HITO 1 DEL 3ER TRIMESTRE DE LM

Memoria académica

Valentín Cortés Amadín

30/04/2024

Contenidos

Fase 1: JavaScript....3

El archivo ‘calculadora.js’....3

El archivo ‘display.js’....3

El archivo ‘index.html’....4

El archivo ‘index.css’....4

El archivo ‘index.js’....4

La función porcentaje ‘%’....5

La función cuadrado ‘^2’....7

La función raíz cuadrada ‘ √ ’....9

Ajustando los iconos en la calculadora....11

Fase 2: JSON....13

El archivo ‘script.js’....13

El archivo ‘index.html’....15

El archivo ‘styles.css’....16

Webgrafía....17

Fase 1: JavaScript

La primera fase del hito individual consistía en realizar un trabajo de investigación sobre un proyecto de una calculadora. El código estaba prácticamente hecho y tan solo había que realizar un par de modificaciones.

El archivo ‘calculadora.js’

Este archivo contiene las definiciones de las funciones que la calculadora puede realizar. En este caso las funciones disponibles son suma, resta, multiplicación, división, porcentaje, potencia cuadrada y raíz cuadrada. Cada función realiza una operación con el/los valores que recibe por entrada y retorna el resultado. Las funciones están recogidas dentro de la clase “Calculadora”:

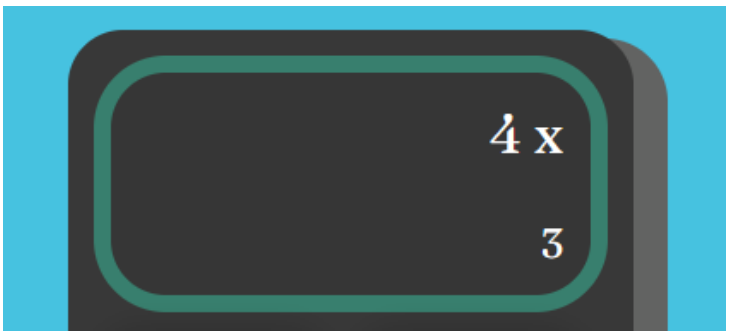
```
class Calculadora {
    sumar(num1, num2) {
        return num1 + num2;
    }

    restar(num1, num2) {
        return num1 - num2;
    }

    dividir(num1, num2) {
        return num1 / num2;
    }
```

El archivo ‘display.js’

Este archivo contiene todo lo relativo a lo que sucede en la pantalla de la calculadora. Incluye líneas de código para visualizar tanto los números tecleados en la entrada como los símbolos, y también recoge declaraciones y definiciones de las funciones encargadas de agregar números u operadores y borrar uno o todos los elementos de la pantalla, así como de computar las operaciones y mostrar el resultado.



El archivo ‘index.html’

Este archivo contiene la estructura web en la que se sitúa la calculadora. Contiene las declaraciones de todos los botones de la calculadora y del espacio dedicado a los caracteres que aparecen en la pantalla. Asimismo, en este archivo convergen todos los demás archivos que conforman la calculadora.

El archivo ‘index.css’

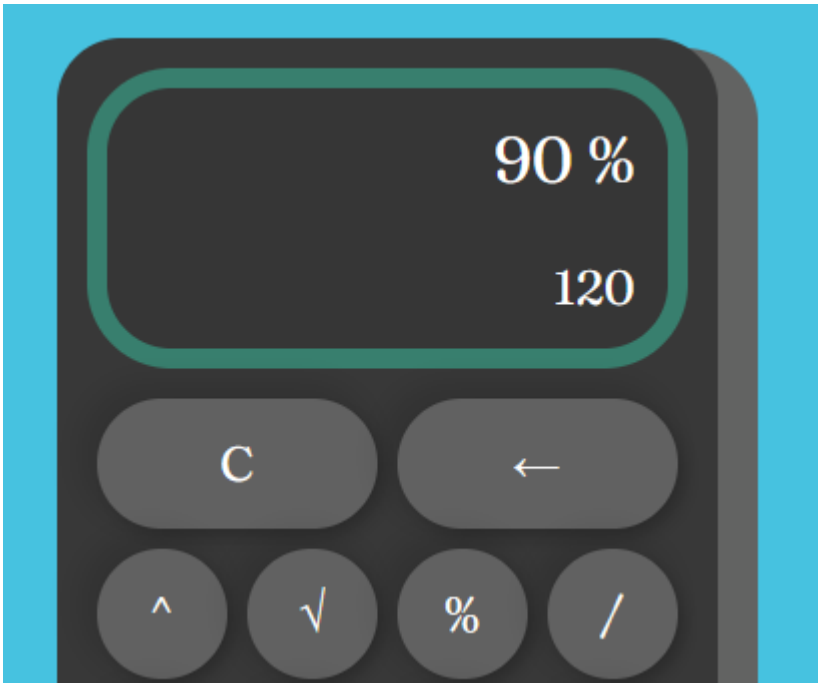
Este archivo se encarga de aportar el diseño y la estética a la calculadora. Define las dimensiones de la pantalla y la botonera, así como el cuerpo de la calculadora.

El archivo ‘index.js’

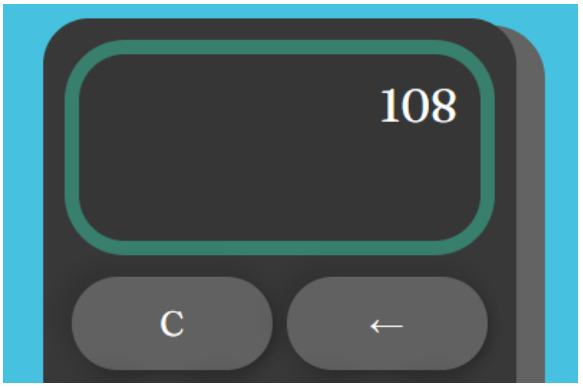
Este archivo es responsable de la lectura de los elementos que introducimos por línea de entrada. Sin este archivo sería imposible que la calculadora reconociera y procesara los dígitos de las operaciones.

La función porcentaje ‘%’

El operador ‘%’ realiza el cálculo de un valor porcentaje sobre otro valor introducido. Para realizar el cálculo primero se inserta el porcentaje y a continuación el valor sobre el cual queremos aplicar dicho porcentaje. Efectuamos el cálculo y nos devuelve el valor al cuál este equivale. Veamos un ejemplo de uso:



En este caso queremos calcular a cuánto equivale el 90% de 120. Para ello introducimos el valor sobre el que queremos aplicar el porcentaje (120), a continuación el símbolo % y finalmente el valor de dicho porcentaje (90). Al pulsar ‘=’ obtendremos el valor correspondiente al 90% de 120, que es 108.



Para implementar esta función en la calculadora primero añadimos la función **porcentaje()** en el código JavaScript de **Calculadora.js**, que tome como valores de entrada dos valores y nos devuelva el resultado de la operación:

```
18      porcentaje(num1, porcentaje) {
19          return (num1 * porcentaje) / 100;
20      }
```

El siguiente paso es añadir el símbolo del porcentaje a la pantalla de la calculadora para que este aparezca cuando pulsemos el botón. Para ello añadimos una línea en la clase Display dentro del archivo **Display.js**:

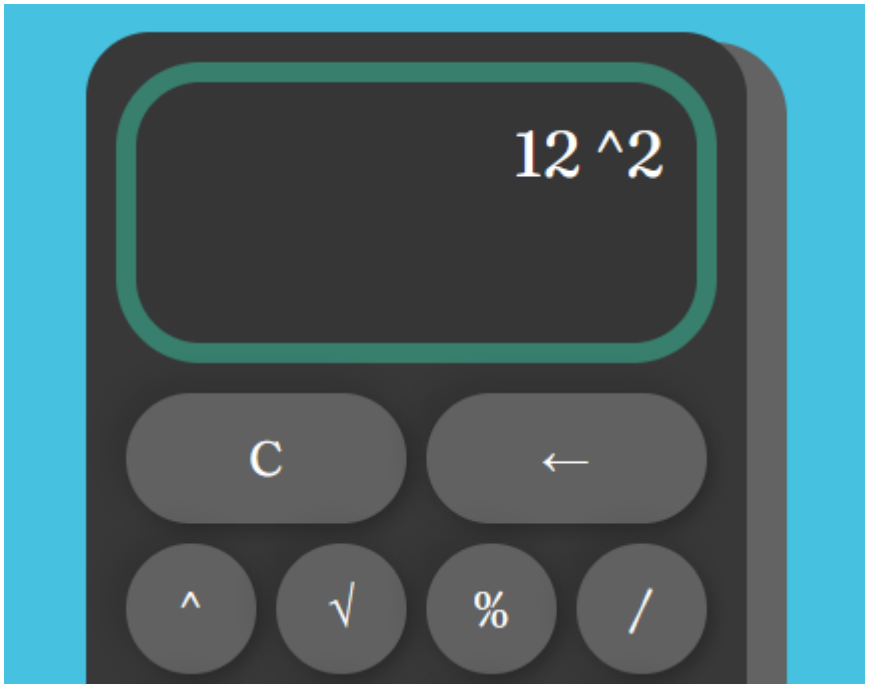
```
class Display {
    constructor(displayValorAnterior, displayValorActual) {
        this.displayValorActual = displayValorActual;
        this.displayValorAnterior = displayValorAnterior;
        this.calculador = new Calculadora();
        this.tipoOperacion = undefined;
        this.valorActual = '';
        this.valorAnterior = '';
        this.signos = {
            sumar: '+',
            dividir: '/',
            restar: '-',
            multiplicar: 'x',
            porcentaje: '%',
```

Ahora debemos crear un botón físico en la calculadora con el cuál poder utilizar esta función. Dentro de **Index.html** añadiremos un botón que tenga como clase “operador” y como valor “porcentaje”, seguido de su correspondiente símbolo:

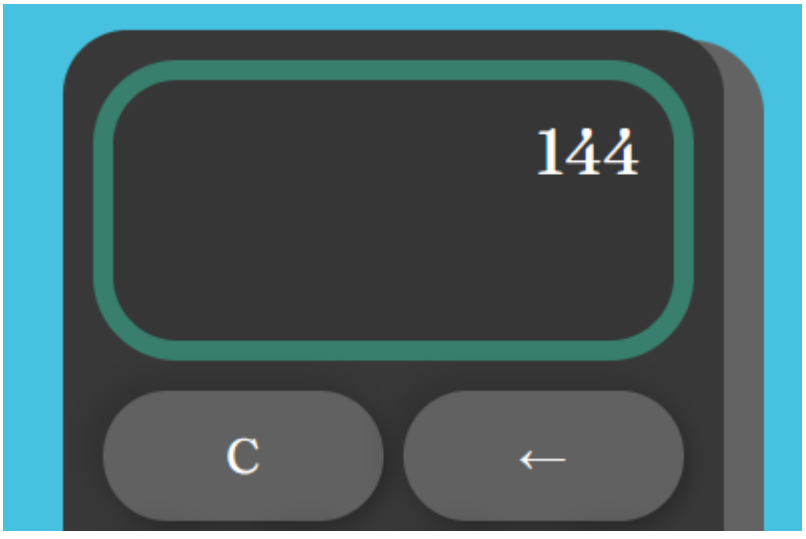
```
<div class="display">
    <div id="valor-anterior"></div>
    <div id="valor-actual"></div>
</div>
<button class="col-2" onclick="display.borrarTodo()">C</button>
<button class="col-2" onclick="display.borrar()">&larr;</button>
<button class="operador" value="cuadrado">^</button>
<button class="operador" value="raizCuadrada">√</button>
<button class="operador" value="porcentaje">%</button>
<button class="operador" value="dividir">/</button>
```

La función cuadrado ‘^2’

El operador ‘^2’ calcula el cuadrado del número que es introducido. Por ejemplo, si queremos calcular el cuadrado de 12 primero tecleamos 12, luego pulsaremos en ‘ ^ ’.



Al presionar ‘ = ’ obtendremos 144, que es el resultado de elevar 12 al cuadrado:



Su implementación es muy simple. Primero escribimos en Calculadora.js la función cuadrado(), que tome como valor de entrada el número introducido en pantalla:

```
22      cuadrado(num1) {
23          return num1 * num1;
24      }
```

A continuación haremos que el símbolo sea visible en pantalla tras pulsar el botón. Para ello nos iremos a **display.js** y dentro de la clase Display añadiremos una línea para el signo correspondiente a cuadrado:

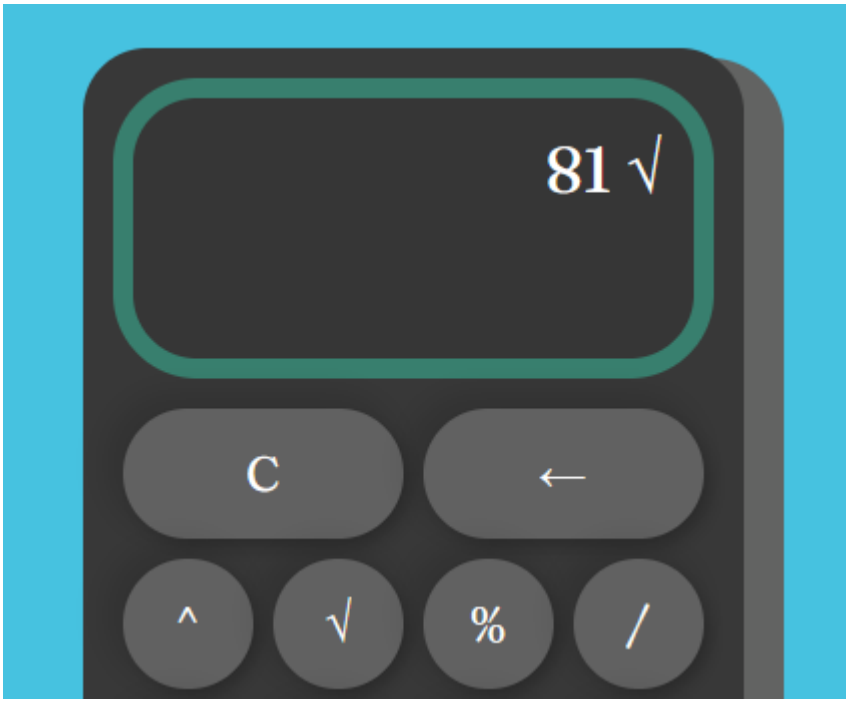
```
class Display {
    constructor(displayValorAnterior, displayValo
        this.displayValorActual = displayValorAct
        this.displayValorAnterior = displayValorA
        this.calculador = new Calculadora();
        this.tipoOperacion = undefined;
        this.valorActual = '';
        this.valorAnterior = '';
        this.signos = {
            sumar: '+',
            dividir: '/',
            restar: '-',
            multiplicar: 'x',
            porcentaje: '%',
            cuadrado: '^2',
```

Por último creamos un botón para esta función. Dentro de Index.html añadimos un botón con clase “operador” y valor “cuadrado”:

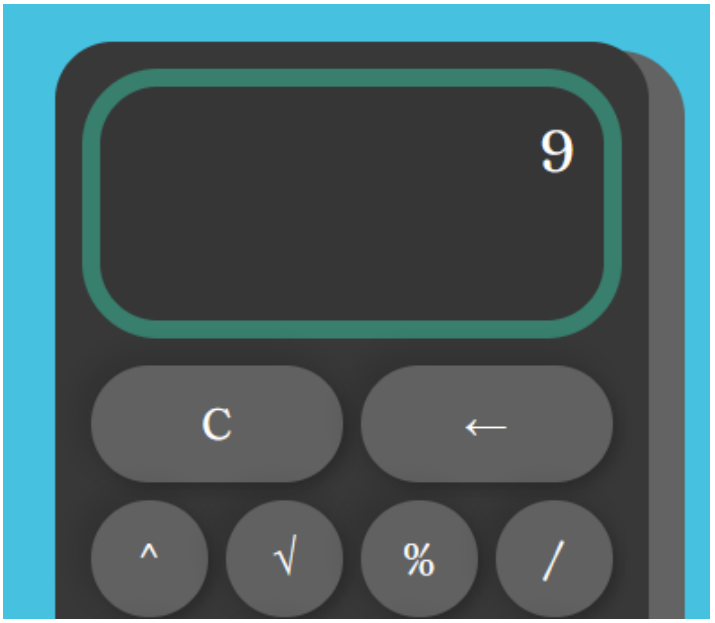
```
<body>
    <div class="container">
        <div class="calculadora">
            <div class="display">
                <div id="valor-anterior"></div>
                <div id="valor-actual"></div>
            </div>
            <button class="col-2" onclick="display.borrarTodo()">C</button>
            <button class="col-2" onclick="display.borrar()">&larr;</button>
            <button class="operador" value="cuadrado">^</button>
            <button class="operador" value="raizCuadrada">√</button>
            <button class="operador" value="porcentaje">%</button>
```

La función raíz cuadrada ‘ √ ’

Esta función, como bien indica su nombre, efectúa la raíz cuadrada del valor introducido en pantalla. Por ejemplo, si quisiéramos calcular la raíz cuadrada de 81, debemos teclear primero 81 y, seguidamente, el símbolo √.



Tras pulsar en ‘ = ’ obtendremos el resultado de dicha operación, que es 9.



No es posible efectuar raíces cuadradas de números negativos, ya que esta simple calculadora está diseñada para aceptar únicamente valores positivos y no ha sido adecuada para reconocer números complejos.

Al igual que en las dos funciones vistas anteriormente el procedimiento para implementar esta función es muy sencillo. Primero escribimos dentro de **Calculadora.js** la función raizCuadrada() que calcule la raíz cuadrada del número introducido por pantalla, haciendo uso de la librería Math() ya incluída por defecto en JavaScript:

```
    raizCuadrada(num1) {
        return Math.sqrt(num1);

    }
```

A continuación debemos de hacer visible el símbolo √ en la pantalla de la calculadora. Para ello añadimos una línea en **Display.js**:

```
class Display {
    constructor(displayValorAnterior, displayValorActual) {
        this.displayValorActual = displayValorActual;
        this.displayValorAnterior = displayValorAnterior;
        this.calculador = new Calculadora();
        this.tipoOperacion = undefined;
        this.valorActual = '';
        this.valorAnterior = '';
        this.signos = {
            sumar: '+',
            dividir: '/',
            restar: '-',
            multiplicar: 'x',
            porcentaje: '%',
            cuadrado: '^2',
            raizCuadrada: '√',
        }
```

Finalmente creamos el botón correspondiente a esta función añadiendo una línea en **Index.html.** Como clase “operador” y como valor “raizCuadrada”:

```
    </div>
    <button class="col-2" onclick="display.borrarTodo()">C</button>
    <button class="col-2" onclick="display.borrar()">&larr;</button>
    <button class="operador" value="cuadrado">^</button>
    <button class="operador" value="raizCuadrada">√</button>
    <button class="operador" value="porcentaje">%</button>
```

Ajustando los iconos en la calculadora

Ya están todas las funciones añadidas, pero ahora debemos reorganizar y redimensionar los botones para que el pad numérico esté centrado y no se vea afectado por la inclusión de las tres nuevas funciones.

Para ello tomamos los botones “Borrar”, “BorrarTodo” e “Igual” y los incluímos en una clase que hemos denominado “col-2”.

```
        <div id="valor-actual"></div>
    </div>
    <button class="col-2" onclick="display.borrarTodo()">C</button>
    <button class="col-2" onclick="display.borrar()">&larr;</button>
    <button class="operador" value="cuadrado">^</button>
    <button class="operador" value="raizCuadrada">√</button>
```

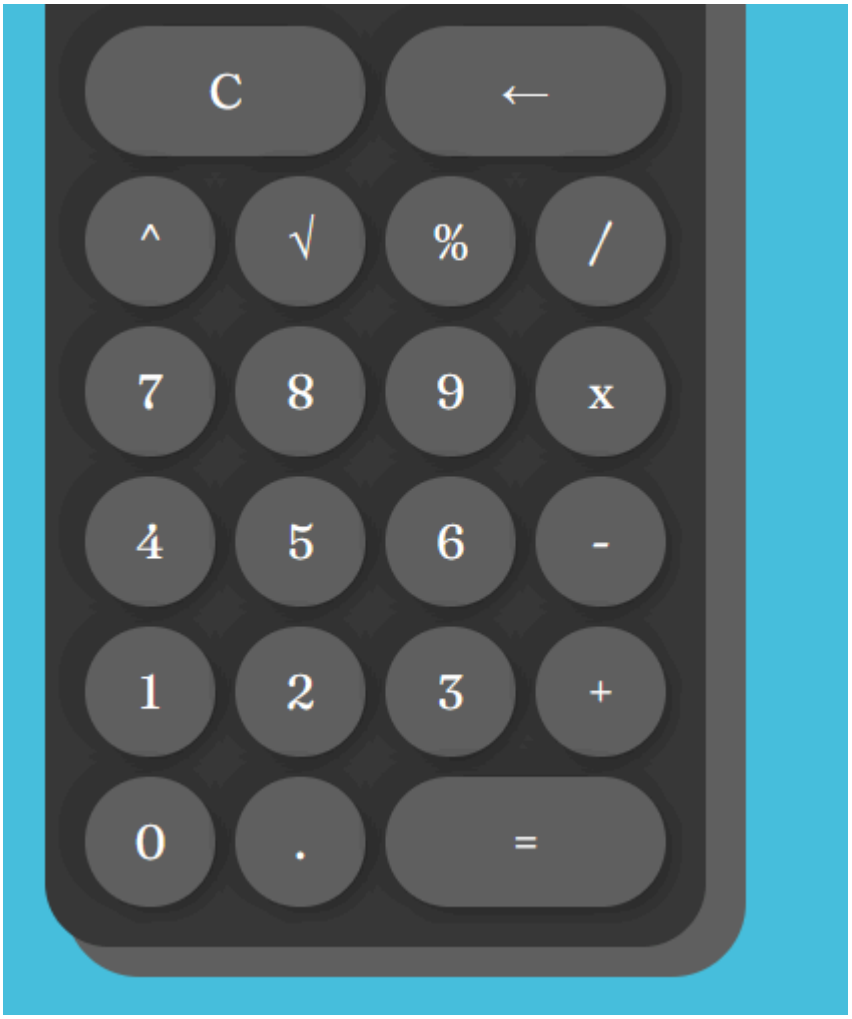
Los botones “Borrar” y “BorrarTodo” conformarán la primera fila de botones, que ocuparán el espacio de 2 botones normales. El botón “Igual” estará en la última fila compartiendo espacio con el 0 y el punto decimal. Les designamos con una clase diferente para ahora poder editar su tamaño en el archivo CSS sin que afecte al resto de la botonera. Haciendo uso de CSS Grid podemos ampliar su dimensión a 2:

```
.col-2 {
    grid-column: span 2;
}
```

El resto de botones de la calculadora mantienen su tamaño estándar tal y como se ha especificado en el documento CSS:

```
button {
    cursor: pointer;
    margin: 5px;
    padding: 0;
    border-radius: 32px;
    font-size: 1.5em;
    border: none;
    background-color: #616161;
    box-shadow: 5px 5px 10px -3px #00000040, -5px -5px 15px 3px #0000001f;
}
```

Y el resultado es el siguiente. Una botonera centrada y organizada, con un pad numérico recogido en un recuadro de 3x3 y el dígito 0 en la columna de abajo:



Fase 2: JSON

En la segunda fase del hito el objetivo es realizar una petición fetch a una API y recopilar los resultados en una página web HTML. En este caso la petición se realiza a la AEMET para obtener datos meteorológicos de todas las provincias de España.

Para llevar a cabo esta tarea necesitaremos tres elementos:

- Un archivo **script.js** que se encargará de ejecutar el fetch a la API y trasladar los resultados al archivo HTML.
- Un archivo **index.html** en el que visualizar los resultados que devuelve la petición fetch.
- Un archivo **styles.css** con el que aportar orden y diseño al archivo HTML, además de aportar funcionalidades de web responsive.

El archivo 'script.js'

Este archivo estará estructurado en dos partes:

1. Primero implementamos el fetch utilizando el evento **addEventListener()**. Este evento en JavaScript se utiliza para adjuntar un manejador de eventos al objeto del documento. Se usa comúnmente para configurar funciones que se llamarán cuando ocurra un cierto evento, como un clic en un botón, el envío de un formulario o la carga de una página. El método `addEventListener` toma dos parámetros requeridos: el evento a escuchar y la función que se llamará cuando ocurra el evento. En este caso el evento a escuchar será el fetch() y la función que será llamada tras el fetch será displayWeatherInfo(), devolviendo los datos que queremos mostrar.

```
document.addEventListener("DOMContentLoaded", function () {
    fetch('https://www.el-tiempo.net/api/json/v2/home')
    .then(response => response.text())
    .then(text => JSON.parse(text))
    .then(data => {
        displayWeatherInfo(data.ciudades);í
    })
    .catch(error => {
        console.error('Error al obtener los datos del tiempo:', error);
    });
```

2. A continuación definimos la función **displayWeatherInfo()**, que primero verificará que existen datos que mostrar y después las almacenará en el archivo HTML usando la función **forEach()** para que de cada ciudad se almacene el nombre, la descripción, la temperatura mínima y la temperatura máxima. La ubicación de los datos insertados viene dado por la función document.**getElementById()** que buscará la estructura **'weather-info**':

```
function displayWeatherInfo(ciudades) {
    const weatherInfoDiv = document.getElementById('weather-info');

    // Este IF verifica que existan datos de tiempo
    if (!ciudades || ciudades.length === 0) {
        weatherInfoDiv.innerHTML = '<p>No se encontraron datos del tiempo disponibles</p>';
        return;
    }

    // Esta variable almacena el HTML que se va a mostrar
    let html = '<h2>El tiempo en varias ciudades:</h2>';
    ciudades.forEach(ciudad => {
        const nombreCiudad = ciudad.name;
        const descripcionCielo = ciudad.stateSky.description;
        const temperaturaMax = ciudad.temperatures.max;
        const temperaturaMin = ciudad.temperatures.min;
```

Finalmente, para cada ciudad insertamos en el HTML una pequeña estructura de los datos e insertamos la información almacenada:

```
    html += `
        <div class="ciudad">
            <h3>${nombreCiudad}</h3>
            <p><strong>Descripción:</strong> ${descripcionCielo}</p>
            <p><strong>Temperatura Máxima:</strong> ${temperaturaMax}ºC</p>
            <p><strong>Temperatura Mínima:</strong> ${temperaturaMin}ºC</p>
        </div>
    `;
});
```

Con la última ciudad se cierra este bucle forEach().

El archivo ‘index.html’

Este archivo almacena la estructura de la página web en el que se mostrará toda la información extraída de la API. Tiene una estructura muy sencilla:

- Primero las líneas que identifican a todo archivo HTML5, además de los links que lo unen al recurso de bootstrap y al archivo CSS que veremos a continuación:

```
<!DOCTYPE html>
<html lang="es">
<meta charset="UTF-8">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Información del Tiempo</title>
    <link rel="stylesheet" href="styles.css">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
```

- Y a continuación el <body>, que contendrá únicamente una sección identificada como “**weather-info**”. Precisamente es aquí donde el archivo “script.js” insertará los datos porque así se especificó en la función document.getElementById(). Por tanto no es necesario indicar la estructura que han de tener los datos en el HTML ya que esa tarea ya la hemos delegado en el archivo “script.js”.

```
</head>
<body>
    <div id="weather-info">
        <!-- Aquí se mostrará la información del tiempo -->
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Y finalmente la llave que vincula el script a la estructura <body> de nuestro archivo HTML.

El archivo ‘styles.css’

Y por último pero no menos importante, tenemos el archivo ‘styles.css’, que será el encargado de que nuestra página web no tenga un aspecto propio de los dominios de 2003, además de organizar los datos en tarjetas y hacer que nuestra página se ajuste a las necesidades de una web responsive.

Su estructura es sencilla. Definimos reglas de estilo para los tres tamaños de fuente utilizados, además de establecer márgenes, bordes y padding para separar en pequeñas tarjetas los datos de cada ciudad:

Webgrafía

https://www.w3schools.com/js/default.asp

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

https://stackoverflow.com/questions/48841097/receive-and-process-json-using-fetch-api-in-javascript

https://meta.stackexchange.com/questions/313951/unable-to-fetch-data-from-the-stack-overflow-api-too-many-requests

https://web.dev/articles/fetch-api-error-handling?hl=es-419

https://dev.to/dionarodrigues/fetch-api-do-you-really-know-how-to-handle-errors-2gj0

https://opendata.aemet.es/centrodedescargas/inicio