Universidad Tecnológica Metropolitana

Estructuras de Datos Aplicadas

ISC. Ruth Betsaida Martínez Domínguez, MGTI

Práctica 5-9

Soberanis Acosta Jimena Monserrat

Desarrollo de Software Multiplataforma

Cuarto Cuatrimestre

4°B

Parcial II

Martes, 24 de octubre de 2023

PRÁCTICA 5

Para esta primera práctica comencé con el html:

ejercicio5 > index.html > html > body > div.container > script

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/styles.css">
    <title>Almacenamiento de Palabras</title>
</head>
<body>
    <div class="container">
        <h1>Almacenamiento de Palabras</h1>
        <div class="input-container">
        <input type="text" id="palabraInput" placeholder="Ingresa una palabra">
        <button id="agregarPalabra">Agregar Palabra</button>
        </div>
        <div id="listas-container"></div>
        <script type="module" src="js/App.js"></script>
    </div>
</body>
</html>
```

```javascript
export class PalabraLista {
    constructor(primeraLetra) {
        this.primeraLetra = primeraLetra;
        this.palabras = [];
    }

    agregarPalabra(palabra) {
        this.palabras.push(palabra);
        this.palabras.sort();
    }

    crearElementoLista() {
        const listaDiv = document.createElement('div');
        listaDiv.classList.add('lista');

        const listaTitulo = document.createElement('h2');
        listaTitulo.textContent = `Lista ${this.primeraLetra}`;
        listaDiv.appendChild(listaTitulo);

        const listaUl = document.createElement('ul');
        this.palabras.forEach(palabra => {
            const listaLi = document.createElement('li');
            listaLi.textContent = palabra;
            listaUl.appendChild(listaLi);
        });
        listaDiv.appendChild(listaUl);

        return listaDiv;
    }
}
```

Después comencé con la primera clase del código llamado “PalabraLista”, donde existe un constructor que toma ‘primeraLetra’. Y después se crea la variable ‘primeraLetra’ y una lista vacía llamada ‘palabras’. Seguí con la función ‘agregarPalabra’ que mete cada palabra en la lista de palabras y estas mismas se ordenan, y por último está la función ‘crearElementoLista’ que básicamente crea una lista de las palabras, crea un ‘div’, así como su título y también ‘ul’ y ‘li’, y en cada ‘li’ está la palabra agregada.

La clase ‘PalabraLista’ se importa a la siguiente llamada ‘App’ que es la que interactúa con el usuario, crea 3 variables que cada uno llama a algo en específico del HTML. Después crea 2 variables de listas inicialmente vacías.

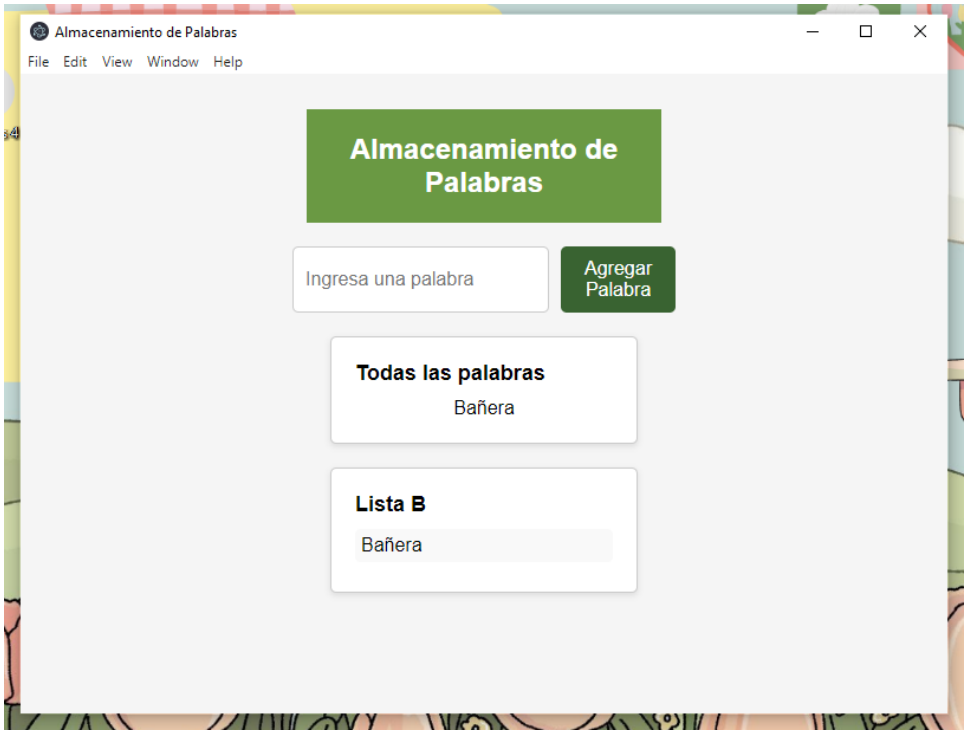
Comencé en la primera función llamada ‘agregarPalabra’ que toma el contenido del input y ese mismo lo que toma es la primera letra inicial y se comprueba si exista alguna lista que tenga la misma letra inicial, sino existe se crea una nueva lista, en caso contrario se agrega esa palabra a la su lista existente. Por último, se llama a la función renderizarListas.

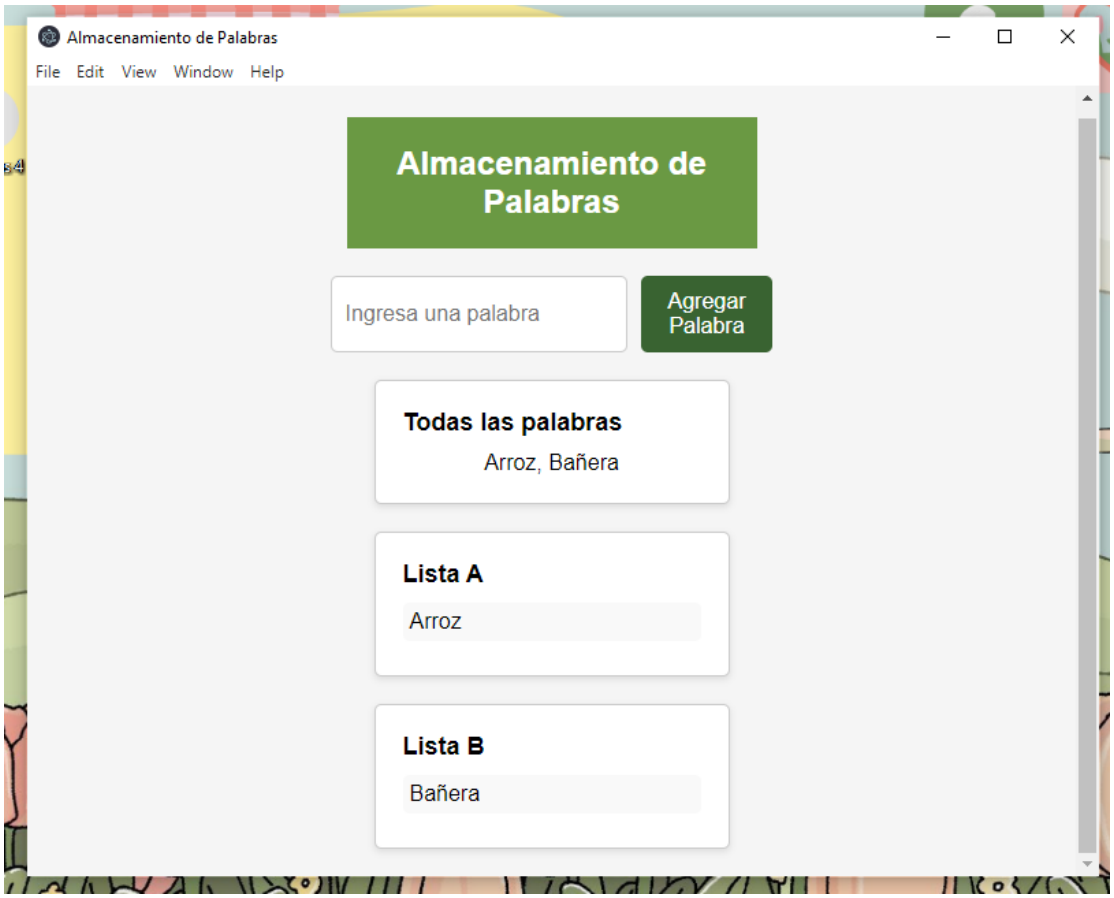
```
1   import { PalabraLista } from './Palabra.js';
2
3   const palabraInput = document.getElementById('palabraInput');
4   const agregarPalabraBtn = document.getElementById('agregarPalabra');
5   const listasContainer = document.getElementById('listas-container');
6
7   const listas = {};
8   const todasLasPalabras = [];
9
10  function agregarPalabra() {
11      const palabra = palabraInput.value.trim();
12      if (palabra.length > 0) {
13          const primeraLetra = palabra[0].toUpperCase();
14
15          if (!listas[primeraLetra]) {
16              listas[primeraLetra] = new PalabraLista(primeraLetra);
17          }
18
19          listas[primeraLetra].agregarPalabra(palabra);
20          todasLasPalabras.push(palabra);
21          todasLasPalabras.sort();
22          palabraInput.value = '';
23
24          renderizarListas();
25      }
26  }
```

Esta función renderizarListas, solamente sirve para crear la lista principal de todas las palabras, cada palabra se agrega en un párrafo separado por ‘,’. Por último, se llama a la función agregarPalabra a través del click en el botón.

```
function renderizarListas() {
    listasContainer.innerHTML = '';

    todasLasPalabras.sort();

    const letrasOrdenadas = Object.keys(listas).sort();

    const todasLasPalabrasDiv = document.createElement('div');
    todasLasPalabrasDiv.classList.add('lista');

    const todasLasPalabrasTitulo = document.createElement('h2');
    todasLasPalabrasTitulo.textContent = 'Todas las palabras';
    todasLasPalabrasDiv.appendChild(todasLasPalabrasTitulo);

    const todasLasPalabrasTexto = document.createElement('p');
    todasLasPalabrasTexto.textContent = todasLasPalabras.join(', ');
    todasLasPalabrasDiv.appendChild(todasLasPalabrasTexto);

    listasContainer.appendChild(todasLasPalabrasDiv);

    letrasOrdenadas.forEach(letra => {
        listasContainer.appendChild(listas[letra].crearElementoLista());
    });
}

agregarPalabraBtn.addEventListener('click', agregarPalabra);
```

Le agregué su respectivo estilo con el CSS y con ayuda del Electron subí mi aplicación:





PRÁCTICA 6

Primeramente, hice su HTML:

```
ejercicio6 > <> index.html > ...
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/style.css">
    <title>Invertir Palabra</title>
</head>
<body>
    <div class="container">
        <h1>Invertir Palabra</h1>
    <div class="input-container">
        <input type="text" id="palabraInput" placeholder="Ingresa una palabra">
        <button id="invertirPalabra">Invertir Palabra</button>
    </div>
    <div id="resultado-container"></div>
    <script type="module" src="js/app.js"></script>
    </div>
</body>
</html>
```

Para los archivos del código, comencé con la clase InversorPalabra que básicamente toma la palabra y la inverte en su orden de palabras y la guarda en ‘palabraInvertida’;

ejercicio6 > js > JS inversorPalabras.js > ...

```
export class InversorPalabra {
    invertir(palabra) {
        const palabraInvertida = palabra.split('').reverse().join('');
        return palabraInvertida;
    }
}
```

Esa clase se importa a la siguiente llamada ‘App’ que es la que interactúa con el usuario, primero toma ciertos elementos del DOM

ejercicio6 > js > JS app.js > invertirPalabraBtn.addEventListener('click') callback

```
import { InversorPalabra } from "./inversorPalabras.js";

const palabraInput = document.getElementById('palabraInput');
const invertirPalabraBtn = document.getElementById('invertirPalabra');
const resultadoContainer = document.getElementById('resultado-container');

const listaPalabras=[];
```

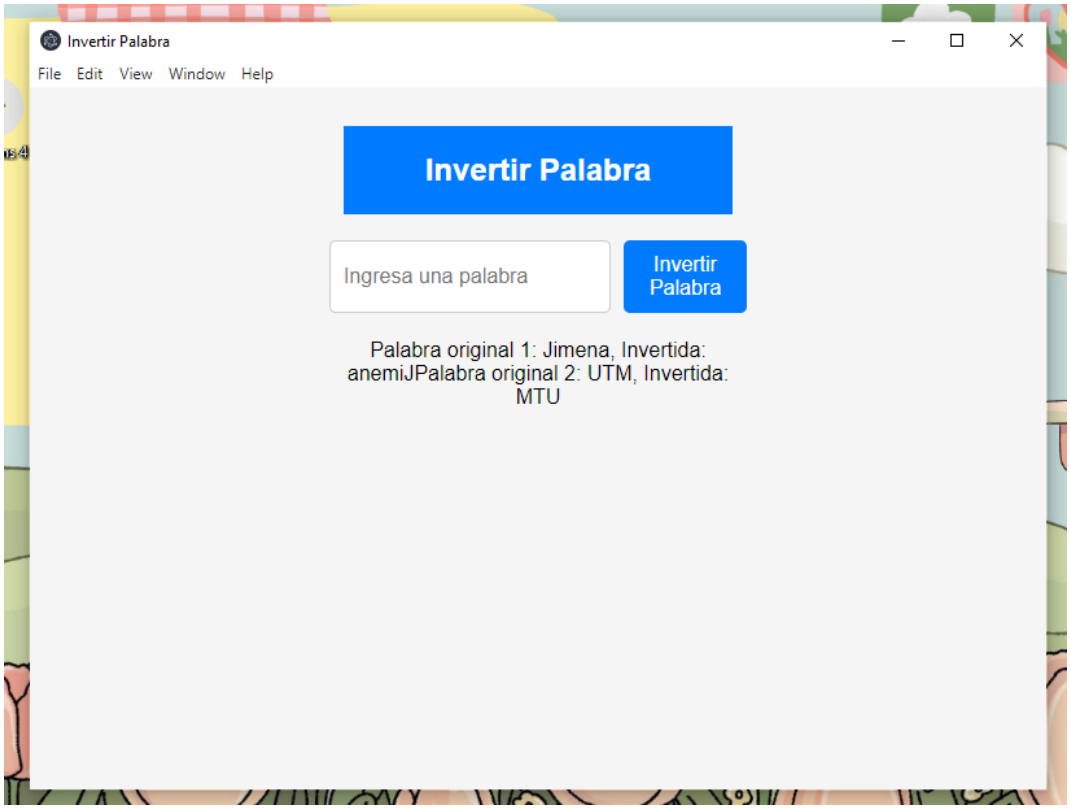
Después, se le agrega un evento al botón, lo que hace es tomar la palabra y llamar a la función de invertir para poder invertir la misma palabra. La palabra ingresada y la palabra invertida se guardan en la lista.

```
invertirPalabraBtn.addEventListener('click', () => {
    const palabra = palabraInput.value.trim();
    const inversor = new InversorPalabra();
    const palabraInvertida = inversor.invertir(palabra);

    listaPalabras.push({ original: palabra, invertida: palabraInvertida });

    actualizarLista();

    palabraInput.value="";
});

```

Y, por último, está la función que se encarga de actualizar la lista. Primero, limpia el contenedor de la lista en el HTML, para después, recorrer el arreglo listaPalabras y crea un nuevo elemento de lista para cada palabra original e invertida en un específico formato. Finalmente, estos elementos se agregan al contenedor 'resultadoContainer', lo que permite mostrar la lista actualizada con las palabras y sus versiones invertidas.

```
function actualizarLista(){
    resultadoContainer.innerHTML=" ";

    listaPalabras.forEach((palabraLi, index) =>{
        const elemento = document.createElement('divLista');
        elemento.textContent = `Palabra original ${index + 1}: ${palabraLi.original}, Invertida: ${palabraLi.invertida}`;
        resultadoContainer.appendChild(elemento);
    })
}
```

Le agregué estilo con un css y con ayuda de Electron lo subí a aplicación:



PRÁCTICA 7

Comencé con el HTML para la práctica:

ejercicio7 > index.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/style.css">
    <title>Verificador de Palíndromos</title>
</head>
<body>
    <div class="container">
        <h1>Verificador de Palíndromos</h1>
        <div class="input-container">
            <input type="text" id="palabraInput" placeholder="Ingresa una palabra">
            <button id="verificarPalindromo">Verificar Palíndromo</button>
        </div>
        <div id="resultado-container"></div>
        <script type="module" src="js/app.js"></script>
    </div>
</body>
</html>
```

Después hice la primera clase llamada VerficadorPalindromo, que como su nombre dice, verificará si la palabra se lee igual al derecho y al revés, esto lo hace con la función esPalindromo, en donde toma la variable palabra, que también puede ser una frase y se utiliza trim para quitar los espacios blancos del inicio y al final, el replace para quitar los espacios en blanco de la frase y ponerlas en minúsculas para convertirla en una palabra, y de último, esta se invierte (de la misma manera que en la práctica anterior) y luego es cuando se comprueba si la palabra/frase es igual a la palabra invertida.

ejercicio7 > js > verificadorPalindromo.js > ...

```js
export class VerificadorPalindromo {
    esPalindromo(palabra) {

        palabra = palabra.trim().replace(/\s/g, '').toLowerCase();

        const palabraInvertida = palabra.split('').reverse().join('');

        return palabra === palabraInvertida;
    }
}
```

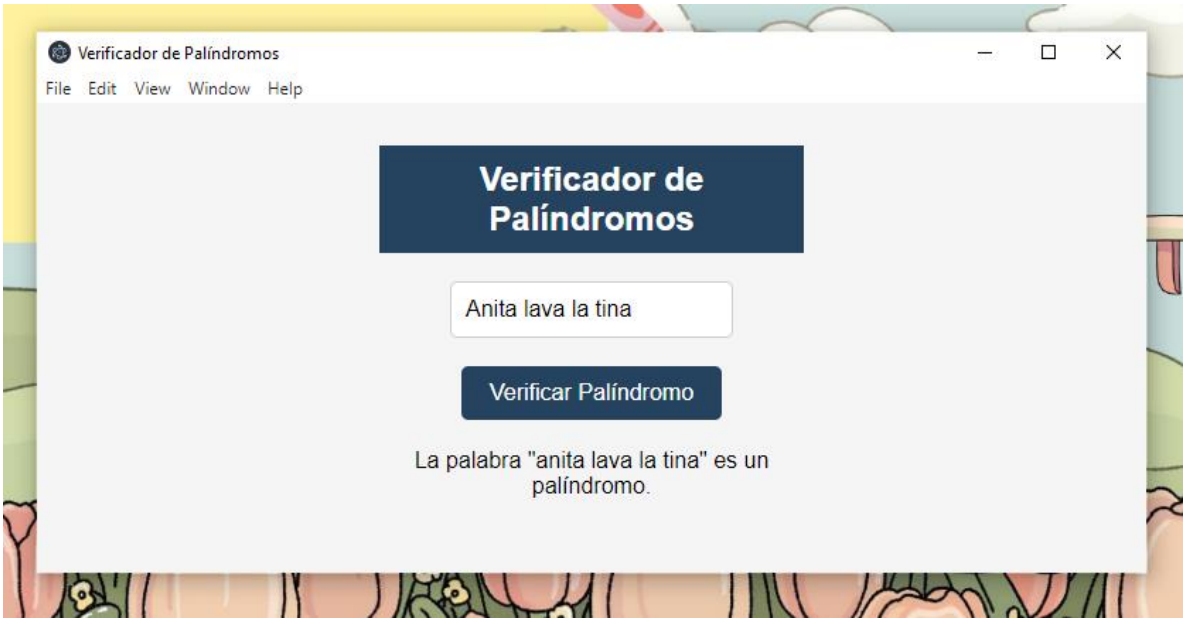
Continué con la parte que interactúa con el usuario, en donde, primero llamé a ciertas partes del DOM, y creé una lista vacía en donde se van guardando cada palabra/frase

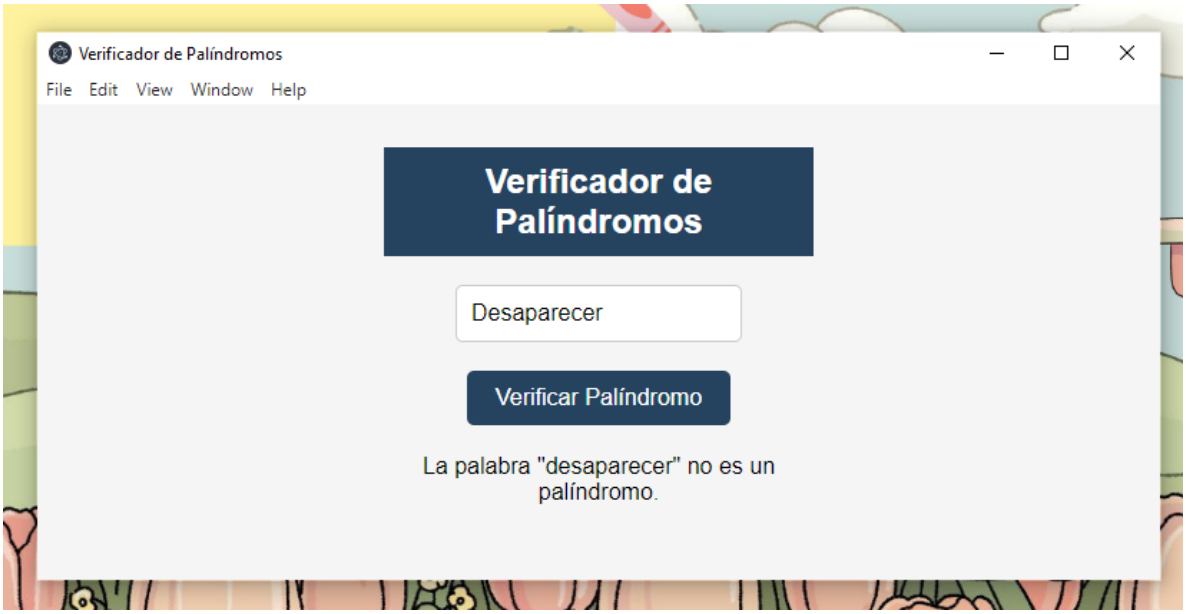
```
ejercicio7 > js > JS app.js > ...
1   import { VerificadorPalindromo } from './verificadorPalindromo.js';
2
3   const palabraInput = document.getElementById('palabraInput');
4   const verificarPalindromoBtn = document.getElementById('verificarPalindromo');
5   const resultadoContainer = document.getElementById('resultado-container');
6
7   const formatoLista = [];
8
```

La última parte es cuando se llama al botón de enviar la palabra, toma la palabra que se envía y comprueba si se lee igual al revés, esa variable se guarda en la lista y después, si es palíndromo o no, aparece con su respectivo formato.

```
8
9   verificarPalindromoBtn.addEventListener('click', () => {
10      const palabra = palabraInput.value.toLowerCase();
11      const verificador = new VerificadorPalindromo();
12      const esPalindromo = verificador.esPalindromo(palabra);
13
14      formatoLista.push(esPalindromo);
15
16      if (esPalindromo) {
17          resultadoContainer.textContent = `La palabra "${palabra}" es un palíndromo.`;
18      } else {
19          resultadoContainer.textContent = `La palabra "${palabra}" no es un palíndromo.`;
20      }
21  });
22
```

Le agregué su respectivo estilo con CSS y lo mandé a aplicación con Electron:




Verificador de Palíndromos
File Edit View Window Help
Verificador de Palíndromos
Desaparecer
Verificar Palíndromo
La palabra "desaparecer" no es un palíndromo.

PRÁCTICA 8

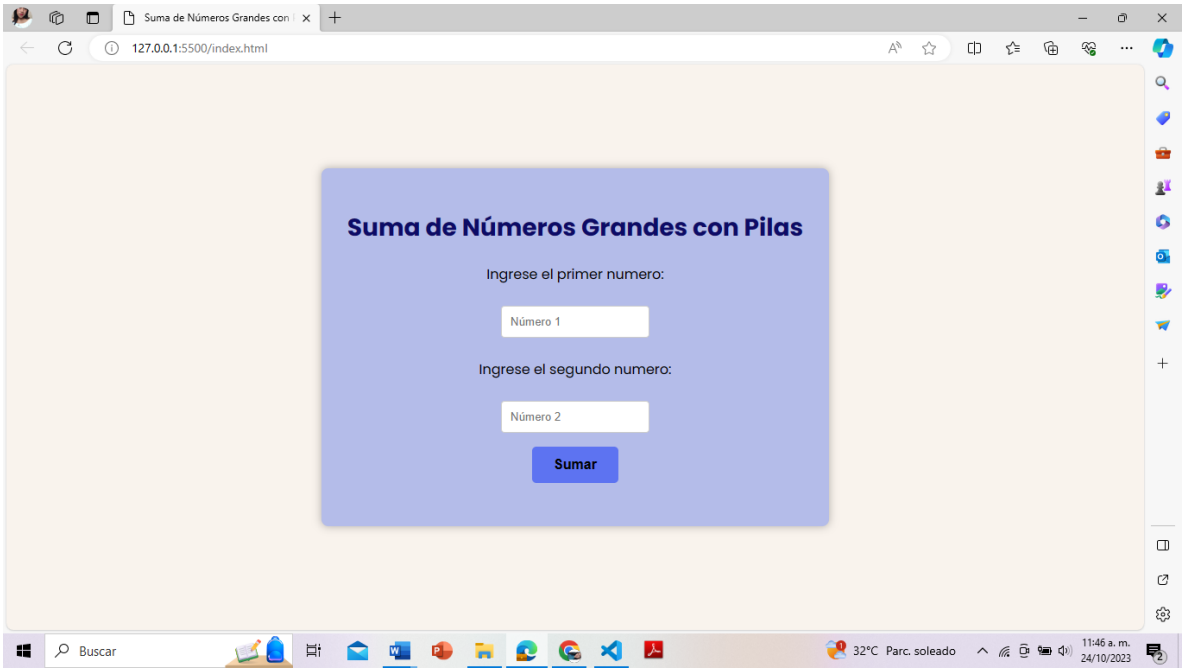
Cree una página una página web que se centra en la suma de números grandes utilizando pilas. La página presenta un título, campos de entrada de texto para dos números, un botón para realizar la suma y un área para mostrar el resultado. Los elementos están contenidos dentro de un contenedor principal con el atributo "id" de "app". El título principal de la página es "Suma de Números Grandes con Pilas," y se proporcionan instrucciones para que los usuarios ingresen los dos números que deseen sumar. Cuando los usuarios hacen clic en el botón "Sumar", se invocará una funcionalidad definida en el archivo JavaScript "script.js" para llevar a cabo la suma de números utilizando pilas. El resultado de la suma se mostrará en el área designada en la página.

index.html

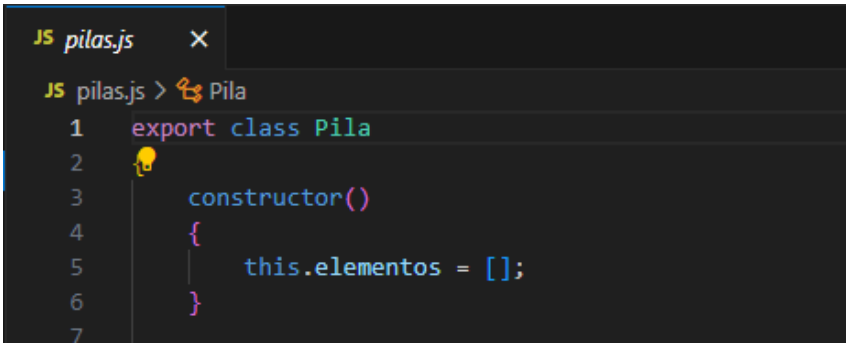
index.html > html > body

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Suma de Números Grandes con Pilas</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <div id="app">

        <h1>Suma de Números Grandes con Pilas</h1>

        <p>Ingrese el primer numero: </p>
        <input type="text" id="numero1" placeholder="Número 1">

        <p>Ingrese el segundo numero: </p>
        <input type="text" id="numero2" placeholder="Número 2">

        <button id="sumarBoton">Sumar</button>

        <div id="resultado"></div>
    </div>

    <script type="module" src="script.js"></script>
</body>
</html>

```

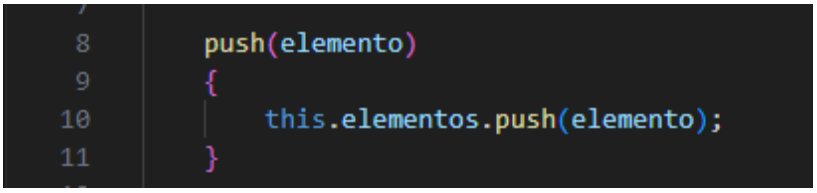
Este es la forma en la que se ve el maquetado con estilo de la página.



Creamos el primer archivo llamado pilas.js. Este archivo se encarga de la parte lógica para generar la pila. Creamos una clase llamada Pila que se exporta, dentro de dicha clase creamos un constructor de la clase inicializada una propiedad elementos como un array vacío.



Luego definimos el método push tomando como parámetro elementos. y lo agrega al final del array elementos lo que significa que se coloca en la parte superior de la pila.



Definimos el método pop, dentro del método creamos una condición if con la condición de para verificar si la pila está vacía o no. Se evaluará como verdadera, lo que significa que hay elementos en la pila que se pueden retirar.

```
13      pop()
14      {
15          if (!this.estaVacia())
16          {
17              return this.elementos.pop();
18          }
19          return null;
20      }
```

Por último, creamos el método llamado estaVacia. Este método verifica la longitud del array elementos que almacena los elementos de la pila.

```
22      estaVacia()
23      {
24          return this.elementos.length === 0;
25      }
26  }
```

Creamos el segundo archivo llamado SumarNumero.js. Este archivo almacena toda la parte lógica de la suma de dos números con pilas. Dentro del archivo comenzamos importando la clase Pila desde el archivo pilas.js y ya luego definimos una función llamada sumarNumerosGrandes en la que toma dos números de entrada: num1 y num2. Se crea tres pilas para cada número y otra para el resultado y por último inicializamos una variable acarreo en 0 que nos sirvió para llevar cualquier valor que se haya sumado de la posición anterior durante la suma.

JS *SumarNumero.js* ×

JS SumarNumero.js > ...

```
1   import { Pila } from './pilas.js';
2
3   export function sumarNumerosGrandes(num1, num2)
4   {
5       const pila1 = new Pila();
6       const pila2 = new Pila();
7       const pilaResultado = new Pila();
8       let acarreo = 0;
```

Seguidamente creamos dos bucles for: el primer for recorre cada digito en el número num1 para iterar sobre los caracteres individuales del número num1. Dentro del bucle agregamos cada dígito a la pila pila1. La función que se encuentra emparentéis del push convierte en un número entero, La base numérica se establece en 10 para asegurar que se interprete correctamente como un número decimal. Luego, estos dígitos convertidos se agregan a una pila llamada pila1. Lo mismo hacemos con el segundo bucle solo que con el num2 y la pila2.

```
10      for (const digito of num1)
11      {
12          pila1.push(parseInt(digito, 10));
13      }
14
15      for (const digito of num2)
16      {
17          pila2.push(parseInt(digito, 10));
18      }
19
```

Creamos un bucle while con ciertas condiciones: La pila1 no está vacia, la pila2 no está vacia o si el acarreo es mayor a 0. Si cumple con cualquiera de las tres condiciones se inicializa el while. Dentro del while se realiza una suma de números grandes, representados como pilas de dígitos. Se extraen los dígitos de cada pila, teniendo en cuenta un posible acarreo de sumas anteriores, y se realiza la suma de dígitos en cada posición. El resultado se almacena en la pila pilaResultado, y se calcula el nuevo acarreo para la siguiente iteración. El proceso continúa hasta que se procesen todos los dígitos de ambas pilas, y el resultado final es construido en pilaResultado.

```
20      while (!pila1.estaVacia() || !pila2.estaVacia() || acarreo > 0)
21      {
22          const digito1 = pila1.pop() || 0;
23          const digito2 = pila2.pop() || 0;
24          const suma = digito1 + digito2 + acarreo;
25          acarreo = Math.floor(suma / 10);
26          pilaResultado.push(suma % 10);
27      }
```

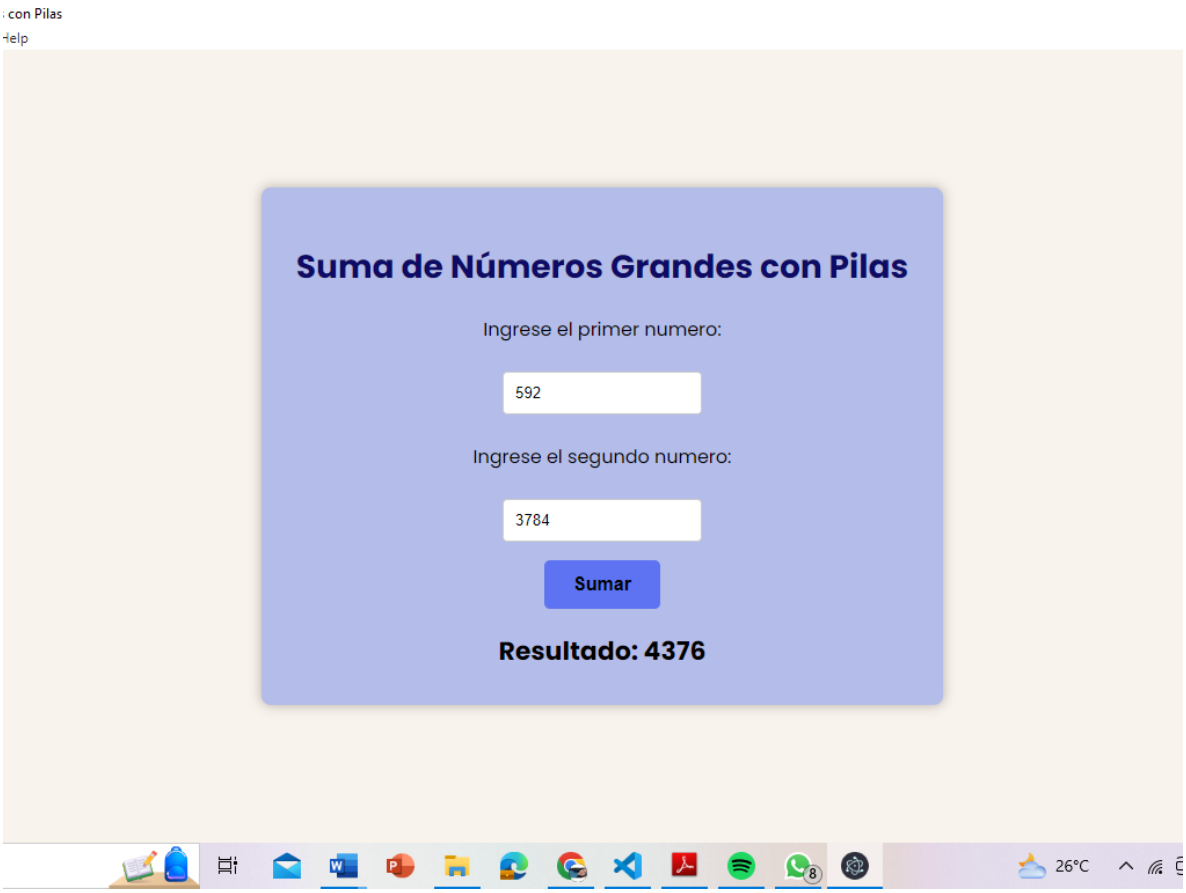
Por último, creamos una variable llamado resultado como una cadena vacía. Luego se inicializa un bucle while que se ejecuta mientras la pilaResultado no este vacía, se extrae el digitó en la parte superior de la pilaResultado usando pop que devuelve el digito y lo elimina de la pila y el digito extraído se agrega al final de la cadena resultado utilizando el operador +=. Una vez que la pila este vacía y se han procesado todos los dígitos del bucle termina y se retorna el resultado.

```
29      let resultado = '';
30
31      while (!pilaResultado.estaVacia())
32      {
33          resultado += pilaResultado.pop();
34      }
35
36      return resultado;
37  }
38
```

Creamos el ultimo archivo llamado script.js se encarga de la interacción con el DOM. Asegúrate de importar estos archivos correctamente en tu HTML para que la aplicación funcione. Dentro del archivo se importó la función sumaNumeroGrandes desde el archivo llamado SumarNumero.js. selecciona el botón con el id "sumarBoton" en el documento HTML y le agrega un oyente de eventos de clic. Cuando se hace clic en el botón, se ejecuta una función anónima que recopila los valores de dos campos de entrada de texto ("numero1" y "numero2"), realiza una suma de números grandes llamando a la función sumarNumerosGrandes, y finalmente muestra el resultado en el elemento con el id "resultado" en la página web. El resultado se presenta como un mensaje de texto precedido por la palabra "Resultado".

```
JS script.js
JS script.js > ...
1   import { sumarNumerosGrandes } from './SumarNumero.js';
2
3   document.getElementById('sumarBoton').addEventListener('click', function() {
4       const num1 = document.getElementById('numero1').value;
5       const num2 = document.getElementById('numero2').value;
6       const resultado = sumarNumerosGrandes(num1, num2);
7       document.getElementById('resultado').textContent = `Resultado: ${resultado}`;
8   });
9
```

Le agregamos estilo con CSS y lo mandamos a aplicación con Electron



PRÁCTICA 9

Comenzamos creando el HTML:

ejercicio9 > index.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/style.css">
    <title>Reemplazo en Pila</title>
</head>
<body>
    <h1>Reemplazo en Pila</h1>
    <div class="input-container">
        <input type="text" id="valorNuevo" placeholder="Valor Nuevo">
        <input type="text" id="valorViejo" placeholder="Valor Viejo">
        <button id="reemplazar">Reemplazar</button>
    </div>
    <div class="resultado-container">
        <p>Pila:</p>
        <div id="resultado"></div>
    </div>
    <script type="module" src="js/app.js"></script>
</body>
</html>
```

Proseguimos con la parte lógica de la práctica, está la clase llamada Pila que toma una lista de 'elementos' vacía. Están cada una de las diferentes cosas que puede suceder con esos elementos, está el método *apilar* que usa push para agregar ese elemento a elementos, luego está *pop* que elimina ese elemento, también está *estaVacia* que toma la lista de elementos y vuelve su tamaño a 0 y, por último, está función obtenerElementos que toma esa lista de elementos.

ejercicio9 > js > pila.js > Pila

```js
export class Pila {
    constructor() {
        this.elementos = [];
    }

    apilar(elemento) {
        this.elementos.push(elemento);
    }

    desapilar() {
        return this.elementos.pop();
    }

    estaVacia() {
        return this.elementos.length === 0;
    }

    obtenerElementos() {
        return [...this.elementos];
    }
}
```

Después seguimos con la otra parte lógica de la práctica, el archivo app que interactúa con el usuario, comenzamos con crear varias constantes que llaman especificas partes del DOM. Está la creación y llenado de la pila original, en donde usan la función de apilar para agregar cada elemento a la pila.

```
ejercicio9 > js > JS app.js > ...
1   import { Pila } from './pila.js';
2
3   const valorNuevoInput = document.getElementById('valorNuevo');
4   const valorViejoInput = document.getElementById('valorViejo');
5   const reemplazarBtn = document.getElementById('reemplazar');
6   const resultadoContainer = document.getElementById('resultado');
7
8   const pilaOriginal = new Pila();
9   pilaOriginal.apilar(1);
10  pilaOriginal.apilar(2);
11  pilaOriginal.apilar(3);
12  pilaOriginal.apilar(4);
13  pilaOriginal.apilar(3);
14  pilaOriginal.apilar(5);
```

Esta pila original usa la función de imprimirPilaOriginal, que como su nombre dice, imprime esta pila y lo hace obteniendo, primeramente, los elementos de la pila (usando la función obtenerElementos), esta función agrega al contenedor de resultados cada elemento que exista en la pila original. Y al final, esta pila se imprime.

```
16  function imprimirPilaOriginal(pila){
17      const elementos = pila.obtenerElementos();
18      elementos.forEach(elemento => {
19          const elementoDiv = document.createElement('div');
20          elementoDiv.textContent = elemento;
21          resultadoContainer.appendChild(elementoDiv);
22      });
23  }
24
25  imprimirPilaOriginal(pilaOriginal);
```

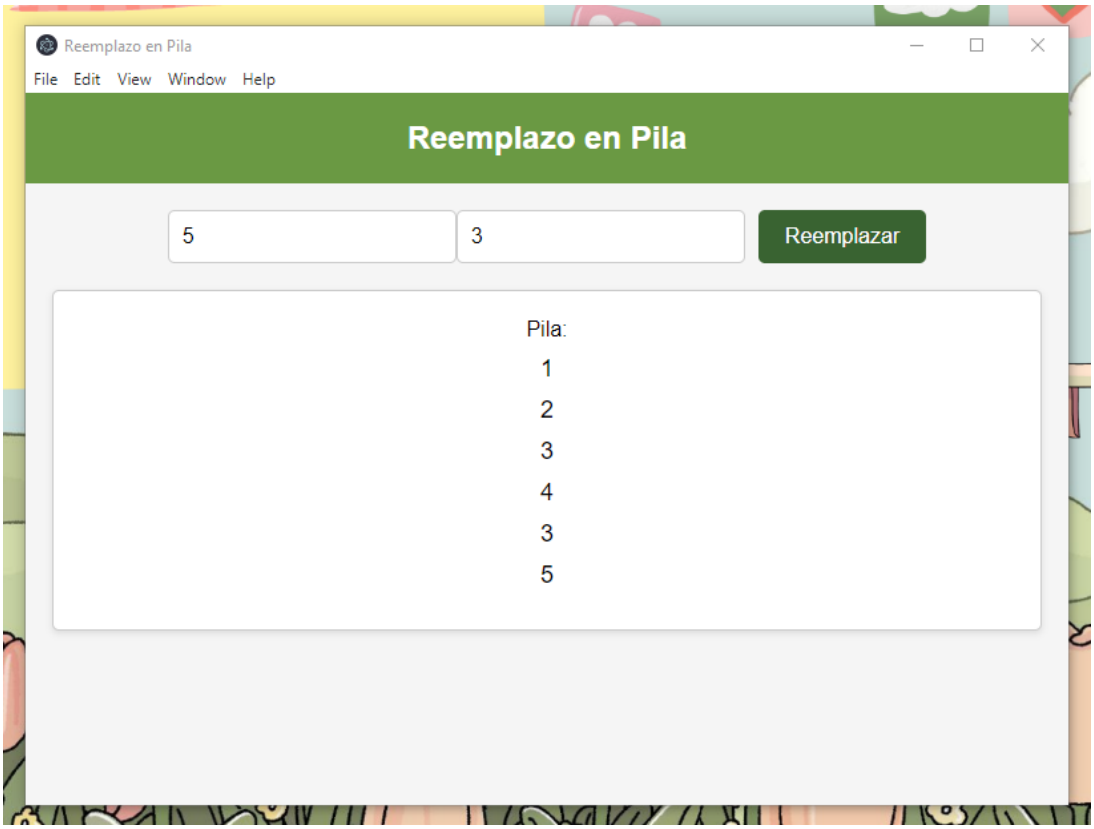
Luego, está la función de renderizarPila que toma la pila, y hace algo similar a la función anterior, pero esta borra el contenido previo del contenedor, es como si se actualizara la lista.

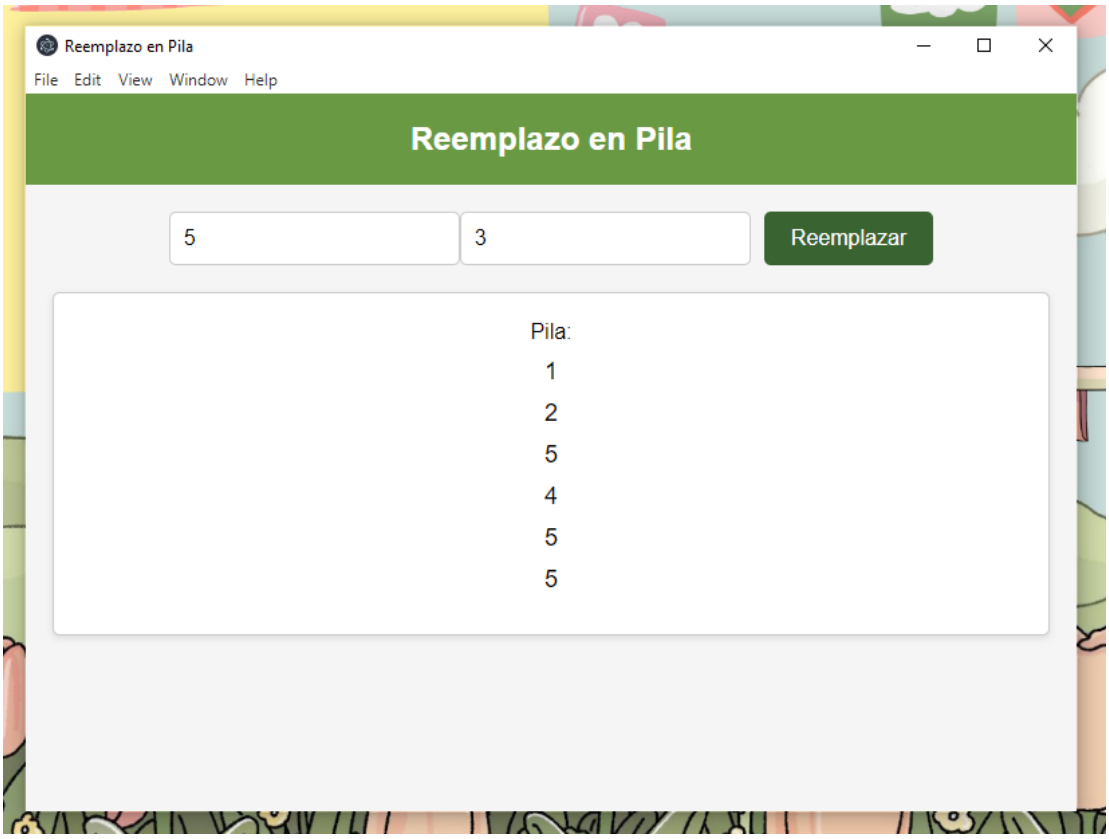
```
function renderizarPila(pila) {
    resultadoContainer.innerHTML = '';
    const elementos = pila.obtenerElementos();
    elementos.forEach(elemento => {
        const elementoDiv = document.createElement('div');
        elementoDiv.textContent = elemento;
        resultadoContainer.appendChild(elementoDiv);
    });
}
```

Por último, esta el botón que genera un evento, este comienza creando los variables que obtiene el valor viejo y nuevo, y los convierte a tipo enteros. Se crea una variable pilaTemporal que mediante bucles se desapilan los elementos de la pila original que coincidan con los valores viejos. Si son iguales, se apila el valor nuevo en la pila temporal; de lo contrario, se apila el elemento original en la pila temporal. Luego, se transfieren los elementos de la pila temporal de vuelta a la pila original. Finalmente, se llama a renderizarPila para mostrar la pila original actualizada en el contenedor de resultados.

```
reemplazarBtn.addEventListener('click', () => {
    const valorNuevo = parseInt(valorNuevoInput.value);
    const valorViejo = parseInt(valorViejoInput.value);

    const pilaTemporal = new Pila();

    while (!pilaOriginal.estaVacia()) {
        const elemento = pilaOriginal.desapilar();
        if (elemento === valorViejo) {
            pilaTemporal.apilar(valorNuevo);
        } else {
            pilaTemporal.apilar(elemento);
        }
    }

    while (!pilaTemporal.estaVacia()) {
        pilaOriginal.apilar(pilaTemporal.desapilar());
    }

    renderizarPila(pilaOriginal);
});
```

Le agregamos estilo con CSS y lo mandamos a aplicación con Electron


Reemplazo en Pila
File Edit View Window Help
Reemplazo en Pila
5
3
Reemplazar
Pila:
1
2
3
4
3
5


Reemplazo en Pila
File Edit View Window Help
Reemplazo en Pila
5
3
Reemplazar
Pila:
1
2
5
4
5
5