HITO 1 LENGUAJES DE MARCAS

3º TRIMESTRE, 1º DAM

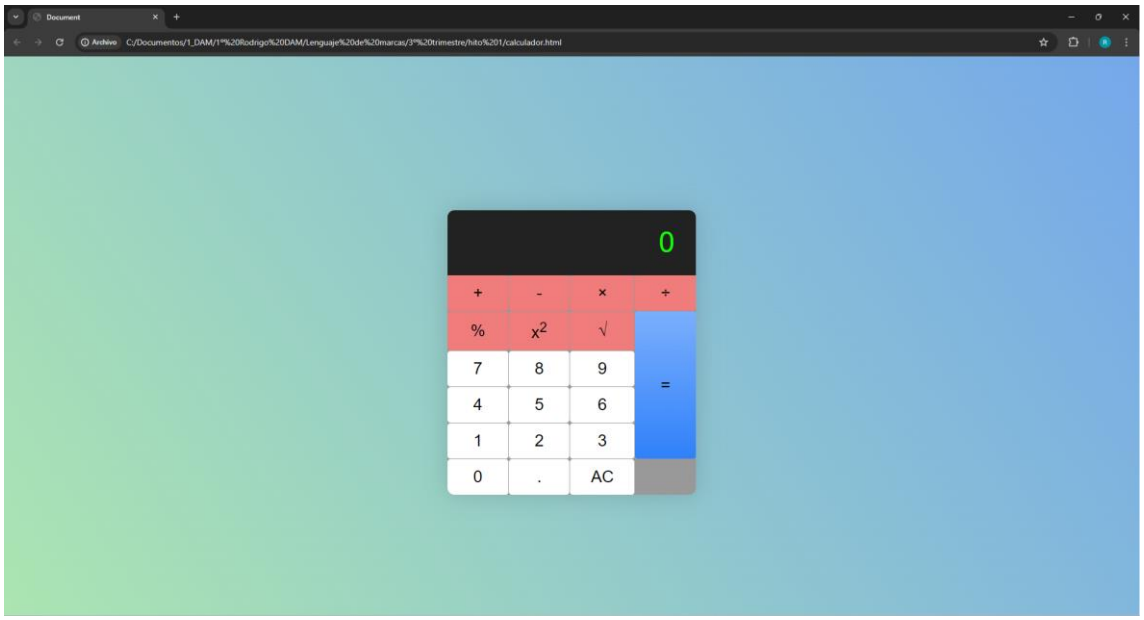
Descripción

JavaScript puro, manipulación de JSON con JavaScript en la web y redacción de una memoria del proyecto.

Rodrigo Díaz Dones
Rodrigo.diazdones.23@campusfp.es

Contenido

CALCULADORA



La creación de este programa de calculadora en JavaScript implica varios pasos y consideraciones para su funcionamiento, utiliza eventos del DOM y lógica de programación para permitir al usuario realizar operaciones matemáticas básicas y algunas funciones adicionales de manera interactiva y eficiente. A continuación, se detallan los pasos y la utilidad de cada bloque de código:

Selección de elementos del DOM: Selecciona los elementos del DOM necesarios para interactuar con la calculadora, como el contenedor principal, las teclas y el display de la calculadora. Esto permite manipular y mostrar los valores correctamente durante las operaciones.

Event listener para las teclas: Se añade un event listener a las teclas de la calculadora para detectar cuándo se hace clic en alguna de ellas. Esto permite capturar la acción del usuario y realizar las operaciones correspondientes.

Lógica de las operaciones básicas: Se implementa la lógica para realizar las operaciones básicas de suma, resta, multiplicación y división. Esto se realiza mediante una función calculate que recibe los números y el operador como parámetros y devuelve el resultado de la operación.

Lógica de la tecla de decimal: Se implementa la lógica para añadir el punto decimal al número mostrado en el display. Esto incluye verificar si el número ya tiene un punto decimal y si se debe añadir uno nuevo o reemplazar el número actual.

Lógica de la tecla de borrado (clear): Se implementa la lógica para borrar el contenido del display y restablecer el estado de la calculadora a su valor inicial. Esto se hace verificando si se debe limpiar el display o restablecer completamente la calculadora.

Lógica de las teclas de operadores y calcular: Se implementa la lógica para realizar operaciones matemáticas y calcular el resultado. Esto incluye el manejo de los valores anteriores y la actualización del display con el resultado de la operación.

Funcionalidades adicionales: Se añaden funcionalidades adicionales como el cálculo del porcentaje, el cuadrado y la raíz cuadrada. Estas funciones amplían las capacidades de la calculadora para realizar más operaciones matemáticas

A continuación paso a detallar de manera más detenida el funcionamiento del código.

Código:

La primera parte del código se enfoca en obtener referencias a elementos HTML relevantes dentro de la calculadora. Se utiliza document.querySelector() para seleccionar elementos por su clase. 'calculator' es el contenedor principal, 'keys' contiene todas las teclas, y 'display' muestra el resultado. Se usan selectores CSS para cada elemento: '.calculator__keys' para las teclas y '.calculator__display' para el display. Estos métodos retornan los elementos correspondientes en el documento HTML

```
const calculator = document.querySelector('.calculator');
const keys = calculator.querySelector('.calculator__keys');
const display = document.querySelector('.calculator__display');
```

El siguiente fragmento de código actúa como un manejador de eventos que se dispara al hacer clic en cualquier elemento dentro del contenedor 'keys'. Comienza estableciendo un evento de clic en 'keys', donde se verifica si el elemento clicado es un botón. Luego, se extrae información relevante del botón clicado, como su acción asociada y el número actual en la pantalla de la calculadora. Dependiendo de la acción del botón, se realizan diversas operaciones matemáticas, como actualizar la pantalla con un número, agregar un decimal, limpiar la pantalla, realizar operaciones como suma, resta, multiplicación y división, calcular el resultado final de la operación actual, obtener el porcentaje del número actual, o calcular el cuadrado del número mostrado.

```
keys.addEventListener('click', e => {
    if (e.target.matches('button')) {
        const key = e.target;
        const action = key.dataset.action;
        const keyContent = key.textContent;
        const displayedNum = display.textContent;

        if (!action) {
            if (displayedNum === '0' ||
calculator.dataset.previousKeyType === 'operator' ||
calculator.dataset.previousKeyType === 'calculate') {
                display.textContent = keyContent;
            } else {
                display.textContent = displayedNum + keyContent;
```

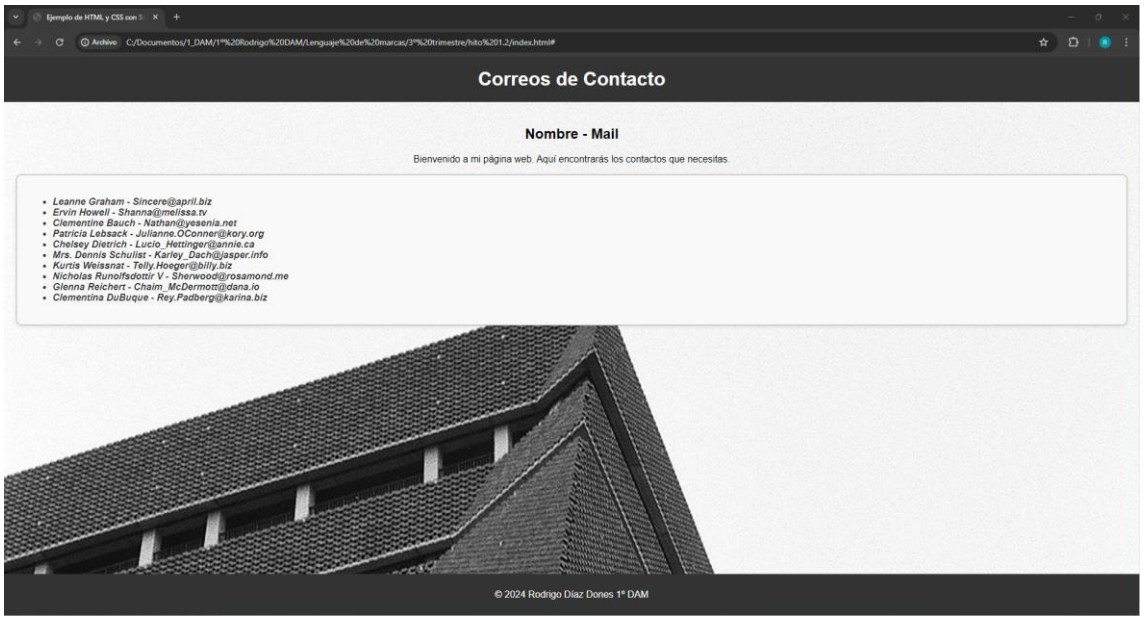
```
            }
            calculator.dataset.previousKeyType = 'number';
        } else if (action === 'decimal') {
            if (!displayedNum.includes('.')) {
                display.textContent = displayedNum + '.';
            } else if (calculator.dataset.previousKeyType === 'operator'
|| calculator.dataset.previousKeyType === 'calculate') {
                display.textContent = '0.';
            }
            calculator.dataset.previousKeyType = 'decimal';
        } else if (action === 'clear') {
            if (key.textContent === 'AC') {
                calculator.dataset.firstValue = '';
                calculator.dataset.modValue = '';
                calculator.dataset.operator = '';
                calculator.dataset.previousKeyType = '';
            } else {
                key.textContent = 'AC';
            }
            display.textContent = 0;
            calculator.dataset.previousKeyType = 'clear';
        } else if (
            action === 'add' ||
            action === 'subtract' ||
            action === 'multiply' ||
            action === 'divide'
        ) {
            const firstValue = calculator.dataset.firstValue;
            const operator = calculator.dataset.operator;
            const secondValue = displayedNum;

            if (
                firstValue &&
                operator &&
                calculator.dataset.previousKeyType !== 'operator' &&
                calculator.dataset.previousKeyType !== 'calculate'
            ) {
                const calcValue = calculate(firstValue, operator,
secondValue);
                display.textContent = calcValue;
                calculator.dataset.firstValue = calcValue;
            } else {
                calculator.dataset.firstValue = displayedNum;
            }

            key.classList.add('is-depressed');
            calculator.dataset.previousKeyType = 'operator';
            calculator.dataset.operator = action;
        } else if (action === 'calculate') {
```

```
            let firstValue = calculator.dataset.firstValue;
            const operator = calculator.dataset.operator;
            const secondValue = displayedNum;

            if (firstValue) {
                if (calculator.dataset.previousKeyType === 'calculate') {
                    firstValue = displayedNum;
                }
                display.textContent = calculate(firstValue, operator,
secondValue);
            }

            calculator.dataset.previousKeyType = 'calculate';
        } else if (action === 'percentage') {
            const percentageValue = parseFloat(displayedNum) / 100;
            display.textContent = percentageValue;
            calculator.dataset.previousKeyType = 'percentage';
        } else if (action === 'square') {
            const squareValue = parseFloat(displayedNum) ** 2;
            display.textContent = squareValue;
            calculator.dataset.previousKeyType = 'square';
        } else if (action === 'square-root') {
            const squareRootValue = Math.sqrt(parseFloat(displayedNum));
            display.textContent = squareRootValue;
            calculator.dataset.previousKeyType = 'square-root';
        }
    }
});
```

La función 'calculate' toma tres argumentos: 'n1', 'operator' y 'n2'. Convierte 'n1' y 'n2' en números decimales usando 'parseFloat()'. Luego, utiliza un condicional 'if' para determinar la operación a realizar según 'operator': suma, resta, multiplicación o división. Realiza la operación correspondiente entre 'firstNum' y 'secondNum', devolviendo el resultado. Esto asegura operaciones precisas incluso con valores de entrada como cadenas de texto.

```
const calculate = (n1, operator, n2) => {
    const firstNum = parseFloat(n1);
    const secondNum = parseFloat(n2);
    if (operator === 'add') return firstNum + secondNum;
    if (operator === 'subtract') return firstNum - secondNum;
    if (operator === 'multiply') return firstNum * secondNum;
    if (operator === 'divide') return firstNum / secondNum;
};
```

PETICIÓN FETCH AL API REST



Este programa combina HTML, CSS y JavaScript para crear una página web que muestra una lista de nombres y correos electrónicos obtenidos de una API externa. A continuación, se detallan los pasos para crear este programa y la utilidad de cada bloque de código:

Por un lado se define la estructura básica del documento HTML con etiquetas como <!DOCTYPE html>, <html>, <head>, <meta>, <title>, <link>, <body>, <header>, <section>, <footer>. Entre ellos se incluye un enlace a un archivo de estilos externo (styles.css) para aplicar estilos a la página.

Dentro del HTML Se crea un encabezado (<header>) con el título "Correos de Contacto", además se agrega una sección principal (<section>) con un título secundario, un párrafo de bienvenida y un contenedor ( ) con un identificador app-container.

Por último se incluye un pie de página (<footer>) con información de derechos de autor.

Por otro lado, se crea un JavaScript para obtener datos de la API, para ello se declara una constante API_URL que contiene la URL de la API de la cual se obtendrán los datos. Para ello, se selecciona el elemento HTML con el id app utilizando document.querySelector("#app").

Para obtener los datos de usuarios se realiza una solicitud (fetch) a la URL de la API, Después se convierte la respuesta en formato JSON y se utiliza el método then para manejar los datos obtenidos, Y a continuación se utiliza el método map para recorrer la lista de usuarios y crear un nuevo array con elementos <li> que contienen el nombre y el correo electrónico de cada usuario.

Finalmente, se actualiza el contenido del elemento app con la lista de usuarios generada.

Código:

Las primeras líneas de este JS definen una constante llamada API_URL que almacena la dirección URL de una API. En este caso, la URL apunta a la API JSONPlaceholder, específicamente a la ruta /users, que devuelve datos simulados de usuarios en formato JSON.

A continuación se utiliza el método document.querySelector() para seleccionar un elemento HTML del DOM. El selector #app indica que se está buscando un elemento con el atributo id igual a "app". El elemento seleccionado se almacenará en la variable HTMLResponse para su posterior manipulación.

```
const API_URL = "https://jsonplaceholder.typicode.com/users";

const HTMLResponse = document.querySelector("#app");
```

En la segunda parte del código, se emplea fetch() para obtener datos de una API remota y manipula la respuesta. La función fetch() realiza una solicitud GET a una URL especificada en la constante API_URL, que apunta a una API que devuelve datos de usuarios en formato JSON. Luego, se encadenan dos métodos .then() para manejar la respuesta de la promesa devuelta por fetch(). El primero convierte la respuesta en un objeto JavaScript utilizando json(), y el segundo recibe los datos obtenidos de la respuesta JSON.

Posteriormente, se utiliza map() para recorrer el array de usuarios y generar elementos de lista HTML (<li>) que contienen el nombre y el email de cada usuario. Se crea una plantilla de lista que contiene estos elementos generados.

Finalmente, se actualiza el contenido HTML del elemento identificado por #app, utilizando innerHTML para asignar una lista desordenada (<ul>) que contiene los elementos de lista generados, utilizando join() para unir los elementos del array en una cadena de texto.

```
fetch(API_URL)
    .then((response) => response.json())
    .then((users) => {
        const tpl = users.map(user => `<li>${user.name} - ${user.email}</li>`);
        HTMLResponse.innerHTML = `<ul>${tpl.join('')}</ul>`;
    });
```