

PORAFOLIO DE EVIDENCIAS

Angel Eduardo Lopez Garcia

Estructura de bases de
datos

Tabla de contenido

◆ ¿Qué hace la tarea en general?	6
■ 1. Función <code>saludar()</code>	6
✓ ¿Qué hace?	6
✓ ¿Para qué sirve en la clase?	7
■ 2. Función <code>teSaludo(nombre)</code>	7
✓ ¿Qué hace?	7
✓ ¿Para qué sirve?	7
■ 3. Función <code>fizzBuzz()</code>	7
✓ ¿Qué hace?	8
✓ ¿Para qué sirve esta función?	8
❗ Nota importante	8
■ ¿Qué se trabajó?	9
✓ Conectar HTML ↔ JavaScript	9
✓ Crear funciones básicas	9
✓ Parámetros en funciones	9
✓ Uso del ciclo <code>for</code> y operadores	9
¿Qué permite hacer este “array” creado a mano?	10
✓ 1. Guardar datos (<code>push()</code>)	10
✓ 2. Eliminar el último (<code>pop()</code>)	10
✓ 3. Obtener un elemento por índice (<code>get()</code>)	10
✓ 4. Eliminar un elemento en cualquier posición (<code>methodDelete(index)</code>)	10
✓ 5. Recorrer índices hacia atrás para agregar al inicio (<code>shiftIndexAdd()</code>)	10
✓ 6. Quitar el primer elemento (<code>remoteFirst()</code>)	10
¿Qué muestra al final?	11
En pocas palabras:	12
¿Qué nos muestra?	12
ARCHIVO HTML	12
¿Qué demuestra esta página?	12
✓ 1. Distintos niveles de encabezados (h1–h6)	12
✓ 2. Uso de <code>
</code> , <code><p></code> y <code><pre></code>	13

✓ 3. Listas ordenadas y desordenadas	13
✓ 4. Formatos de texto	13
✓ 5. Enlaces externos e internos.....	13
✓ 6. Mostrar imágenes	13
(nosotros.html)	13
◆ ¿Qué contiene?.....	14
✓ 2. Sirven para practicar navegación interna	14
✓ 3. La página principal demuestra todas las etiquetas básicas de HTML	14
✓ 4. Las otras dos páginas son páginas informativas complementarias.....	15
¿Qué enseña?.....	16
✓ 1. Uso de contenedores de Bootstrap.....	16
✓ 2. Cómo funciona el sistema Grid de Bootstrap	16
✓ 3. Cómo hacer diseños responsivos	17
✓ 4. Uso de archivos CSS externos.....	17
✓ 5. Uso de Bootstrap desde CDN	17
mistyle.css	17
📌 ¿Qué hace exactamente este CSS?.....	18
✓ 1. Da un color por defecto (pantallas pequeñas)	18
✓ 2. Cambia de color según cada breakpoint de Bootstrap	18
🎯 Propósito dentro de la Semana 3	18
RESUMEN GENERAL	19
📌 ¿Qué hace este archivo en conjunto?	19
✓ 1. Declara números de distintos tipos	19
✓ 2. Imprime una serie de números grandes en incrementos	19
✓ 3. Suma un número con el valor global number	20
✓ 4. Muestra una tabla de multiplicar ascendente	20
✓ 5. Muestra una tabla de multiplicar descendente.....	20
HTML.....	20
📌 ¿Qué muestra esta página?.....	21
✓ 1. Explica qué son los tipos de datos numéricos	21
✓ 2. Uso de la etiqueta <pre> para texto formateado.....	21

✓ 3. Muestra una tabla comparativa	21
✓ 4. Carga el archivo JavaScript	21
<input checked="" type="checkbox"/> Dom	22
📌 ¿Qué hace este código en conjunto?	22
✓ 1. Llena una lista HTML automáticamente	22
✓ 2. Saluda al usuario y cambia el título de la pestaña	22
✓ 3. Convierte una lista en encabezados <h4>.....	23
✓ 4. Usa un array de nombres ('alumnos').....	23
<input checked="" type="checkbox"/> index	24
📌 ¿Qué hace este archivo en conjunto?	24
✓ 1. Lee datos de un formulario	24
✓ 2. Convierte esos valores a números.....	24
✓ 3. Calcula la suma.....	25
✓ 4. Genera contenido HTML dinámico	25
✓ 5. Inserta el resultado en la página usando el DOM	25
<input checked="" type="checkbox"/> (HTML + JS)	25
📌 1. Práctica 1 – Llenar una lista usando innerHTML.....	25
📌 2. Práctica 2 – Solicitar un nombre y saludar	26
📌 3. Práctica 3 – Convertir párrafos <p> en encabezados <h4>	26
<input checked="" type="checkbox"/> INNER HTML.....	27
✓ 1. Mostrar variables y sus tipos.....	27
✓ 2. Manejo de un arreglo dinámico con inputs.....	28
✓ 3. Agregar elementos al arreglo y mostrarlos	28
◆ Paso 1: Leer el nombre desde un input.....	28
◆ Paso 2: Limpiar el input.....	28
◆ Paso 3: Guardar el nombre en el arreglo.....	28
◆ Paso 4: Limpiar la lista HTML.....	28
◆ Paso 5: Volver a imprimir todo lo que haya en el arreglo	28
(HTML + JS).....	29
✓ 1. Botón para mostrar variables básicas.....	29
✓ 2. Sección para manejar un arreglo (array)	30

✓ 3. Carga del JS	30
CODE	31
📌 ¿Qué hace el código en conjunto?	31
✓ 1. Muestra variables y tipos en pantalla.....	31
✓ 2. Maneja un arreglo simple (<code>datos []</code>).....	32
✓ 3. Trabaja con una estructura tipo “struct”	32
✓ 4. Arreglo de estructuras (<code>miEstructuraArreglo []</code>).....	33
✓ RESUMEN GENERAL — Semana 7	33
✓ 1. Mostrar tipos de datos en pantalla	34
✓ 2. Manejar un arreglo simple (<code>datos []</code>)	34
✓ 3. Manejar una estructura de datos (tipo “struct”).....	35
✓ 4. Arreglo de estructuras (structs dentro de un array)	35
✓ Semana (Listas Enlazadas Sencillas)	36
📌 ¿Qué es una lista enlazada simple?.....	37
📌 ¿Qué hace el código?	37
✓ 1. Crea una lista enlazada manualmente	37
✓ 2. Intenta recorrer la lista con <code>for (i of lista)</code>	37
✗ ¿Qué está MAL?	38
✓ Cómo DEBERÍA recorrerse (para que funcione)	38
(Listas Enlazadas Dobles).....	39
📌 ¿Qué hace el código en conjunto?	39
✓ 1 Se muestra un ejemplo de estructura simple (no usada)	39
✓ 2 Se define la clase <code>Nodes</code>	39
✓ 3 Se define la clase <code>myDoublyLinkedList</code>	40
★ Constructor	40
★ <code>add(value)</code>	40
★ <code>insert(index, value)</code>	40
★ <code>getTheIndex(index)</code>	41
★ <code>remove(index)</code>	41
★ <code>search()</code>	41
✓ 4 Se crea una lista y se agregan nodos	41

✓ (Estructuras de Datos Avanzadas en JavaScript)	42
DETALLE DE LO QUE HACE CADA PARTE	42
1 MyArray — Implementación manual de un arreglo	42
2 Doubly Linked List — Lista Enlazada Doble.....	43
3 Graph — Implementación de un grafo	43
4 HashTable — Tabla Hash	44
5 Queue — Cola (FIFO)	44
6 Stack — Pila (LIFO).....	44
7 Binary Search Tree — Árbol Binario de Búsqueda	45
★ ESTRUCTURAS DE DATOS AVANZADAS	45
A) ESTRUCTURAS DE DATOS AVANZADAS (Set, Diccionario, HashTable, Árboles, LinkedList).....	45
B) CONSUMO DE APIs (Proyecto Rick & Morty con HTML + JS + CSS)	45
A ESTRUCTURAS DE DATOS AVANZADAS	45
1 Set en JavaScript + operaciones de conjuntos.....	46
2 Dictionary — Implementación manual de un Diccionario	46
3 HashTable — Tabla Hash con direccionamiento abierto	46
4 BinaryTree — Árbol Binario de Búsqueda + Traversals	47
5 linkedList — Lista enlazada personalizada.....	47
B CONSUMO DE API — PROYECTO RICK & MORTY	47
1 Estructura HTML.....	48
2 JavaScript para consumir API.....	48
✓ Versión con fetch + .then ()	48
✓ Versión moderna con async/await	48
3 Renderizado de tarjetas de personajes	48
4 CSS avanzado estilo “Rick & Morty API”	48

Semana 1

La **Semana 1** introduce el uso básico de JavaScript dentro de un archivo HTML y enseña cómo crear funciones simples que interactúan con el usuario.

- ◊ ¿Qué hace la tarea en general?

1. Conecta HTML con JavaScript

— El HTML carga el archivo `code.js` para poder ejecutar funciones desde botones o eventos.

2. Muestra mensajes emergentes al usuario

— Una función (`saludar()`) muestra un mensaje tipo *alert* como bienvenida.
— Otra (`teSaludo(nombre)`) muestra un saludo personalizado usando un dato que escribe el usuario.

3. Ejecuta un ejercicio clásico de programación: FizzBuzz

— La función `fizzBuzz()` recorre una serie de números y, según ciertas condiciones, imprime en la consola:

- “Fizz”
- “Buzz”
- “FizzBuzz”
- O el número directamente

Esto es una práctica para aprender ciclos y condicionales.

4. Permite que el usuario interactúe con la página

— A través del HTML, seguramente hay botones o campos que permiten:

- Llamar a `saludar()`
- Ingresar un nombre y llamar a `teSaludo()`
- Ejecutar la función `fizzBuzz()` y ver el resultado en la consola

1. Función `saludar()`

```
function saludar() {
    alert("Hola, Bienvenidos, este ejemplo muestra como
incorporar JS en HTML");
}
```

¿Qué hace?

- Muestra un mensaje emergente (alerta) en el navegador.
- Es el ejemplo más básico para verificar que el JavaScript está conectado correctamente con el HTML.

✓ ¿Para qué sirve en la clase?

- Para enseñarte cómo crear una función.
- Cómo ejecutar código al presionar un botón en HTML.
- Cómo enviar un mensaje al usuario.

■ ■ ■ 2. Función teSaludo (nombre)

```
function teSaludo(nombre) {  
    alert("Hola " + nombre + ", Bienvenidos, este ejemplo  
muestra como incorporar JS en HTML");  
}
```

✓ ¿Qué hace?

- Recibe un **parámetro** llamado nombre.
- Muestra un mensaje personalizado usando ese nombre.

Ejemplo:

Si llamas teSaludo ("Angel"), aparece:

👉 “Hola Angel, Bienvenidos...”

✓ ¿Para qué sirve?

- Para practicar **parámetros dentro de funciones**.
- Para demostrar cómo hacer que la página responda según datos que coloca el usuario.
- Tu HTML seguramente tiene un <input> donde escribes el nombre.

■ ■ ■ 3. Función fizzBuzz ()

```
function fizzBuzz() {  
    for(let num=1; num<=100; num*=4) {  
        if(num % 3 === 0 && num % 5 === 0) {  
            console.log("FizzBuzz");  
        } else if(num % 3 === 0) {  
            console.log("Fizz");  
        } else if(num % 5 === 0) {  
            console.log("Buzz");  
        } else {  
            console.log(num);  
        }  
    }  
}
```

```
    }  
}  
  
✓ ¿Qué hace?
```

Es una variante del famoso problema **FizzBuzz**, que se usa para practicar ciclos y condiciones.

- Recorre números empezando en **1**
- Pero ojo: aquí **no suma 1**, sino que **multiplica por 4 cada vuelta**:

Vuelta	num
1	1
2	4
3	16
4	64
5	256 (ya no entra porque pasa de 100)

Luego aplica la lógica:

- ✓ Si el número es divisible entre **3 y 5**, imprime "FizzBuzz"
 - ✓ Si solo entre 3 → "Fizz"
 - ✓ Si solo entre 5 → "Buzz"
 - ✓ Si no, imprime el número
- ✓ ¿Para qué sirve esta función?

- Para practicar ciclos `for`
- Para practicar operadores lógicos y condicionales (`if`, `else if`)
- Para trabajar divisiones y el operador módulo `%`
- Para ver cómo usar `console.log()` en vez de solo `alert`

! Nota importante

El ciclo está raro a propósito (`num *= 4`).
Normalmente debería ser `num++`.

Esto está hecho para que aprendas a analizar cómo cambia una variable con diferentes operaciones.

¿Qué se trabajó?

Resumen claro:

- ✓ Conectar HTML ↔ JavaScript

Se enseña cómo vincular un archivo JS y cómo ejecutar funciones desde botones.

- ✓ Crear funciones básicas

Cómo se declaran, cómo se llaman, cómo mostrar mensajes.

- ✓ Parámetros en funciones

`teSaludo (nombre)` es para aprender cómo recibir datos desde HTML.

- ✓ Uso del ciclo `for` y operadores

La función `fizzBuzz ()` sirve para practicar:

- ciclos
- condicionales
- divisibilidad
- consola del navegador

Index.html

- ◆ *¿Qué hace la tarea en general?*
 1. **Conecta HTML con JavaScript**
 - El HTML carga el archivo `code.js` para poder ejecutar funciones desde botones o eventos.
 2. **Muestra mensajes emergentes al usuario**
 - Una función (`saludar ()`) muestra un mensaje tipo *alert* como bienvenida.
 - Otra (`teSaludo (nombre)`) muestra un saludo personalizado usando un dato que escribe el usuario.
 3. **Ejecuta un ejercicio clásico de programación: FizzBuzz**
 - La función `fizzBuzz ()` recorre una serie de números y, según ciertas condiciones, imprime en la consola:
 - “Fizz”
 - “Buzz”
 - “FizzBuzz”

- O el número directamente
Esto es una práctica para aprender ciclos y condicionales.

4. Permite que el usuario interactúe con la página

— A través del HTML, seguramente hay botones o campos que permiten:

- Llamar a `saludar()`
- Ingresar un nombre y llamar a `teSaludo()`
- Ejecutar la función `fizzBuzz()` y ver el resultado en la consola

Arrays

¿Qué permite hacer este “array” creado a mano?

Tu clase simula las operaciones básicas de un arreglo real:

✓ 1. Guardar datos (`push`)

Añade un nuevo elemento al final, igual que `array.push()` de JavaScript.

✓ 2. Eliminar el último (`pop`)

Quita el último elemento y lo devuelve.

✓ 3. Obtener un elemento por índice (`get`)

Permite leer algo como `array[2]` pero usando `get(2)`.

✓ 4. Eliminar un elemento en cualquier posición (`methodDelete(index)`)

Quita un elemento del medio y “recorre” los demás para llenar el hueco.

Ejemplo:

Miguel, Juan, Oscar
→ borro el índice 1 (Juan)
→ quedan Miguel, Oscar

✓ 5. Recorrer índices hacia atrás para agregar al inicio (`shiftIndexAdd`)

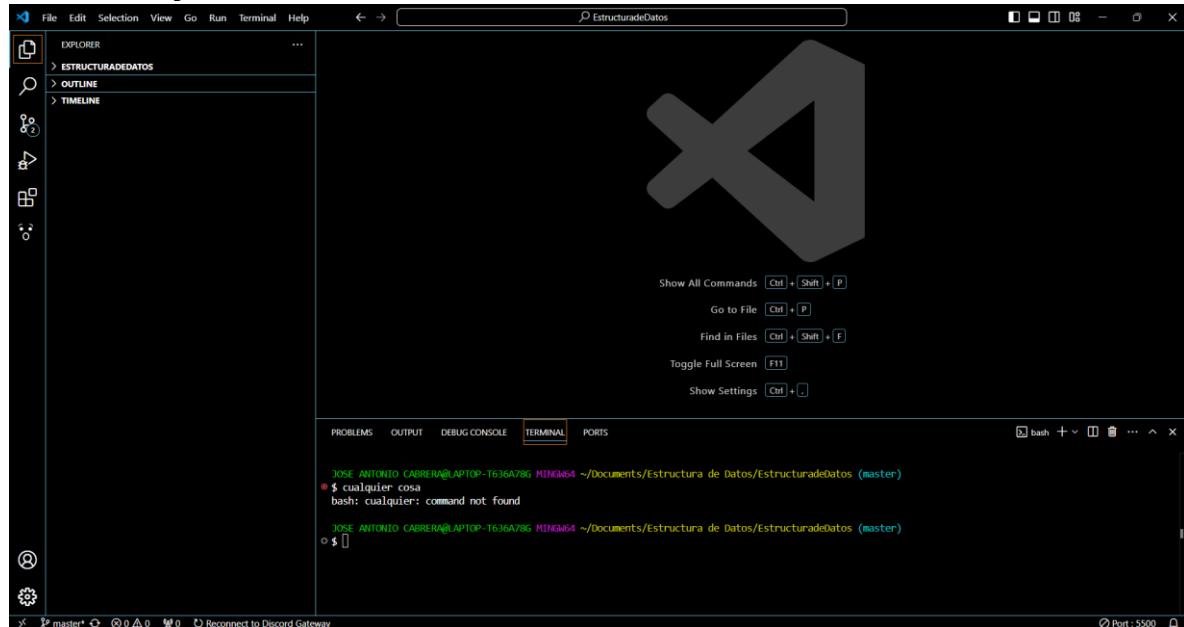
Simula `unshift()` de los arrays reales: meter algo al principio.

✓ 6. Quitar el primer elemento (`remoteFirsts`)

Simula `shift()`: elimina el índice 0 y recorre todo lo demás.

¿Qué muestra al final?

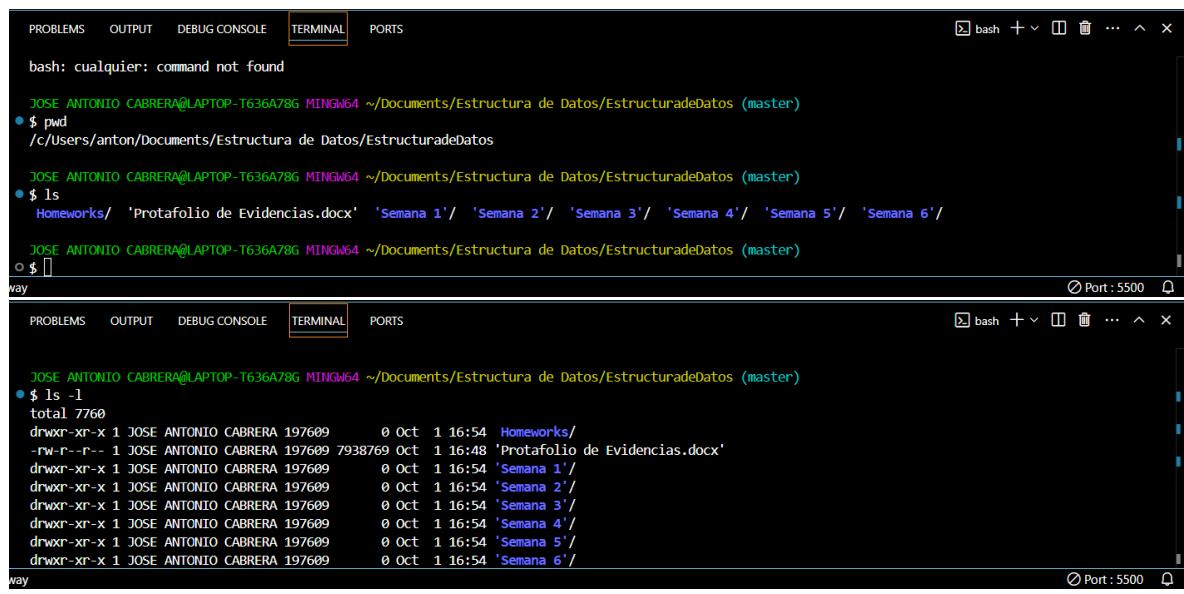
```
novo.push("Numero pi");
novo.push("Numero de euler");
novo.push("Miguel");
console.log(novo);
novo.methodDelete(1);
console.log(novo);
```



A screenshot of the Visual Studio Code interface. The title bar says "EstructuradeDatos". The left sidebar has icons for Explorer, Search, Outline, and Timeline. The main area is dominated by a large, solid gray 'X' shape. Below the 'X' are several keyboard shortcut keys: "Show All Commands" (Ctrl + Shift + P), "Go to File" (Ctrl + P), "Find in Files" (Ctrl + Shift + F), "Toggle Full Screen" (F11), and "Show Settings" (Ctrl + ,). At the bottom of the interface, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected and shows the following command-line session:

```
JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ cualquier cosa
bash: cualquier: command not found

JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ []
```



Three screenshots of a terminal window from VS Code, showing different command-line sessions. The first two screenshots are identical, showing the same error message as the previous one. The third screenshot shows a successful execution of the 'ls -l' command.

Terminal 1 (Top Two Screenshots):

```
bash: cualquier: command not found

JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ pwd
/c/Users/anton/Documents/Estructura de Datos/EstructuradeDatos

JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ ls
Homeworks/ 'Protafolio de Evidencias.docx' 'Semana 1'/ 'Semana 2'/ 'Semana 3'/ 'Semana 4'/ 'Semana 5'/ 'Semana 6'

JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ []
```

Terminal 2 (Bottom Screenshot):

```
way
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash: Port : 5500
JOSE ANTONIO CABRERA@LAPTOP-T636A78G MINGW64 ~/Documents/Estructura de Datos/EstructuradeDatos (master)
$ ls -l
total 7760
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 Homeworks/
-rw-r--r-- 1 JOSE ANTONIO CABRERA 197609 7938769 Oct 1 16:48 'Protafolio de Evidencias.docx'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 1'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 2'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 3'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 4'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 5'
drwxr-xr-x 1 JOSE ANTONIO CABRERA 197609 0 Oct 1 16:54 'Semana 6'
```

Semana 2

- Cómo crear varias páginas web dentro de un mismo sitio.
- Cómo navegar entre ellas usando enlaces.
- Cómo estructurar contenido básico en HTML.

En pocas palabras:

Es una página informativa que muestra un título, un texto descriptivo y un menú de navegación para ir a otras páginas del mismo sitio.

¿Qué nos muestra?

- Un título: "Acerca de Nosotros"
- Un breve texto que dice que es un sitio de ejemplo.
- Un menú con tres enlaces:
 - **Inicio (index.html)**
 - **Nosotros (nosotros.html)**
 - **Acerca de (acerca.html)**

Este menú sirve para practicar **cómo enlazar varias páginas entre sí**.

ARCHIVO HTML

Este archivo es una práctica completa de **HTML básico**, donde se muestran y se prueban muchas de las etiquetas fundamentales del lenguaje.

El propósito es **aprender a usar las etiquetas más comunes de HTML**, ver cómo se muestran en pantalla y entender su formato.

¿Qué demuestra esta página?

- ✓ 1. Distintos niveles de encabezados (h1–h6)

Se muestra cómo lucen los títulos grandes y pequeños.

✓ 2. Uso de
, <p> y <pre>

Para practicar saltos de línea, párrafos normales y texto con formato preservado.

✓ 3. Listas ordenadas y desordenadas

Se enseña cómo crear listas numeradas y con viñetas.

✓ 4. Formatos de texto

Ejemplos de:

- Negritas
- Cursivas
- Texto marcado
- Pequeño
- Tachado
- Subrayado
- Subíndices (H_2O)
- Superíndices (x^2)

✓ 5. Enlaces externos e internos

- Enlaces a Google, YouTube, Facebook, etc.
- Menú de navegación para cambiar entre páginas del mismo sitio:
 - index.html
 - nosotros.html
 - acerca.html

✓ 6. Mostrar imágenes

Carga varias imágenes desde la carpeta `assets/` con diferentes extensiones (.jpg, .png).

(`nosotros.html`)

Este archivo es otra página simple del mismo sitio web, diseñada para practicar **navegación interna entre páginas HTML**.

❖ ¿Qué contiene?

- Un título: “**Nosotros**”
- Un texto breve explicando que aquí se hablaría de la misión y visión del sitio.
- Un menú de navegación con enlaces a:
 - **Inicio (index.html)**
 - **Nosotros (esta misma página)**
 - **Acerca de (acerca.html)**

¿Que es lo que hace esto en conjunto?

Los tres archivos funcionan como **páginas diferentes de un mismo sitio**:

- **index.html** (la página grande con encabezados, listas, imágenes, etc.)
- **nosotros.html**
- **acerca.html**

Todas están conectadas entre sí mediante un menú de navegación.

✓ 2. Sirven para practicar navegación interna

Cada página contiene un menú:

```
<a href="index.html">Inicio</a>
<a href="nosotros.html">Nosotros</a>
<a href="acerca.html">Acerca de</a>
```

Esto te enseña cómo funcionan los enlaces entre páginas dentro de un sitio web.

✓ 3. La página principal demuestra todas las etiquetas básicas de HTML

El archivo grande te permite practicar:

- Encabezados (h1–h6)
- Párrafos
- Saltos de línea
- Texto con formato (negritas, cursivas, subrayado, tachado, subíndices, superíndices, etc.)
- Listas ordenadas y desordenadas
- Imágenes
- Enlaces externos e internos

Es una **demostración completa del HTML elemental**.

✓ 4. Las otras dos páginas son páginas informativas complementarias

- `nosotros.html` explica que ahí iría la misión/visión.
- `acerca.html` explica que es información del sitio.

Estas páginas están hechas para entender cómo crear **secciones separadas** dentro de un sitio.

The screenshot shows the GitHub interface for creating a new personal access token. The left sidebar has 'Personal access tokens' selected under 'Tokens'. The main area is titled 'New personal access token (classic)'. It includes fields for 'Note' (set to 'Estructura de Datos'), 'Expiration' (set to 'No expiration'), and a note about setting an expiration date for security. Below these, there's a 'Select scopes' section with various checkboxes for repository access, deployment status, public repositories, repository invitations, and security events. The 'repo' scope is checked by default.

The screenshot shows the GitHub interface for managing personal access tokens. The left sidebar has 'Personal access tokens' selected under 'Tokens'. The main area is titled 'Personal access tokens (classic)' and shows a list of generated tokens. One token is listed: 'ghp_SA98YIKE9QhbobKwSn50cjnnyg2bql1Jep8c' with a green checkmark, labeled 'EstructuraDatos'. A note says 'This token has no expiration date.' Below the token list is a note about token usage and a link to the GitHub API documentation.

Semana 3

Esta semana es una práctica dedicada a **Bootstrap**, el framework más usado para diseñar páginas web responsivas, organizadas y modernas.

El propósito de la Semana 3 es:

Aprender cómo funcionan los contenedores, el sistema de columnas (Grid) y los breakpoints responsivos de Bootstrap.

Es una práctica completamente visual y estructural, sin lógica ni JavaScript propio.

¿Qué enseña?

- ✓ 1. Uso de contenedores de Bootstrap

Muestra los diferentes tipos:

- container
- container-fluid
- container-sm
- container-md
- container-lg
- container-xl
- container-xxl

Esto te enseña **cómo el tamaño del contenido cambia** dependiendo del ancho de la pantalla.

- ✓ 2. Cómo funciona el sistema Grid de Bootstrap

Bootstrap usa filas (`row`) y columnas (`col`) para acomodar el contenido.

El archivo muestra:

- Dos columnas simples
- Columnas dentro de `container-fluid`
- Columnas responsivas que cambian color con CSS
- Grids de 2, 6 y 12 columnas
- Columnas que cambian su ancho usando clases como:
 - `col-2`
 - `col-6`
 - `col-8`

Esto sirve para comprender cómo dividir la pantalla en secciones.

✓ 3. Cómo hacer diseños responsivos

Algunas columnas tienen clases personalizadas (`responsive-bg`) y breakpoints para visualizar cómo cambia el diseño según el tamaño del dispositivo.

Es decir:

-  Se ve de una forma
 -  Se adapta automáticamente en pantallas grandes
 -  Las columnas se distribuyen diferente
-

✓ 4. Uso de archivos CSS externos

El archivo carga:

```
<link rel="stylesheet" href="mistyle.css">
```

Ese CSS se usa para cambiar colores o estilos en las columnas responsivas.

✓ 5. Uso de Bootstrap desde CDN

El archivo incluye los CDN de:

- Bootstrap CSS
- Bootstrap JS
- Popper (dependencia para algunos componentes)

Esto te enseña la forma correcta de utilizar Bootstrap sin instalar nada.

[mistyle.css](#)

Este archivo **define colores de fondo diferentes** para las columnas que tienen la clase:

```
.responsive-bg
```

La idea es mostrar visualmente **cómo cambia el diseño dependiendo del tamaño de pantalla**, usando los breakpoints oficiales de Bootstrap.

💡 ¿Qué hace exactamente este CSS?

- ✓ 1. Da un color por defecto (pantallas pequeñas)

Para móviles (menos de 576px):

```
background-color: var(--bs-success); /* Verde */
```

- ✓ 2. Cambia de color según cada breakpoint de Bootstrap

Tamaño de pantalla	Clase	Color aplicado	Significado
--------------------	-------	----------------	-------------

≥ 576px	sm	--bs-primary	Azul
≥ 768px	md	--bs-secondary	Gris
≥ 992px	lg	--bs-danger	Rojo
≥ 1200px	xl	--bs-warning	Amarillo
≥ 1400px	xxl	--bs-info	Celeste

Esto te permite ver **en tiempo real** cómo una columna va cambiando de color al redimensionar la ventana.

⌚ Propósito dentro de la Semana 3

Mostrar visualmente cómo funciona el sistema responsive de Bootstrap.

En el HTML, las columnas que usan:

```
<div class="col border responsive-bg">
```

Cambiarán de color cuando la pantalla sea:

- 📱 pequeña → verde
- 📲 mediana → azul
- 💻 grande → rojo
- 💻 gigante → celeste
- ...etc.

Esto te ayuda a **ver y entender mejor los puntos de quiebre (breakpoints)**.

Semana 4

RESUMEN GENERAL

La Semana 4 es una práctica de **JavaScript básico**, centrada en trabajar con:

- Variables numéricas
- Ciclos (`for`)
- Operaciones matemáticas simples
- Funciones que imprimen resultados en consola

El objetivo de esta semana es reforzar la lógica de programación y el uso de números en JS.

🔗 ¿Qué hace este archivo en conjunto?

✓ 1. Declara números de distintos tipos

- `number = 80` (número entero)
- `decimal = 15.8` (número decimal)
- `legibleNumber = 5_000_000` (número grande con separadores visuales)

Esto sirve para mostrar diferentes formas de escribir valores numéricos.

✓ 2. Imprime una serie de números grandes en incrementos

La función:

```
function saludarNúmero() {
    for(x=1000000; x<=legibleNúmero; x+=1000000) {
        console.log("Saludo número:", x, "\n");
    }
}
```

Imprime:

```
1,000,000
2,000,000
3,000,000
...
hasta llegar a 5,000,000.
```

Esto sirve para practicar ciclos que incrementan en grandes cantidades.

-
- ✓ 3. Suma un número con el valor global `number`

```
function entornoSuma(num) {  
    num = num + number  
    return console.log(num)  
}
```

Si `number = 80` y llamas `entornoSuma(20)`, imprime:

→ 100

Esto sirve para practicar cómo usar variables globales + parámetros.

- ✓ 4. Muestra una tabla de multiplicar ascendente

```
tablaMultiplicar(num)
```

Imprime:

`num × 1`
`num × 2`
...
`num × 10`

- ✓ 5. Muestra una tabla de multiplicar descendente

```
tablaMultiplicarM(num)
```

Imprime:

`num × 10`
`num × 9`
...
`num × 1`

Esto refuerza el uso de ciclos decrecientes.

HTML

Este archivo es una página de **teoría sobre los tipos de datos numéricos en JavaScript**. No ejecuta lógica propia (solo carga el `code.js`), sino que sirve como **material de apoyo** para entender el contenido de la semana.

💡 ¿Qué muestra esta página?

- ✓ 1. Explica qué son los tipos de datos numéricos

Incluye texto que describe:

- Qué es una variable numérica
 - Cómo se crean los números en JavaScript
 - Qué significa que los números sean un **tipo primitivo**
-

- ✓ 2. Uso de la etiqueta <pre> para texto formateado

Ahí se explica:

- Que los números pueden escribirse directamente
 - Que también pueden crearse como objetos usando `new Number()`
-

- ✓ 3. Muestra una tabla comparativa

La tabla explica dos formas de declarar números:

Constructor	Descripción
<code>new Number(number)</code>	Crea un objeto numérico a partir de un número.
<code>number</code>	Forma natural y recomendada de escribir números.

- ✓ 4. Carga el archivo JavaScript

Al final incluye:

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

Esto conecta la teoría del HTML con el código práctico de la Semana 4, donde están las funciones numéricas (saludos, suma, tablas, etc.).

Semana 5

Dom

La Semana 5 es una práctica enfocada en **manipulación del DOM** (Document Object Model).

El objetivo es aprender a:

- Cambiar el contenido de elementos HTML
- Leer valores de inputs
- Crear elementos nuevos desde JavaScript
- Insertarlos en la página dinámicamente
- Modificar el título del documento
- Trabajar con listas y arrays dentro del DOM

Todo esto forma parte de un nivel básico–intermedio de manejo de páginas web con JS.

¿Qué hace este código en conjunto?

-  1. Llena una lista HTML automáticamente

La función:

```
function llenarLista() { ... }
```

Usa `document.getElementById()` para insertar texto dentro de elementos como:

- idea1
- idea2
- idea3
- idea4
- idea5
- idea6

Sirve para **rellenar contenido de una lista** desde JavaScript.

-  2. Saluda al usuario y cambia el título de la pestaña

```
function saludar() {
  let nombre = document.getElementById("nombre").value;
  document.getElementById("miSaludo").innerHTML = "Hola " + nombre + ...
  alumnos.push(nombre);
  document.title = "Clase de " + nombre;
}
```

Esta función:

- Lee el nombre escrito en un input
- Pone un saludo personalizado en pantalla
- Agrega ese nombre al array `alumnos`
- Cambia el nombre del documento (el título de la pestaña)

Aquí practicas:

- Leer inputs
 - Escribir en un div o párrafo
 - Modificar el DOM
 - Manipular arreglos
 - Cambiar metadatos del documento
-

✓ 3. Convierte una lista `` en encabezados `<h4>`

```
function mostrarParrafos() {  
    let parrafos = document.getElementsByTagName("li");  
    let contenedor = document.getElementById("contenedor");  
  
    for (...) {  
        let nuevoP = document.createElement("h4");  
        nuevoP.textContent = parrafos[i].textContent;  
        contenedor.appendChild(nuevoP);  
    }  
}
```

Esta función:

- Toma todos los `` de la página
- Crea nuevos elementos `<h4>`
- Copia el contenido de cada ``
- Inserta los `<h4>` en un contenedor nuevo

Esto sirve para practicar:

- Recorrer elementos HTML
 - Crear elementos con JavaScript
 - Insertarlos dinámicamente
 - Replicar o transformar contenido
-

✓ 4. Usa un array de nombres ('alumnos')

```
var alumnos = ["Ana", "Luis", "Carlos", "Marta", "Sofía"];
```

Y en la función `saludar()` se le agrega un nuevo nombre.

Aquí aprendes cómo mezclar:

- Arrays
- Inputs
- DOM
- Inserción dinámica

index

Esta semana es una práctica de **formularios y manipulación del DOM**, enfocada en leer datos del usuario, procesarlos y mostrar resultados dinámicamente en la página.

¿Qué hace este archivo en conjunto?

- ✓ 1. Lee datos de un formulario

La función obtiene los valores escritos por el usuario en dos inputs:

```
const valor1 = document.getElementById("dato1").value;  
const valor2 = document.getElementById("dato2").value;
```

Aprendes:

- Cómo acceder a inputs por su ID
- Cómo recuperar valores del formulario

-
- ✓ 2. Convierte esos valores a números

```
const suma = Number(valor1) + Number(valor2);
```

Porque los datos de un input vienen como texto.

Esto enseña:

- Conversión de cadenas a números
 - Evitar concatenación de strings (ej: “2” + “3” → “23”)
-

✓ 3. Calcula la suma

Hace la operación matemática:

```
Number(valor1) + Number(valor2)
```

Es un ejemplo básico de procesamiento de datos.

✓ 4. Genera contenido HTML dinámico

Crea una frase usando **template strings**:

```
const resultadoHTML = `<p>La suma es: <strong>${suma}</strong></p>`;
```

Con esto practicas:

- Interpolación de variables con \${ }
 - Crear HTML desde JavaScript
-

✓ 5. Inserta el resultado en la página usando el DOM

```
document.getElementById("resultado").innerHTML = resultadoHTML;
```

Es decir:

- Toma un <div> vacío
- Le mete el texto con la suma
- Actualiza la página sin recargar

(HTML + JS)

Este archivo HTML está diseñado para **probar y demostrar** las funciones del archivo code.js que me enviaste antes.

La semana completa se enfoca en **manipulación del DOM con innerHTML y creación dinámica de elementos**.

El HTML contiene 3 ejercicios principales:

🔗 1. Práctica 1 – Llenar una lista usando innerHTML

El HTML define una lista vacía:

```
<ul>
  <li id="idea1"></li>
  <li id="idea2"></li>
  <li id="idea3"></li>
  <li id="idea4"></li>
  <li id="idea5"></li>
</ul>
<button onclick="llenarLista()">Da Clic para llenar lista</button>
```

Y la función `llenarLista()` del JS **llena esos ** con ideas o mensajes.

✓ Objetivo:

Aprendes cómo cambiar el contenido de elementos HTML usando `innerHTML`.



2. Práctica 2 – Solicitar un nombre y saludar

El usuario escribe su nombre:

```
<input type="text" id="nombre" placeholder="Escribe tu nombre">
<button onclick="saludar()">Saludar</button>
<h3 id="miSaludo"></h3>
```

La función `saludar()` hace:

- Lee el nombre del input
- Muestra un saludo en `miSaludo`
- Guarda el nombre en el arreglo `alumnos`
- Cambia el **título de la pestaña** usando `document.title`

✓ Objetivo:

Manipulación del DOM + manejo de inputs + modificación del título.



3. Práctica 3 – Convertir párrafos <p> en encabezados <h4>

El documento tiene muchos <p>:

```
<p>Este es un párrafo de ejemplo 1.</p>
...
<p>Este es un párrafo de ejemplo 8.</p>
```

Y un botón:

```
<button onclick="mostrarParrafos()">muestra el parrafo</button>
<div id="contenedor"></div>
```

La función `mostrarParrafos()`:

- Obtiene todos los `<p>` con `getElementsByName ("p")`
- Crea nuevos `<h4>` por cada uno
- Copia su texto
- Los mete dentro del `div` contenedor

✓ **Objetivo:**

Crear elementos nuevos desde JS y agregarlos al DOM dinámicamente.

Semana 6

INNER HTML

Esta semana se enfoca en dos temas:

1. Mostrar variables básicas en pantalla usando `innerHTML`
2. Trabajar con arreglos (arrays): agregar, listar, eliminar y editar elementos

Es una práctica más avanzada de **DOM + arrays + actualización dinámica de contenido.**

✓ 1. Mostrar variables y sus tipos

La función:

```
function mostrarVariables()
```

Hace lo siguiente:

- Declara 4 variables de distintos tipos:
 - String → nombre
 - Number → edad
 - Boolean → esEstudiante
 - Double → estatura
- Muestra en el HTML:
 - El valor de cada variable
 - Su tipo (`typeof`)

Ejemplo del resultado:

```
Nombre: Juan Pérez (Tipo: string)
Edad: 30 (Tipo: number)
Es Estudiante: true (Tipo: boolean)
Estatura: 1.70 (Tipo: number)
```

✓ Objetivo:

Aprender tipos de datos JS y cómo imprimirllos usando `innerHTML` y template strings.

✓ 2. Manejo de un arreglo dinámico con inputs

La variable global:

```
let datos = [];
```

sirve como lista donde vas guardando nombres.

✓ 3. Agregar elementos al arreglo y mostrarlos

La función:

```
function datosarreglo()
```

hace:

- ◊ Paso 1: Leer el nombre desde un input
`nombre = document.getElementById("nombre").value;`
- ◊ Paso 2: Limpiar el input
`document.getElementById("nombre").value = "";`
- ◊ Paso 3: Guardar el nombre en el arreglo
`datos.push(nombre);`
- ◊ Paso 4: Limpiar la lista HTML
`document.getElementById("elementos").innerHTML = "";`
- ◊ Paso 5: Volver a imprimir todo lo que haya en el arreglo

Por cada elemento crea:

- Nombre del elemento
- Botón para eliminar
- Botón para editar

```

datos.forEach(function(item, index) {
    document.getElementById("elementos").innerHTML += 
        `<p>Elemento ${index + 1}: ${item}
        <button
        onclick="eliminarElemento(${index})">Eliminar</button>
        <button onclick="editarElemento(${index})">Editar</button>
        </p>`;
});

```

✓ Objetivo:

Aprender cómo:

- Leer inputs
- Guardar cosas en un array
- Volver a renderizar la lista en pantalla
- Generar botones con funciones dinámicas
- Manipular índices

(HTML + JS)

Esta página sirve para **demostrar dos conceptos principales**:

1. **Mostrar diferentes tipos de variables JavaScript en pantalla.**
2. **Trabajar con un arreglo (array) agregando elementos dinámicamente.**

Es una práctica orientada a aprender **DOM + arrays + eventos + innerHTML**.

✓ 1. Botón para mostrar variables básicas

El HTML incluye:

```

<button onclick="mostrarVariables()">Mostrar Datos Variables</button>
<div id="resultado"></div>

```

Cuando el usuario da clic:

- Se ejecuta `mostrarVariables()` desde `code.js`.
- Esa función muestra en pantalla:
 - nombre (string)
 - edad (number)
 - esEstudiante (boolean)
 - estatura (number/double)
- También muestra **el tipo de dato** usando `typeof`.

 *Esto es pura demostración de variables primitivas en JS.*

✓ 2. Sección para manejar un arreglo (array)

El HTML tiene:

```
<input type="text" id="nombre" placeholder="Nombre">
<button onclick="datosarreglo()">Agregar Datos</button>
<div id="elementos"></div>
```

Cuando el usuario escribe algo y da clic:

- Se ejecuta `datosarreglo()`
- Lo que hace:
 - Lee el nombre del input
 - Lo agrega al arreglo `datos[]`
 - Vuelve a generar la lista completa desde cero
 - Muestra cada elemento con:
 - un botón para **eliminar**
 - un botón para **editar**

Esto convierte el HTML en una **lista dinámica**, donde el usuario va agregando elementos a un arreglo y el DOM se actualiza.

💡 *Aquí se practica manipulación avanzada del DOM, ciclos, arreglos y eventos.*

✓ 3. Carga del JS

Al final:

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

Este archivo contiene toda la lógica.

El HTML es solo la interfaz donde se muestra el resultado.

Semana 7

CODE

La Semana 7 combina tres temas importantes:

1. **Tipos de datos básicos (string, number, boolean, double)**
2. **Arreglos dinámicos (añadir, mostrar y manipular una lista)**
3. **Estructuras tipo “struct” en JavaScript (objetos) y arreglos de estructuras**

Aquí pasas de trabajar solo con variables simples a manejar:

- Objetos
- Arreglos de objetos
- Captura de datos desde formularios
- Relleno de estructuras con propiedades

💡 ¿Qué hace el código en conjunto?

✓ 1. Muestra variables y tipos en pantalla

La función:

```
mostrarVariables()
```

Hace lo siguiente:

- Declara nombre, edad, estatura, soltero
- Muestra sus valores y su tipo (`typeof`) en el div “resultado”

- Usa `innerHTML` y template strings

💡 *Esto repasa los tipos primitivos de JavaScript.*

✓ 2. Maneja un arreglo simple (`datos[]`)

La función:

```
datosarreglo()
```

Hace:

1. Lee el nombre del input
2. Lo agrega al arreglo `datos`
3. Limpia el input
4. Vuelve a generarlo en pantalla
5. Muestra botones para:
 - “Eliminar”
 - “Editar”

💡 *Es ejercicio de arreglos + DOM dinámico.*

✓ 3. Trabaja con una estructura tipo “struct”

Declarada así:

```
let miEstructura = {  
    nombre: "",  
    edad: 0,  
    telefono: "",  
    soltero: false,  
    estatura: 0.1  
};
```

La función:

```
datoStruct()
```

Rellena esa estructura con los datos que el usuario escribe en inputs:

- nombre
- edad

- teléfono
- estatura
- soltero

💡 *Esto simula llenar un “struct” como en C/C++, pero en JavaScript se hace usando objetos.*

✓ 4. Arreglo de estructuras (`miEstructuraArreglo[]`)

Aquí ya no guardas solo textos como en `datos[]`.

Ahora guardas **objetos completos**, iguales al “struct”.

La función:

```
datoStructA()
```

- Lee los valores del usuario
- Los envía a `addStructA()`

La función:

```
addStructA(nombre, edad, telefono, estatura, soltero)
```

- Crea un objeto con esos valores
- Lo mete dentro del arreglo `miEstructuraArreglo`
- Imprime el arreglo en consola

Ejemplo de lo que guarda:

```
[  
  { nombre: "Ana", edad: 22, telefono: "12345", estatura: 1.63, soltero:  
  "Sí" },  
  { nombre: "Luis", edad: 30, telefono: "66778", estatura: 1.75, soltero:  
  "No" }  
]
```

💡 *Esto es manejo de arreglos avanzados usando objetos.*

✓ RESUMEN GENERAL — Semana 7

La Semana 7 es una práctica completa sobre **estructuras de datos en JavaScript**, combinando:

- Tipos de datos básicos
- Arreglos simples
- Objetos (tipo “struct”)
- Arreglos de objetos
- Captura de datos desde formularios
- Impresión dinámica en el DOM

Esta semana marca la transición de datos simples → datos estructurado

✓ 1. Mostrar tipos de datos en pantalla

El usuario da clic en:

```
<button onclick="mostrarVariables()">Mostrar Datos Variables</button>
```

La función muestra:

- nombre (string)
- edad (number)
- esEstudiante (boolean)
- estatura (number/double)
- **Y su tipo (typeof)**

➡ **Objetivo:** comprender los tipos primitivos de JavaScript y mostrarlos dinámicamente usando innerHTML.

✓ 2. Manejar un arreglo simple (`datos[]`)

En la segunda parte el usuario escribe un nombre:

```
<input type="text" id="nombre">
<button onclick="datosarreglo()">Agregar Datos</button>
```

La función:

- Toma el nombre
- Lo guarda en el arreglo `datos[]`
- Vuelve a imprimir todos los elementos en pantalla
- Añade botones:
 - **Eliminar**
 - **Editar**

➡ **Objetivo:** aprender cómo funcionan los arreglos y cómo se muestran/actualizan en el DOM.

✓ 3. Manejar una estructura de datos (tipo “struct”)

En esta parte el usuario llena varios campos:

- nombre
- edad
- teléfono
- estatura
- soltero (select)

Con este botón:

```
<button onclick="datoStruct()">Agregar Datos</button>
```

La función datoStruct():

- Toma cada dato
- Se lo asigna a la estructura global:

```
miEstructura = {  
    nombre: "",  
    edad: 0,  
    telefono: "",  
    soltero: false,  
    estatura: 0.1  
}
```

➡ **Objetivo:**

Aprender cómo se rellenan y usan **objetos** en JavaScript como si fueran “structs”.

(NOTA: Este no imprime el struct en pantalla, solo lo llena).

✓ 4. Arreglo de estructuras (structs dentro de un array)

El usuario llena otra versión del formulario:

```
<input id="nombre2">  
<input id="edad2">  
<input id="telefono2">  
<input id="es2">
```

```
<select id="genero2">...</select>
<button onclick="datoStructA()">Agregar Datos</button>
```

La función:

```
datoStructA()
```

- Lee los valores
- Se los pasa a addStructA()

Entonces:

```
addStructA(...)
```

crea un objeto:

```
{
  nombre: "",
  edad: 0,
  telefono: "",
  estatura: 0,
  soltero: true/false
}
```

y lo guarda dentro de:

```
miEstructuraArreglo[]
```

Además, lo muestra en consola.

➡ Objetivo:

Trabajar con **arreglos compuestos por objetos estructurados** (similar a una base de datos en miniatura).

Semana 7

Semana (Listas Enlazadas Sencillas)

Este código intenta representar y recorrer una **lista enlazada simple (Singly Linked List)**.

💡 ¿Qué es una lista enlazada simple?

Es una estructura donde cada nodo contiene:

- un **valor**
- un **puntero** (`next`) al siguiente nodo

Ejemplo:

1 → 2 → 3 → null

💡 ¿Qué hace el código?

✓ 1. Crea una lista enlazada manualmente

```
let singlyLinked = {
  head: {
    value: 1,
    next: {
      value: 2,
      next: {
        value: 3,
        next: null
      }
    }
  }
};
```

Esto forma la estructura:

- Nodo 1 → valor 1 → apunta al nodo 2
- Nodo 2 → valor 2 → apunta al nodo 3
- Nodo 3 → valor 3 → apunta a null (fin de la lista)

💡 Esto es una lista enlazada construida “a mano”.

✓ 2. Intenta recorrer la lista con `for (i of lista)`

```
const recorrer = (lista) => {
  for (i of lista) {
    console.log(i);
  }
};

recorrer(singlyLinked);
```

Esto NO funciona.

Porque:

- singlyLinked **no es iterable**
 - No es un array
 - No tiene método [Symbol.iterator]
-

¿Qué está MAL?

La función `recorrer` está hecha como si fuera un arreglo:

```
for (i of lista)
```

pero una lista enlazada se recorre así:

- Empieza en `head`
 - Mientras `next` no sea `null`
 - Ir moviéndose nodo por nodo
-

Cómo DEBERÍA recorrerse (para que funcione)

Solo para ayudarte a entenderlo:

```
const recorrer = (lista) => {
  let actual = lista.head;
  while (actual !== null) {
    console.log(actual.value);
    actual = actual.next;
  }
};
```

Esto imprimiría:

```
1
2
3
```

(Listas Enlazadas Dobles)

En esta semana trabajas con una estructura de datos avanzada:

👉 Las listas enlazadas dobles (Doubly Linked Lists).

A diferencia de las listas simples, aquí **cada nodo tiene dos punteros**:

- `next` → apunta al siguiente nodo
- `prev` → apunta al nodo anterior

Esto permite recorrer la lista hacia adelante **y hacia atrás**.

🔗 ¿Qué hace el código en conjunto?

✓ 1 Se muestra un ejemplo de estructura simple (no usada)

```
let doublyLinked = { ... }
```

Este es solo un ejemplo de cómo podría verse una lista simple, pero **no participa** en el programa principal.

✓ 2 Se define la clase `Nodes`

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

Cada nodo contiene:

- un valor
- un puntero al siguiente nodo (`next`)
- un puntero al nodo anterior (`prev`)

➡ Esto es la base de una doubly linked list.

✓ 3 Se define la clase `myDoublyLinkedList`

```
class myDoublyLinkedList { ... }
```

Esta clase administra toda la lista.

Incluye:

★ Constructor

Crea la lista con un primer nodo como cabeza (`head`) y cola (`tail`) al mismo tiempo.

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

★ `add(value)`

Agrega un nodo al final de la lista.

Hace:

1. Crea un nodo nuevo
2. Lo enlaza con la cola actual (`prev`)
3. Actualiza la cola (`tail`)
4. Aumenta la longitud

💡 Es como `push()` en un array.

★ `insert(index, value)`

Inserta un nuevo nodo en cualquier posición.

Pasos:

1. Si el índice es mayor que el tamaño → lo agrega al final
2. Busca el nodo actual en ese índice
3. Busca el nodo anterior
4. Ajusta los punteros `next` y `prev`

💡 Este método demuestra la potencia de las listas enlazadas.

★ getTheIndex(index)

Recorre la lista desde la cabeza hasta llegar al índice deseado.

```
while(counter != index) {  
    currentNode = currentNode.next;  
}
```

💡 Función auxiliar para obtener nodos por posición.

★ remove(index)

Elimina un nodo en una posición específica.

1. Obtiene el nodo que va antes del eliminado
2. Obtiene el que va después
3. Reconnecta ambos
4. Reduce la longitud

💡 Simula el comportamiento de `splice()`.

★ search()

Recorre toda la lista e imprime los valores.

✓ 4 Se crea una lista y se agregan nodos

```
let myList = new myDoublyLinkedList("Nodo numero cero");  
myList.add("Primer nodo");  
myList.add("Segundo nodo");  
myList.add("Tercer nodo");  
myList.add("Cuarto nodo");  
myList.search();
```

Esto genera:

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

💡 Se demuestra el funcionamiento de la lista enlazada doble.

Semana 9

(Estructuras de Datos Avanzadas en JavaScript)

En esta semana se implementan **desde cero** varias estructuras de datos clásicas que normalmente existen en lenguajes como Java, C++, Python, pero aquí se construyen manualmente con JavaScript para entender cómo funcionan internamente.

Estas estructuras incluyen:

1. **Arrays personalizados**
2. **Linked Lists (simples y dobles)**
3. **Grafos (Graph)**
4. **Tablas Hash (Hash Table)**
5. **Colas (Queue)**
6. **Pilas (Stack)**
7. **Árbol Binario de Búsqueda (Binary Search Tree)**

El propósito de esta semana es comprender **cómo se manejan datos en memoria**, cómo se enlazan nodos y cómo se construyen estructuras de forma manual.

DETALLE DE LO QUE HACE CADA PARTE

1 MyArray — Implementación manual de un arreglo

Código:

```
class MyArray {}
```

Este “array” casero implementa:

- `push()` → agrega al final
- `pop()` → elimina el último
- `get()` → obtiene un índice
- `delete()` → elimina un índice específico
- `unshift()` → agrega al inicio
- `shift()` → elimina el primer elemento

 **Objetivo:** aprender cómo funcionan internamente los arreglos y cómo reacomodan índices.

2 Doubly Linked List — Lista Enlazada Doble

Implementada en:

```
class MyDoubleLinkedList {}
```

Permite:

- `append(value)` → agregar al final
- `prepend(value)` → agregar al inicio
- `insert(index, value)` → insertar en cualquier posición
- `remove(index)` → eliminar un nodo
- `getTheIndex()` → obtener un nodo por índice

Cada nodo tiene:

- `value`
- `next` → puntero al siguiente
- `prev` → puntero al anterior

 **Objetivo:** entender estructuras de nodos y punteros en ambas direcciones.

3 Graph — Implementación de un grafo

```
class Graph {}
```

Este grafo usa listas de adyacencia:

- `addVertices(node)` → agrega un nodo
- `addEdge(n1, n2)` → crea una conexión bidireccional

Ejemplo de conexiones:

```
1 - 6 - 3 - 5
|     |
4 - 8
```

 **Objetivo:** aprender cómo se representa un grafo con objetos y arrays.

4 HashTable — Tabla Hash

```
class HashTable {}
```

Funciones implementadas:

- `hashMethod(key)` → obtiene dirección de memoria
- `set(key, value)` → guarda un par clave–valor
- `get(key)` → recupera un valor
- `delete(key)` → elimina
- `getAllKey()` → obtiene todas las llaves

Las colisiones se resuelven con **arrays internos (bucket lists)**.

 **Objetivo:** entender cómo funcionan internamente los diccionarios/mapas.

5 Queue — Cola (FIFO)

```
class Queue {}
```

Métodos:

- `peek()` → ver el primero
- `enqueue(value)` → agregar al final
- `dequeue()` → sacar el primero

Característica:

FIFO: First In, First Out

 **Objetivo:** aprender funcionamiento de colas usando nodos enlazados.

6 Stack — Pila (LIFO)

```
class Stack {}
```

Métodos:

- `peek()` → ver el tope
- `push(value)` → agregar arriba
- `pop()` → sacar el de arriba

Característica:

LIFO: Last In, First Out

💡 **Objetivo:** comprender cómo funciona una pila usando nodos.

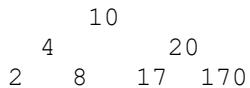
7 Binary Search Tree — Árbol Binario de Búsqueda

```
class BinarySearchTree { }
```

Implementa:

- `insert(value)` → inserta en posición correcta
- `search(value)` → busca un nodo

Organización:



Reglas del BST:

- Valores menores → izquierda
- Valores mayores → derecha

💡 **Objetivo:** entender árboles binarios y búsqueda eficiente.

Semana 10

★ ESTRUCTURAS DE DATOS AVANZADAS

En esta semana se ven dos grandes temas:

- A) ESTRUCTURAS DE DATOS AVANZADAS (Set, Diccionario, HashTable, Árboles, LinkedList)
 - B) CONSUMO DE APIs (Proyecto Rick & Morty con HTML + JS + CSS)
-

Ⓐ ESTRUCTURAS DE DATOS AVANZADAS

1 Set en JavaScript + operaciones de conjuntos

Se practica:

- `add()` → agrega un elemento
- `has()` → verifica si existe
- `size` → cantidad de elementos
- `values()` → obtener iterador

Y se implementan las operaciones matemáticas:

- ✓ **UNIÓN (A ∪ B)**
- ✓ **INTERSECCIÓN (A ∩ B)**
- ✓ **DIFERENCIA (A – B)**

Objetivo: aprender operaciones clásicas de conjuntos con código.

2 Dictionary — Implementación manual de un Diccionario

`Dictionary()`

Simula un diccionario usando un objeto interno `items`.

Incluye:

- `set(key, value)`
- `get(key)`
- `delete(key)`
- `keys()`
- `values()`

Objetivo: entender mapas clave–valor sin usar Map().

3 HashTable — Tabla Hash con direccionamiento abierto

La implementación incluye:

- `hash()` → primer hash
- `secondHash()` → doble hashing
- `put(key,value)` → agregar
- `get(key)` → buscar

Se maneja:

- ✓ Colisiones
- ✓ Recorrido circular ($\text{mod } \%$)
- ✓ Restricción: solo claves numéricas

Objetivo: entender cómo funcionan internamente las tablas hash.

4 BinaryTree — Árbol Binario de Búsqueda + Traversals

Se construye una clase:

BinaryTree

Métodos:

- `addChild()` → insertar ordenado
- `removeChild()` → eliminar nodo (3 casos)
- `print()` → imprimir ordenado

Incluye **tres recorridos**:

- **IN_ORDER** → izquierda, nodo, derecha
- **PRE_ORDER** → nodo, izquierda, derecha
- **POST_ORDER** → izquierda, derecha, nodo

Objetivo: comprender el manejo real de árboles binarios.

5 linkedList — Lista enlazada personalizada

Se implementa:

- `push()`
- `pop()` (incluye caso de 1 elemento)
- `get(index)`
- `isEmpty()`

Objetivo: reforzar listas enlazadas y acceso secuencial.

B CONSUMO DE API — PROYECTO RICK & MORTY

1 Estructura HTML

Página sencilla con:

- <main> donde se inyectan las tarjetas
 - Conexión al script
 - Título y links
-

2 JavaScript para consumir API

Existen dos versiones:

- ✓ Versión con `fetch + .then()`
`getCharacters(done => ...)`
- ✓ Versión moderna con `async/await`
`async function getCharacters() { ... }`

Ambas toman datos desde:

<https://rickandmortyapi.com/api/character?page=1>

3 Renderizado de tarjetas de personajes

Cada personaje genera una tarjeta:

- Imagen
- Nombre
- Estado
- Especie
- Tipo
- Género

Y se inserta dinámicamente en el <main> o <div id="grid">.

4 CSS avanzado estilo “Rick & Morty API”

El estilo:

- Usa fondo oscuro con gradientes
- Tarjetas con hover y sombras
- Diseño responsivo

- Botones tipo “glass / neon”
- Fuentes modernas (Inter)

Objetivo: practicar **interfaz de consumo de API + diseño moderno**.

The screenshot shows the Rick & Morty Demo website on the left and the Chrome DevTools Elements tab on the right. The website displays three character cards: Rick Sanchez, Morty Smith, and Summer Smith. The DevTools Elements tab shows the DOM structure and the CSS styles applied to the elements. The selected element is a style.css rule for a card, which includes properties like background-color, border-radius, and padding.

The screenshot shows the Chrome DevTools Sources tab with the code.js file open. The file contains JavaScript code for fetching characters from an API and creating cards for them. The code includes an asynchronous function to get characters and a function to create a card for each character object. The DevTools sidebar shows the project structure with files like index.html, code.js, and style.css.

```

const API_URL = "https://rickandmortyapi.com/api/character?page=1";
async function getCharacters(){
  try{
    const res = await fetch(API_URL);
    if(!res.ok) throw new Error('Network response was not ok');
    const data = await res.json();
    return data.results || [];
  }catch(err){
    console.error('Error fetching characters:', err);
    return [];
  }
}

function createCard(person){
  const div = document.createElement('article');
  div.className = 'card';

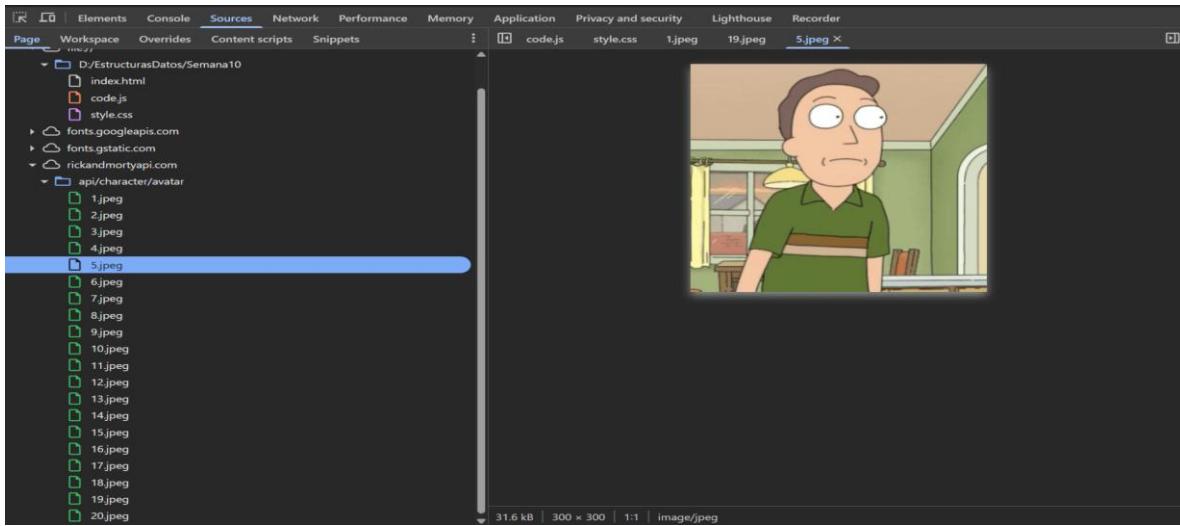
  const img = document.createElement('img');
  img.className = 'thumb';
  img.src = person.image;
  img.alt = person.name;

  const title = document.createElement('h3');
  title.textContent = person.name;

  const status = document.createElement('div');
  const badge = document.createElement('span');
  badge.className = 'badge';
  badge.textContent = person.status || 'Unknown';
  status.appendChild(badge);

  const info = document.createElement('p');
  info.className = 'muted';
  info.textContent = `${person.species}${person.type ? ' ' + person.type : ''} ${person.gender}`;
}

```



Semana 11

(E-commerce)

Voy a explicarte cada parte de estos códigos que conforman un sistema de carrito de compras sencillo para un e-commerce.

1. HTML (Código 2)

Este es el esqueleto de la página web que contiene:

Un header con el título y el ícono del carrito (que muestra la cantidad de items)

Una sección principal (container) con la lista de productos

Un panel lateral (cartTab) para mostrar el carrito de compras que:

Se puede abrir/cerrar

Muestra los productos agregados

Tiene botones para cerrar y para pagar

2. JavaScript (Código 1)

Es el cerebro de la aplicación y hace lo siguiente:

Selección de elementos DOM

```
let iconCart = document.querySelector('.icon-cart'); // Icono del carrito  
let closeCart = document.querySelector('.close'); // Botón para cerrar  
let body = document.querySelector('body'); // Body para efectos visuales  
let listProductsHTML = document.querySelector('.listProduct'); // Contenedor de  
productos  
let listCartHTML = document.querySelector('.listCart'); // Contenedor del carrito  
let iconCartSpan = document.querySelector('.icon-cart span'); // Contador del carrito
```

Funcionalidad Básica

```
// Abrir/cerrar el carrito  
iconCart.addEventListener('click', () => {  
    body.classList.toggle('showCart');  
});
```

Manejo de Productos

`addDataToHTML()`: Toma los datos de productos y los muestra en la página como elementos HTML.

Crea un div por cada producto con su nombre, precio y botón para agregar al carrito.

Evento click en productos: Detecta cuando se hace clic en “Add To Cart” y llama a `addToCart()`.

Manejo del Carrito

addToCart(product_id):

Si el producto no está en el carrito, lo agrega con cantidad 1

Si ya está, incrementa su cantidad

Guarda el carrito en localStorage y actualiza la vista

addCartToHTML():

Muestra los productos del carrito en el panel lateral

Calcula y muestra la cantidad total de items

Muestra el precio total por producto (precio unitario * cantidad)

addCartToMemory(): Guarda el carrito en localStorage para persistencia.

Inicialización

```
const initApp = () => {
  // Carga los productos desde un JSON
  fetch('products.json')
    .then(response => response.json())
    .then(data => {
      listProducts = data;
      addDataToHTML();
      // Si hay carrito guardado, lo carga
    })
}
```

```
if(localStorage.getItem('cart')){  
    carts=JSON.parse(localStorage.getItem('cart'));  
    addCartToHTML();  
}  
});  
  
initApp();
```

3. Datos de Productos (Código 3)

Archivo JSON que contiene:

20 productos de muebles

Cada producto tiene:

id (identificador único)

name (nombre del producto)

price (precio)

image (ruta de la imagen)

4. CSS (Código 4)

Estilos que dan formato a todo el sistema:

Diseño general: Fuentes, márgenes, colores de fondo

Grid de productos: Muestra 4 productos por fila en disposición de cuadrícula

Panel del carrito:

Posición fija a la derecha (fuera de vista inicialmente)

Se desliza al hacer clic en el icono del carrito

Estilos para los items, cantidades y botones

Efectos hover/active: Cambios visuales al interactuar con botones

Responsividad: Uso de max-width para adaptarse a pantallas pequeñas

Flujo Completo

La página carga y `initApp()` obtiene los productos del JSON

Muestra los productos en la página con `addDataToHTML()`

Cuando el usuario hace clic en “Add To Cart”:

Se agrega el producto al carrito

Se actualiza el contador del carrito

Se guarda en `localStorage`

Se muestra en el panel del carrito

El usuario puede abrir/cerrar el carrito con los botones correspondientes

Este código implementa todas las funcionalidades básicas de un carrito de compras: visualización de productos, agregar/eliminar items, persistencia de datos y una interfaz amigable.

[Copiar](#)[Copiar HTML](#)

Semana 12

Proyecto Pokédex

Este es un proyecto de Pokédex que consume la API de Pokémon para mostrar información sobre los diferentes pokémones. Aquí está el análisis de lo que se implementó:

Estructura general

Configuración de VS Code: Se estableció el puerto 5501 para Live Server

CSS: Estilos completos para la aplicación con:

Variables CSS para colores de tipos de pokémon

Diseño responsivo con grids que se adaptan a diferentes tamaños de pantalla

Estilos para botones, tarjetas de pokémon y elementos de navegación

JavaScript: Lógica para:

Consumir la API de Pokémon

Mostrar los pokémones en tarjetas

Filtrar por tipo usando los botones del header

HTML: Estructura básica con:

Header con botones de filtrado

Área principal para mostrar los pokémones

Características implementadas

Diseño responsivo: 1 columna en móvil, 2 en tablet y 3 en desktop

Filtrado por tipo: Permite ver pokémones de un tipo específico

Visualización de datos: Muestra ID, nombre, imagen, tipos, altura y peso

Efectos visuales: Hover en botones, sombras y transiciones

Tipografía: Uso de Google Fonts (Rubik)

Posibles mejoras

Paginación: Actualmente muestra 150 pokémones, podrías implementar carga por páginas

Búsqueda: Añadir un campo para buscar pokémones por nombre

Detalle de pokémon: Implementar una vista detallada al hacer clic en una tarjeta

Loading states: Mostrar indicadores de carga mientras se obtienen los datos

Manejo de errores: Controlar cuando la API no responda correctamente

El proyecto está bien estructurado y muestra un buen uso de HTML, CSS y JavaScript para crear una aplicación funcional que consume una API externa.



Semana 13

EduTrack - Sistema de Asistencia y Participación

Descripción General

EduTrack es un sistema completo de gestión de asistencia y participación estudiantil diseñado para instituciones educativas. Ofrece una interfaz moderna y eficiente que permite a los docentes registrar, monitorear y analizar el rendimiento académico de sus estudiantes.

Características Principales

Gestión de Asistencia

- Registro rápido de asistencia por clase y fecha
- Marcación masiva (todos presentes/ausentes)
- Registro de llegadas tardías
- Historial completo de asistencia por estudiante

Seguimiento de Participación

- Evaluación de participación con sistema de puntuación (1-10)
- Categorización por tipo de participación (respuesta, pregunta, participación activa, presentación)
- Notas descriptivas para cada registro
- Análisis de patrones de participación

Reportes y Análisis

- Dashboard con estadísticas en tiempo real
- Reportes detallados de asistencia y participación
- Análisis individual de estudiantes
- Vista general por clase
- Exportación de datos en formato JSON

Interfaz Moderna

- Diseño responsive y accesible
- Animaciones suaves y micro-interacciones
- Panel de control intuitivo
- Notificaciones en tiempo real

Arquitectura del Sistema

Tecnologías Utilizadas

- **Frontend**: HTML5, CSS3, JavaScript ES6+
- **Framework CSS**: Tailwind CSS
- **Animaciones**: Anime.js
- **Almacenamiento**: LocalStorage (base de datos local)
- **Tipografías**: Inter (texto principal), JetBrains Mono (datos)

Estructura de Base de Datos

Estudiantes (students)

```
```javascript
{
 id: "string", // Identificador único
 name: "string", // Nombre completo
 email: "string", // Correo electrónico
 studentId: "string", // Matrícula
 classGroup: "string", // Grupo (10A, 10B, etc.)
 enrollmentDate: "date", // Fecha de inscripción
 status: "string" // Estado (active, inactive)
}
```

```

Clases (classes)

```
```javascript
{
 id: "string", // Identificador único
 name: "string", // Nombre de la clase
 subject: "string", // Código de materia
 teacherId: "string", // ID del docente
 schedule: "string", // Horario
 semester: "string", // Semestre
 maxStudents: "number" // Capacidad máxima
}
```

```

Registros de Asistencia (attendance_records)

```
```javascript
{
 id: "string", // Identificador único
 studentId: "string", // ID del estudiante
 classId: "string", // ID de la clase
 date: "date", // Fecha
 status: "string", // present, absent, late, excused
 checkInTime: "string", // Hora de entrada
 notes: "string" // Notas adicionales
}
```

```

Registros de Participación (participation_records)

```
```javascript
{
 id: "string", // Identificador único
 studentId: "string", // ID del estudiante
 classId: "string", // ID de la clase
 date: "date", // Fecha
 score: "number", // Puntaje (1-10)
 type: "string", // Tipo de participación
 description: "string", // Descripción
 teacherNotes: "string" // Notas del docente
}
```

```

Procesos del Sistema

1. Registro de Asistencia

1. Seleccionar clase y fecha
2. Abrir modal de asistencia
3. Marcar estado de cada estudiante (presente/ausente/tarde)
4. Guardar registros en base de datos
5. Actualizar estadísticas en dashboard

2. Registro de Participación

1. Seleccionar estudiante y clase
2. Elegir tipo de participación

3. Asignar puntaje (1-10)
4. Agregar notas descriptivas
5. Guardar registro y actualizar promedios

[### 3. Generación de Reportes](#)

1. Seleccionar tipo de reporte
2. Aplicar filtros (período, clase, estudiante)
3. Calcular estadísticas
4. Renderizar visualizaciones
5. Exportar datos si es necesario

[### 4. Análisis y Alertas](#)

- Detección automática de baja asistencia (< 70%)
- Identificación de estudiantes con baja participación
- Generación de alertas tempranas
- Recomendaciones para intervención

[## Instalación y Uso](#)

[### Requisitos Previos](#)

- Navegador web moderno (Chrome, Firefox, Safari, Edge)
- No requiere instalación de software adicional

[### Pasos de Instalación](#)

1. Descargar todos los archivos del sistema
2. Abrir `index.html` en un navegador web
3. El sistema está listo para usar

Uso Inicial

1. El sistema carga con datos de ejemplo
2. Los datos se almacenan localmente en el navegador
3. Se pueden agregar, editar y eliminar registros
4. Los datos persisten entre sesiones

Funcionalidades Avanzadas

Exportación de Datos

- Exportación completa en formato JSON
- Incluye todos los registros y configuraciones
- Ideal para respaldos y migración de datos

Análisis Estadístico

- Cálculo de tasas de asistencia porcentuales
- Promedios de participación ponderados
- Rankings de rendimiento general
- Identificación de patrones y tendencias

Sistema de Notificaciones

- Notificaciones toast para acciones exitosas
- Alertas de validación en formularios
- Mensajes de confirmación para acciones importantes
- Indicadores visuales de estado

Diseño y Experiencia de Usuario

Principios de Diseño

- **Claridad**: Información organizada y fácil de encontrar
- **Eficiencia**: Flujos de trabajo optimizados para tareas repetitivas
- **Accesibilidad**: Interfaz intuitiva para usuarios de todos los niveles
- **Profesionalismo**: Apariencia seria y confiable para el entorno educativo

Paleta de Colores

- **Primarios**: Azul profundo (#1e3a8a), Azul claro (#3b82f6)
- **Estado**: Verde (#10b981), Rojo (#ef4444), Ámbar (#f59e0b)
- **Fondos**: Gris claro (#f8fafc), Blanco (#ffffff)

Tipografía

- **Principal**: Inter (sans-serif moderna)
- **Datos**: JetBrains Mono (monospace para números)

Seguridad y Privacidad

Protección de Datos

- Almacenamiento local en el navegador
- No se envían datos a servidores externos
- Cumplimiento con normativas de privacidad estudiantil
- Control total sobre la información por parte del usuario

Acceso y Permisos

- Sistema de un solo usuario
- No requiere autenticación
- Acceso directo a todas las funcionalidades
- Responsabilidad del usuario la protección de datos

Solución de Problemas

Problemas Comunes

1. **Datos no guardados**: Verificar que el navegador permite LocalStorage
2. **Interfaz no responde**: Recargar la página y verificar JavaScript
3. **Reportes vacíos**: Asegurar que hay registros para el período seleccionado
4. **Exportación fallida**: Verificar permisos de descarga del navegador

Soporte Técnico

- Documentación completa incluida
- Código comentado y estructurado
- Arquitectura modular para fácil mantenimiento
- Sin dependencias externas complejas

Futuras Mejoras

Funcionalidades Planificadas

- Sistema de autenticación multi-usuario

- Sincronización en la nube
- Aplicación móvil complementaria
- Integración con sistemas LMS
- Reportes personalizables
- Análisis predictivo

[### Optimizaciones](#)

- Mejora en el rendimiento con grandes volúmenes de datos
- Funcionalidad offline completa
- Exportación en múltiples formatos (PDF, Excel, CSV)
- Temas de interfaz personalizables

[## Conclusión](#)

EduTrack representa una solución completa y moderna para la gestión de asistencia y participación estudiantil. Su diseño intuitivo, combinado con funcionalidades avanzadas de análisis y reportes, lo convierte en una herramienta valiosa para instituciones educativas que buscan optimizar sus procesos académicos.

El sistema está diseñado para ser escalable, mantenable y adaptable a las necesidades específicas de cada institución, proporcionando una base sólida para la gestión efectiva del rendimiento estudiantil.

Semana 14

[Aplicación ToDo](#)

Este es un sistema de gestión de tareas (ToDo) completo que funciona en el navegador usando localStorage para persistir los datos. Vamos a analizar sus componentes:

Características principales

Funcionalidad CRUD completa:

Crear nuevas tareas

Leer/visualizar tareas existentes

Actualizar tareas (edición y cambio de estado)

Eliminar tareas

Filtrado y búsqueda:

Filtrado por estado (todas, pendientes, completadas)

Búsqueda por texto en título o descripción

Interfaz de usuario:

Formulario para agregar tareas

Listado de tareas con acciones

Modal de edición

Diseño responsive

Estructura del código

HTML (Código 2):

Define la estructura de la aplicación con paneles para registro y consulta

Incluye formularios y controles de interfaz

Contiene el modal de edición oculto

CSS (Código 3):

Estilos modernos con variables CSS

Diseño responsive que se adapta a pantallas grandes

Estilización de componentes (tarjetas, botones, formularios)

JavaScript (Código 1):

Lógica principal de la aplicación

Manejo de eventos y actualización de la interfaz

Gestión del almacenamiento local (localStorage)

Funcionamiento interno

Almacenamiento: Usa localStorage con la clave ‘todo_semana14_tasks_v1’

Estructura de datos: Cada tarea tiene:

```
{  
  id: String,    // ID único basado en timestamp  
  title: String, // Título de la tarea  
  description: String, // Descripción opcional  
  done: Boolean, // Estado de completado  
  createdAt: String // Fecha de creación en ISO  
}
```

Renderizado: La función renderTasks() filtra y muestra las tareas según los criterios de búsqueda y filtro seleccionados

Posibles mejoras

Validación más robusta de los campos de entrada

Notificaciones visuales para acciones (éxito/error)

Soporte para categorías o etiquetas

Exportación/importación de datos

Sincronización con un backend

Esta aplicación es un excelente ejemplo de una SPA (Single Page Application) funcional usando solo tecnologías web básicas (HTML, CSS, JavaScript) sin dependencias externas.

Semana 15

Código 1: Representaciones de Gráficas (Grafos)

Este código muestra diferentes formas de representar un grafo:

[Lista de Aristas](#)

Almacena las conexiones como pares [nodo1, nodo2].

Ejemplo: [[0, 2], [2, 3], [2, 1], [1, 3]].

[Lista de Adyacencia](#)

Cada nodo guarda sus vecinos conectados.

Puede ser un arreglo: [[2], [2,3], [0,1,3], [1,2]].

O un objeto: {

0: [2],

1: [2, 3],

2: [0, 1, 3],

3: [1, 2]

}

[Matriz de Adyacencia](#)

Un arreglo bidimensional donde 1 indica conexión y 0 significa que no hay conexión.

Ejemplo: [

```
[0, 0, 1, 0],  
[0, 0, 1, 1],  
[1, 1, 0, 1],  
[0, 1, 1, 0]  
]
```

También puede representarse como un objeto con arreglos.

Código 2: Clase Grafo (Implementación Básica)

Define una clase Graph con métodos para:

addVertex(nodo) → Agrega un nodo al grafo.

addEdge(nodo1, nodo2) → Crea una conexión bidireccional entre dos nodos.

Ejemplo de uso:

```
const myGraph = new Graph();  
  
myGraph.addVertex(0);  
  
myGraph.addVertex(1);  
  
myGraph.addVertex(2);  
  
myGraph.addEdge(0, 2);  
  
myGraph.addEdge(1, 2);
```

Código 3: Implementación de Tabla Hash

Una clase HashTable con los siguientes métodos:

hashMethod(key) → Convierte una clave en un índice usando un algoritmo simple.

set(key, value) → Almacena un par clave-valor.

get(key) → Recupera el valor asociado a una clave.

delete(key) → Elimina un par clave-valor.

getAllKeys() → Devuelve todas las claves almacenadas.

Ejemplo de uso:

```
const myHashTable = new HashTable(50);
myHashTable.set("name", "John");
myHashTable.set("age", 25);
console.log(myHashTable.get("name")); // "John"
myHashTable.delete("age");
console.log(myHashTable.getAllKeys()); // ["name"]
```

En el Código 4 (Doubly Linked List):

En el método insert, hay un error: const newNode = Node(value); debería ser const newNode = new Node(value); (falta el new)

La condición if (index === 1) debería ser if (index === 0) para ser consistente con el prepend

Faltaría manejar el caso cuando la lista está vacía o con un solo nodo

[En el Código 5 \(Singly Linked List\):](#)

Este código está bien estructurado, pero podrías agregar validaciones adicionales en los métodos

En el Código 6:

Hay varios errores de sintaxis en la función determinaValor:

num = 10 debería ser num == 10 (comparación no asignación)

El operador && está mal usado para comparar múltiples valores

En numberToSpanish, hay un error con < que debería ser <

La función myFunction2 tiene una estructura incorrecta con llaves innecesarias

Hay algunos console.log comentados que podrían eliminarse

Semana 16

[Análisis del Proyecto Pokémon con Estructuras de Datos](#)

Basado en los fragmentos de código proporcionados, este es un proyecto JavaScript que combina:

[Componentes principales](#)

[Estructuras de datos implementadas:](#)

Código 1: Cola (Queue) usando lista enlazada con clase Node

Código 2: Pila (Stack) usando lista enlazada

Código 3: Árbol binario de búsqueda (BST)

Aplicación de Pokémon con API:

Código 4: Estilos CSS con temática de tipos Pokémon

Código 5: Estructura HTML para mostrar tarjetas de Pokémon con filtros por tipo

Código 6: Esqueleto JavaScript para llamadas API y visualización de Pokémon

Observaciones clave

Falta implementar la API:

Las funciones apiRequest() y paintPokemon() están vacías

El event listener hace referencia a peticionApi() que no está definida

Características modernas:

Usa CSS Grid para diseño responsive (grid-template-columns: repeat(auto-fit, minmax(350px, 1fr)))

Unidades modernas CSS (dvh, prefijo d para unidades viewport)

Características ES6 (clases, async/await)

Estructuras completas:

Las 3 estructuras (Cola, Pila, BST) están implementadas con sus métodos principales

Recomendaciones para completar el proyecto

Integración con la API:

```
const apiRequest = async () => {
  try {
    const response = await fetch('https://pokeapi.co/api/v2/pokemon?limit=151');
    const data = await response.json();
    return data.results;
  } catch (error) {
    console.error('Error al obtener Pokémon:', error);
  }
};
```

Mostrar Pokémon:

```
const paintPokemon = (pokemonList) => {
  pokemonList.forEach(async (pokemon) => {
    const response = await fetch(pokemon.url);
    const pokemonData = await response.json();
    // Crear y añadir elementos de tarjeta usando pokemonData
  });
};
```

});

};