14_tasks_v1'`)
- `tasks`: Array que almacena objetos de tareas con estructura:

```javascript
{
  id: string (timestamp),
  title: string,
  description: string,
  done: boolean,
  createdAt: string (ISO date)
}
```

Funciones Principales:

Función	Descripción
`loadTasks()`	Carga tareas desde localStorage al iniciar
`saveTasks()`	Guarda tareas en localStorage (persiste datos)
`bindEvents()`	Vincula eventos a elementos del DOM
`onAddTask()`	Crea una nueva tarea
`renderTasks()`	Renderiza lista de tareas con filtrado y búsqueda
`openEdit()` / `closeEdit()`	Abre y cierra modal de edición
`onSaveEdit()`	Guarda cambios de tareas editadas
`removeTask()`	Elimina una tarea con confirmación
`toggleDone()`	Marca/desmarca una tarea como completada

index.html - Interfaz (73 líneas)

Estructura HTML semántica con dos paneles principales:

Sección 1 - Registrar Tarea:
- Formulario con campos para título y descripción
- Botones para registrar y limpiar

Sección 2 - Consultar Tareas:

- Campo de búsqueda en tiempo real
- Dropdown de filtro (Todas/Pendientes/Completadas)
- Lista de tareas dinámica con acciones (Editar/Marcar/Eliminar)

Modal de Edición:
- Diálogo para editar título, descripción y estado de completado
- Cierra con botón cancelar o tecla Escape

styles.css - Estilos (50 líneas)

Diseño responsivo y moderno:
- **Paleta de colores:** Variables CSS personalizadas (--bg, --accent, --danger)
- **Layout:** Grid responsivo (1 columna en móvil, 2 columnas en desktop)
- **Componentes:** Paneles, formularios, botones, modal con overlay
- **Tipografía:** Segoe UI, Roboto, fuentes fallback

demo/objetos.js - Ejemplo Educativo

Breve demostración del concepto de objetos en JavaScript como estructura de datos para almacenar propiedades relacionadas.

🔍 Análisis de Contenido

Patrones y Conceptos Implementados

1. **IIFE (Closure):**
- Encapsulación de variables globales para evitar contaminación del scope global
- `tasks` array es privado dentro del módulo

2. **DOM Manipulation:**
- Selección de elementos con `getElementById`
- Creación dinámica de elementos (`createElement`)
- Eventos: `submit`, `input`, `change`, `click`, `keydown`

3. **LocalStorage API:**
- Persistencia de datos en JSON
- Lectura/escritura con manejo de excepciones
- Sincronización bidireccional: aplicación ↔ navegador

4. **Operaciones de Array:**
   - `.push()` para agregar tareas
   - `.filter()` para búsqueda y filtrado
   - `.find()` para localizar tareas por ID
   - `.findIndex()` para actualizar índices

5. **Búsqueda y Filtrado:**
   - Búsqueda por substring (case-insensitive)
   - Filtrado por estado (completada/pendiente)
   - Renderizado condicional

6. **UX Enhancements:**
   - Modal accesible con confirmación de eliminación
   - Emoji visual (✅) para tareas completadas
   - Mensaje "No hay tareas" cuando está vacío
   - Soporte para cerrar modal con tecla Escape
   - Atributos ARIA para accesibilidad

Flujo de Datos

```
Carga (init)
    ↓
loadTasks() → localStorage → tasks array
    ↓
renderTasks() → DOM
    ↓
Usuario interactúa (agregar/editar/eliminar)
    ↓
Modificar array + saveTasks()
    ↓
renderTasks() actualiza vista
```

Características Destacadas

✅ **CRUD Completo** - Todas las operaciones de datos implementadas
✅ **Persistencia** - Datos se mantienen entre sesiones
✅ **Búsqueda en Tiempo Real** - Filtrado instantáneo mientras escribe
✅ **Filtrados Múltiples** - Por estado (todo/pendiente/completado)
✅ **Modal de Edición** - Interfaz dedicada para actualizar tareas
✅ **Confirmación de Eliminación** - Previene borrados accidentales

✅ **Responsivo** - Funciona en dispositivos móviles y desktop
✅ **Accesible** - Atributos ARIA y navegación por teclado

🚀 Cómo Usar

1. Abrir `index.html` en un navegador
2. Ingresar título (requerido) y descripción (opcional)
3. Hacer clic en "Registrar" para agregar tarea
4. Usar búsqueda/filtros para consultar tareas
5. Hacer clic en "Editar" para modificar una tarea
6. Marcar como completada o eliminar según sea necesario

💡 Concepto Educativo

Esta semana enseña el **ciclo completo de desarrollo de una aplicación web funcional**, consolidando:

- Manipulación del DOM
- Gestión de estado con arrays y objetos
- Persistencia de datos (localStorage)
- Patrones de encapsulación (IIFE)
- Interfaz de usuario interactiva y responsiva

Es un ejemplo práctico de cómo las estructuras de datos (arrays, objetos) se utilizan en aplicaciones reales para almacenar y gestionar información.

SEMANA 15

Análisis - Semana 15

🔧 Lenguaje

JavaScript (ES6+)

📁 Estructura de Archivos

```
Semana15/
├── code.js                    # Ejemplos de funciones y conceptos generales
├── graph/
│   ├── graph.js               # Implementación de clase Graph (lista adyacente)
│   └── graph_2.js             # Representaciones alternativas de grafos
├── hashTable/
│   └── hashTable.js           # Implementación de tabla hash con métodos básicos
└── linkedList/
    ├── singly.js              # Implementación de lista enlazada simple
    └── dobly.js               # Implementación de lista enlazada doble
```

🎯 Objetivo

Proporcionar implementaciones de estructuras de datos fundamentales en JavaScript, enfocándose en:

- Grafos (representaciones y operaciones básicas)
- Tablas Hash (almacenamiento clave-valor con resolución de colisiones)
- Listas Enlazadas (simples y dobles)

Complementado con ejemplos de conceptos generales de JavaScript como scope, condicionales y operadores.

📝 Descripción General

Semana 15 es un módulo de consolidación que reúne cuatro estructuras de datos esenciales en programación. Cada estructura se implementa como una

clase con métodos para operaciones fundamentales (insertar, eliminar, acceder a datos).

El contenido está orientado a desarrolladores que necesitan comprender cómo funcionan internamente estas estructuras de datos, siendo útil para:

- Entrevistas técnicas
- Optimización de algoritmos
- Construcción de aplicaciones eficientes

🔍 Análisis de Contenido

1. code.js - Conceptos Generales

Contiene ejemplos educativos sobre:

- Alcance de variables: Diferencia entre variables globales y locales
- Funciones condicionantes: Clasificación de números (unidades, decenas, centenas)
- Sentencias switch: Procesamiento de respuestas
- Conversor de números a palabras: Convierte números (0-1,000,000) a su representación en español

Ejemplo:

```
function determinaValor(num) {
    if (num > 0 && num <= 10) {
        return "El número es una unidad";
    }
    // ... más condiciones
}
```

---

2. gráfico/ - Grafos

graph_2.js

Muestra 4 representaciones diferentes de un mismo grafo:

Representación	Tipo	Ventaja
Lista de aristas	Array de aristas	Simple, compacto
Lista adyacente	Array/Objeto de adyacencia	Eficiente para grafos dispersos
Matriz adyacente	Matriz 2D	Acceso O(1) entre nodos
Matriz basada en objetos	Objeto con matriz	Notación clara

Ejemplo de grafo:

```
  2 - 0
   / \
  1 - 3
```

graph.js
Implementación de clase Graph con:

- `constructor()`: Inicializa estructura
- `addVertex(node)`: Añade un vértice
- `addEdge(node1, node2)`: Conecta dos vértices (grafo no dirigido)

---

3. hashTable/ - Tabla Hash

hashTable.js implementa una tabla hash con manejo de colisiones por encadenamiento:

Métodos:

- `hashMethod(key)`: Función hash que calcula índice basado en caracteres
- `set(key, value)`: Inserta/actualiza un par clave-valor
- `get(key)`: Recupera un valor por clave
- `delete(key)`: Elimina un par clave-valor
- `getAllKeys()`: Retorna todas las claves almacenadas

Características:

- Usa arrays internos para resolver colisiones
- Complejidad O(1) promedio para operaciones

---

4. linkedList/ - Listas Enlazadas

singly.js - Lista Enlazada Simple

Estructura de nodo con referencia a siguiente ():`next`

Métodos:

- `append(value)`: Añade elemento al final
- `prepend(value)`: Añade elemento al inicio
- `insert(index, value)`: Inserta en posición específica
- `remove(index)`: Elimina elemento en índice
- `getTheIndex(index)`: Busca nodo en posición

Ventajas: Memoria dinámica, inserción/eliminación O(1) en extremos

dobly.js - Lista Enlazada Doble

Cada nodo tiene referencias tanto a siguiente como a anterior (y):`prevnext`

Métodos adicionales:

- Permite navegación bidireccional
- Más memoria pero mejor flexibilidad
- Métodos similares a singly con adaptaciones para `prev`

Diferencia clave:

```
// Singly: node.next
// Doubly: node.next y node.prev
```

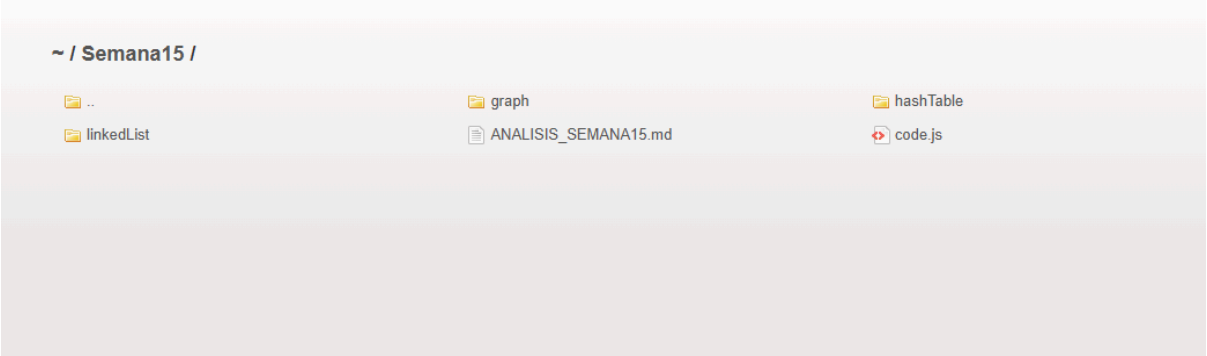
💡 Casos de Uso

Estructura	Ideal Para

Grafo	Redes sociales, rutas, relaciones complejas
Tabla hash	Búsquedas rápidas, caché, diccionarios
LinkedList Simple	Pilas, colas, secuencias dinámicas
LinkedList Doble	Navegación bidireccional, editores de texto

🚀 Conclusión

Semana 15 proporciona una base sólida en estructuras de datos clásicas, implementadas desde cero sin dependencias externas. Es ideal para comprender cómo funcionan internamente y para prepararse en desarrollo de software avanzado.



Análisis - Semana 15

🔧 Lenguaje

JavaScript (ES6+)

📁 Estructura de Archivos

```
Semana15/
├── code.js                 # Ejemplos de funciones y conceptos generales
├── graph/
│   ├── graph.js            # Implementación de clase Graph (lista adyacente)
│   └── graph_2.js          # Representaciones alternativas de grafos
├── hashTable/
│   └── hashTable.js        # Implementación de tabla hash con métodos básicos
└── linkedList/
    ├── singly.js           # Implementación de lista enlazada simple
    └── dobly.js            # Implementación de lista enlazada doble
```

🎯 Objetivo

Proporcionar implementaciones de **estructuras de datos fundamentales** en JavaScript, enfocándose en:

- Grafos (representaciones y operaciones básicas)
- Tablas Hash (almacenamiento clave-valor con resolución de colisiones)
- Listas Enlazadas (simples y dobles)

Complementado con ejemplos de conceptos generales de JavaScript como scope, condicionales y operadores.

📝 Descripción General

Semana 15 es un módulo de consolidación que reúne cuatro estructuras de datos esenciales en programación. Cada estructura se implementa como una clase con métodos para operaciones fundamentales (insertar, eliminar, acceder a datos).

El contenido está orientado a **desarrolladores que necesitan comprender cómo funcionan internamente estas estructuras de datos**, siendo útil para:
- Entrevistas técnicas
- Optimización de algoritmos
- Construcción de aplicaciones eficientes

🔍 Análisis de Contenido

1. **code.js** - Conceptos Generales
Contiene ejemplos educativos sobre:
- **Scope (Alcance de variables)**: Diferencia entre variables globales y locales
- **Funciones condicionantes**: Clasificación de números (unidades, decenas, centenas)
- **Switch statements**: Procesamiento de respuestas
- **Conversor de números a palabras**: Convierte números (0-1,000,000) a su representación en español

Ejemplo:
```javascript
function determinaValor(num) {
    if (num > 0 && num <= 10) {
    return "El número es una unidad";
                }
    // ... más condiciones
                }
```

---

2. **graph/** - Grafos

graph_2.js
Muestra **4 representaciones diferentes** de un mismo grafo:

Representación	Tipo	Ventaja
Edge List	Array de aristas	Simple, compacto
Adjacent List	Array/Objeto de adyacencia	Eficiente para grafos dispersos
Adjacent Matrix	Matriz 2D	Acceso O(1) entre nodos
Object-based Matrix	Objeto con matriz	Notación clara

Ejemplo de grafo:

```
 2 - 0
  / \
1 - 3
```

graph.js

Implementación de clase Graph con:
- `constructor()`: Inicializa estructura
- `addVertex(node)`: Añade un vértice
- `addEdge(node1, node2)`: Conecta dos vértices (grafo no dirigido)

---

3. **hashTable/** - Tabla Hash

hashTable.js implementa una tabla hash con manejo de colisiones por encadenamiento:

Métodos:
- `hashMethod(key)`: Función hash que calcula índice basado en caracteres
- `set(key, value)`: Inserta/actualiza un par clave-valor
- `get(key)`: Recupera un valor por clave
- `delete(key)`: Elimina un par clave-valor
- `getAllKeys()`: Retorna todas las claves almacenadas

Características:
- Usa arrays internos para resolver colisiones
- Complejidad O(1) promedio para operaciones

---

4. **linkedList/** - Listas Enlazadas

singly.js - Lista Enlazada Simple

Estructura de nodo con referencia a siguiente (`next`):

Métodos:

- `append(value)`: Añade elemento al final
- `prepend(value)`: Añade elemento al inicio
- `insert(index, value)`: Inserta en posición específica
- `remove(index)`: Elimina elemento en índice
- `getTheIndex(index)`: Busca nodo en posición

Ventajas: Memoria dinámica, inserción/eliminación O(1) en extremos

dobly.js - Lista Enlazada Doble

Cada nodo tiene referencias tanto a siguiente como a anterior (`prev` y `next`):

Métodos adicionales:

- Permite navegación bidireccional
- Más memoria pero mejor flexibilidad
- Métodos similares a singly con adaptaciones para `prev`

Diferencia clave:

```javascript
// Singly: node.next
// Doubly: node.next y node.prev
```

💡 Casos de Uso

Estructura	Ideal Para
Grafo	Redes sociales, rutas, relaciones complejas
Hash Table	Búsquedas rápidas, caché, diccionarios
LinkedList Simple	Pilas, colas, secuencias dinámicas
LinkedList Doble	Navegación bidireccional, editores de texto

🚀 Conclusión

Semana 15 proporciona una **base sólida en estructuras de datos clásicas**, implementadas desde cero sin dependencias externas. Es ideal para comprender cómo funcionan internamente y para prepararse en desarrollo de software avanzado.

SEMANA 16

Análisis Semana 16 - Estructuras de Datos

Lenguaje

- JavaScript (ES6+)
- HTML5
- CSS3

Estructura de Archivos

```
Semana16/
├── index.html          # Interfaz web principal
├── index.js            # Lógica de la aplicación (Pokemon API)
├── index.css           # Estilos de la aplicación
├── queue/
│   └── queue.js        # Implementación de Cola (Queue)
├── stack/
│   └── stack.js        # Implementación de Pila (Stack)
└── tree/
    └── tree.js         # Implementación de Árbol Binario de Búsqueda (BST)
```

Objetivo

Proporcionar una introducción práctica a tres estructuras de datos fundamentales en informática:

1. Cola (Cola): Estructura FIFO (Primero en entrar, primero en salir)
2. Pila (Stack): Estructura LIFO (Último en entrar, primero en salir)
3. Árbol Binario de Búsqueda: Estructura jerárquica para búsqueda eficiente

Además, implementar una aplicación web interactiva que consume la API de Pokémon, permitiendo filtrar Pokémon por tipo.

Descripción General

Componentes Web

index.html - Interfaz de Usuario

- Estructura HTML semántica en español
- Barra de navegación con botones para filtrar por tipo de Pokémon (Normal, Fuego, Agua, Planta, Eléctrico, Hielo, Lucha, Veneno, Tierra, Volador, Psíquico, Bicho, Roca, Fantasma, Dragón, Oscuro, Acero, Hada)
- Sección principal (main) donde se renderizarán las tarjetas de Pokémon
- Diseño responsive preparado para una galería de elementos

index.css - Estilos

- Fuente personalizada: "Bitcount Prop Single" de Google Fonts
- Fondo oscuro (#222) con texto blanco
- Grid layout responsive con para tarjetas de 350px mínimo`auto-fit`
- Estilos para botones con colores específicos por tipo de Pokémon
- Flexbox para centrado y espaciado

index.js - Lógica de la Aplicación

- Estructura base para consumir API de Pokémon
- Funciones principales:
  - `apiRequest()`: Realiza solicitudes a la API
  - `paintPokemon()`: Renderiza los datos de Pokémon en la UI
  - `paintPokemonForType()`: Filtra y muestra Pokémon por tipo
- Listeners de eventos para DOMContentLoaded y clicks en botones
- Uso de DocumentFragment para optimizar manipulación del DOM

Estructuras de Datos

queue/queue.js - Implementación de Cola

```
Clase Node: Nodo individual con valor y referencia al siguiente
Clase Queue:
  - Propiedades: first (inicio), last (final), length (tamaño)
  - enqueue(value): Añade elementos al final
```

- dequeue(): Extrae elementos del inicio
- peek(): Consulta el primer elemento sin remover
- Aplicación FIFO: Ideal para sistemas de espera

stack/stack.js - Implementación de Pila

Clase Node: Nodo individual con valor y referencia al siguiente
Clase Stack:
- Propiedades: top (cima), bottom (base), length (tamaño)
- push(value): Añade elementos en la cima
- pop(): Extrae elemento de la cima
- peek(): Consulta el elemento superior sin remover
- Aplicación LIFO: Ideal para deshacer/rehacer, parseo de expresiones

árbol/tree.js - Árbol Binario de Búsqueda (BST)

Clase Node: Nodo con left (izquierda), right (derecha), value
Clase BinarySearchTree:
- Propiedad: root (raíz del árbol)
- insert(value): Inserta valores manteniendo propiedad BST
- search(value): Búsqueda iterativa
- recursiveSearch(value, tree): Búsqueda recursiva
- Aplicación: Búsqueda eficiente O(log n) en casos balanceados

---

Análisis de Contenido

Propósito Educativo

Esta semana integra teoría de estructuras de datos con una aplicación práctica real:

1. Conceptos Clave Implementados:
    - Encapsulación: Clases con métodos privados y públicos
    - Abstracción: Interfaces limpias para operaciones (enqueue/dequeue, push/pop, insert/search)
    - Gestión de referencias: Punteros en listas enlazadas
    - Recorrido de árboles: Búsqueda recursiva e iterativa
2. Aplicación Pokémon:
    - Demuestra uso de APIs en JavaScript moderno

    - Implementa filtrado dinámico por tipo
    - Manejo de eventos del usuario
    - Renderizado eficiente con DocumentFragment
3. Complejidad Algorítmica:
    - Cola: Enqueue O(1), Dequeue O(1)
    - Acumular: Empujar O(1), Poblar O(1)
    - BST: Insertar O(log n), Buscar O (log n) - promedio

Puntos de Aprendizaje

- ✅ Entender diferencias entre FIFO y LIFO
- ✅ Implementar estructuras desde cero sin librerías
- ✅ Consumo de APIs externas (REST)
- ✅ Manipulación eficiente del DOM
- ✅ Búsqueda en estructuras jerárquicas

Estado del Código

- Incompleto: Las funciones , y están sin implementar`apiRequest()` `paintPokemon()` `paintPokemonForType()`
- Estructura lista: Todas las clases están completamente funcionales
- UI preparada: HTML y CSS listos para recibir datos dinámicos

---

Resumen

La Semana 16 presenta un proyecto integral que combina estructuras de datos fundamentales con desarrollo web moderno. Los estudiantes deben completar la lógica de consumo de API y renderizado, practicando así tanto estructuras de datos como integración con APIs reales.

Análisis Semana 16 - Estructuras de Datos

Lenguaje

- **JavaScript** (ES6+)
- **HTML5**
- **CSS3**

Estructura de Archivos

```
Semana16/
├── index.html          # Interfaz web principal
├── index.js            # Lógica de la aplicación (Pokemon API)
├── index.css           # Estilos de la aplicación
├── queue/
│   └── queue.js        # Implementación de Cola (Queue)
├── stack/
│   └── stack.js        # Implementación de Pila (Stack)
└── tree/
    └── tree.js         # Implementación de Árbol Binario de Búsqueda (BST)
```

Objetivo

Proporcionar una introducción práctica a **tres estructuras de datos fundamentales** en informática:

1. **Cola (Queue)**: Estructura FIFO (First In, First Out)
2. **Pila (Stack)**: Estructura LIFO (Last In, First Out)
3. **Árbol Binario de Búsqueda**: Estructura jerárquica para búsqueda eficiente

Además, implementar una aplicación web interactiva que consume la **API de Pokémon**, permitiendo filtrar Pokémon por tipo.

Descripción General

Componentes Web

index.html - Interfaz de Usuario
- Estructura HTML semántica en español
- Barra de navegación con botones para filtrar por tipo de Pokémon (Normal, Fire, Water, Grass, Electric, Ice, Fighting, Poison, Ground, Flying, Psychic, Bug, Rock, Ghost, Dragon, Dark, Steel, Fairy)
- Sección principal (main) donde se renderizarán las tarjetas de Pokémon
- Diseño responsive preparado para una galería de elementos

index.css - Estilos
- Fuente personalizada: "Bitcount Prop Single" de Google Fonts
- Fondo oscuro (#222) con texto blanco
- Grid layout responsive con `auto-fit` para tarjetas de 350px mínimo
- Estilos para botones con colores específicos por tipo de Pokémon
- Flexbox para centrado y espaciado

index.js - Lógica de la Aplicación
- Estructura base para consumir API de Pokémon
- Funciones principales:
  - `apiRequest()`: Realiza solicitudes a la API
  - `paintPokemon()`: Renderiza los datos de Pokémon en la UI
  - `paintPokemonForType()`: Filtra y muestra Pokémon por tipo
- Listeners de eventos para DOMContentLoaded y clicks en botones
- Uso de DocumentFragment para optimizar manipulación del DOM

Estructuras de Datos

queue/queue.js - Implementación de Cola
```javascript
Clase Node: Nodo individual con valor y referencia al siguiente
Clase Queue:
- Propiedades: first (inicio), last (final), length (tamaño)
- enqueue(value): Añade elementos al final
- dequeue(): Extrae elementos del inicio
- peek(): Consulta el primer elemento sin remover
- Aplicación FIFO: Ideal para sistemas de espera
```

stack/stack.js - Implementación de Pila
```javascript
Clase Node: Nodo individual con valor y referencia al siguiente
Clase Stack:
- Propiedades: top (cima), bottom (base), length (tamaño)
- push(value): Añade elementos en la cima
- pop(): Extrae elemento de la cima
- peek(): Consulta el elemento superior sin remover
- Aplicación LIFO: Ideal para deshacer/rehacer, parseo de expresiones
```

tree/tree.js - Árbol Binario de Búsqueda (BST)
```javascript
Clase Node: Nodo con left (izquierda), right (derecha), value
Clase BinarySearchTree:
- Propiedad: root (raíz del árbol)
- insert(value): Inserta valores manteniendo propiedad BST
- search(value): Búsqueda iterativa
- recursiveSearch(value, tree): Búsqueda recursiva
- Aplicación: Búsqueda eficiente O(log n) en casos balanceados
```

Análisis de Contenido

Propósito Educativo
Esta semana integra **teoría de estructuras de datos** con una **aplicación práctica real**:

1. **Conceptos Clave Implementados**:
   - **Encapsulación**: Clases con métodos privados y públicos

- **Abstracción**: Interfaces limpias para operaciones (enqueue/dequeue, push/pop, insert/search)
- **Gestión de referencias**: Punteros en listas enlazadas
- **Recorrido de árboles**: Búsqueda recursiva e iterativa

2. **Aplicación Pokémon**:
- Demuestra uso de APIs en JavaScript moderno
- Implementa filtrado dinámico por tipo
- Manejo de eventos del usuario
- Renderizado eficiente con DocumentFragment

3. **Complejidad Algorítmica**:
- **Queue**: Enqueue O(1), Dequeue O(1)
- **Stack**: Push O(1), Pop O(1)
- **BST**: Insert O(log n), Search O(log n) - promedio

Puntos de Aprendizaje
- ✅ Entender diferencias entre FIFO y LIFO
- ✅ Implementar estructuras desde cero sin librerías
- ✅ Consumo de APIs externas (REST)
- ✅ Manipulación eficiente del DOM
- ✅ Búsqueda en estructuras jerárquicas

Estado del Código
- **Incompleto**: Las funciones `apiRequest()`, `paintPokemon()` y `paintPokemonForType()` están sin implementar
- **Estructura lista**: Todas las clases están completamente funcionales
- **UI preparada**: HTML y CSS listos para recibir datos dinámicos

---

Resumen

La Semana 16 presenta un proyecto integral que combina **estructuras de datos fundamentales** con desarrollo web moderno. Los estudiantes deben completar la lógica de consumo de API y renderizado, practicando así tanto estructuras de datos como integración con APIs reales.

CURSO

https://developers.google.com/profile/badges/recognitions/learnings