Javascript

Semana 5

Índice

Resultado de aprendizaje, contenido y criterios de evaluación

esultado de aprendizaje
RA3. Escribe código, identificando y aplicando las funcionalidades aportadas por los objetos predefinidos del lenguaje.
Contenido
7 - Mecanismos del navegador para el almacenamiento y recuperación de información. 8 - Depuración y documentación del código.
Criterios de evaluación
g) Se han utilizado mecanismos del navegador web para almacenar información y recuperar su contenido. h) Se ha depurado y documentado el código

Resultado de aprendizaje
R4. Programa código para clientes web analizando y utilizando estructuras definidas por el usuario
Contenido
5 - Creación de objetos. 6 - Definición de métodos y propiedades
Criterios de evaluación
g) Se ha creado código para definir la estructura de objetos. h) Se han creado métodos y propiedades k) Se ha depurado y documentado el código

Resultado de aprendizaje
RA5. Desarrolla aplicaciones web interactivas integrando mecanismos de manejo de eventos
Contenido
2 - Utilización de formularios desde código. 3 - Modificación de apariencia y comportamiento. 4 - Validación y envío. 5 - Expresiones regulares. 6 - Prueba y documentación del código
Criterios de evaluación
d) Se ha creado un código que capture y utilice eventos e) Se han reconocido las capacidades del lenguaje relativas a la gestión de formularios web. f) Se han validado formularios web utilizando eventos g) Se han utilizado expresiones regulares para facilitar los procedimientos de validación h) Se ha probado y documentado el código

Resultado de aprendizaje
RA6. Desarrolla aplicaciones web analizando y aplicando las características del modelo de objetos del documento
Contenido
4 - Programación de eventos 5 - Diferencias en las implementaciones del modelo 6 - Independencia de las capas de implementación de aplicaciones web
Criterios de evaluación
d) Se han creado nuevos elementos de la estructura y modificado elementos ya existentes. e) Se han asociado acciones a los eventos del modelo h) Se han independizado las tres capas de implementación (contenido, aspecto y comportamiento), en aplicaciones web

Semana 5. Manipulación del DOM y eventos

En este tema en cómo seleccionar elementos y manipularlos.

Sesión 1

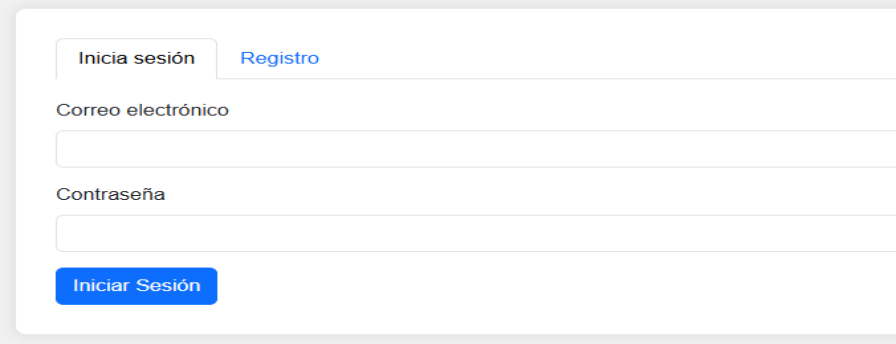
Ejercicio propuesto: registro y autenticación

Ejercicio propuesto: registro y autenticación

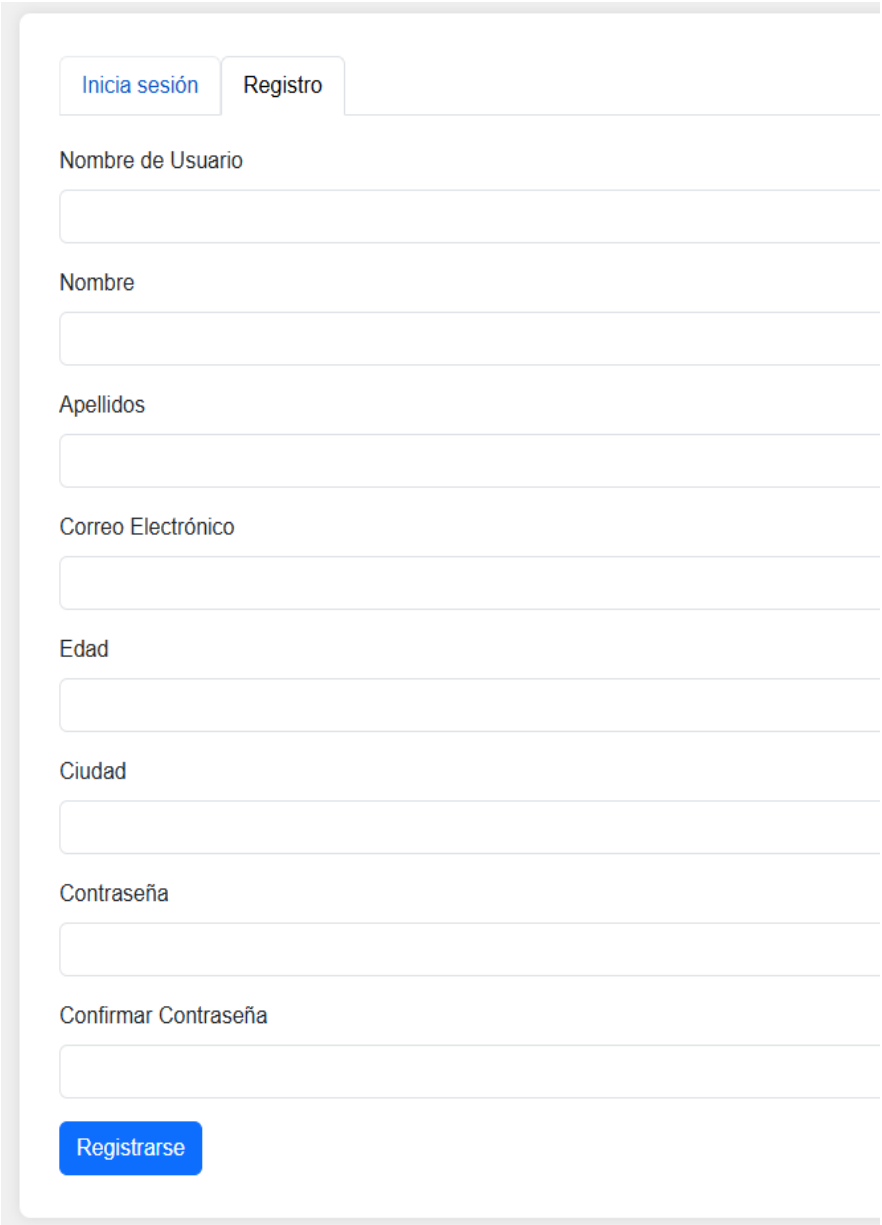
Debes crear una aplicación web que permita a los usuarios registrarse e iniciar sesión en el lado del cliente, utilizando HTML, CSS (Tailwind o Bootstrap) y JavaScript. Validando usuario y contraseña con expresiones regulares y limitaciones en longitud.
La interfaz constará de 2 tabs Registro / Inicia de sesión. Puedes usar una barra de menú si quieres.
Al pulsar uno u otros se actualizará la <section> correspondiente: registro o inicio de sesión.
El campo usuario será un correo. Los datos del inicio de sesión los almacenamos en sessionStore().
Para el Registro usa los campos que se ven en la captura de imagen de abajo.

En este ejercicio solo crea la interface (html y css).

Captura de Inicio de sesión.



Captura de Registro.



Sesión 2

Ejercicio propuesto: continuación del registro y autenticación

Ejemplo guiado: continuación del registro y autenticación

Implementaremos las acciones con Javascript:
- Al pulsar uno u otro tab se actualizará la <section> correspondiente: registro o inicio de sesión.
- El campo usuario será un correo. Limitarlo en longitud y validarlo con expresiones regulares.
- El campo contraseña contendrá como mínimo:al menos una letra mayúscula, un número, un símbolo y una longitud mínima de 8 caracteres. Validarlo con expresiones regulares.
- Al pulsar el botón "Registrarse" los datos se almacenamos en localStore().
- Para el Registro usa los campos que se ven en la captura de imagen de abajo.
- En el Registro limita la longitud de los campos, valida el correo con expresiones regulares. Y si el campo es texto o número según proceda.

Pasos:

1.- Inicia el .js con:

```
document.addEventListener("DOMContentLoaded", function () {
....
}
```

Esto escucha el evento DOMContentLoaded (ocurre cuando ha terminado de cargar el DOM) de document. Cuando ocurra ejecutará nuestro código.

2.- Accede a los elemento: resgiterForm y loginForm, por su Id. Para escuchar el evento "submit". Por ejemplo:

```
document.getElementById("registerForm")?.addEventListener("submit", (e) => {
  if (!validateRegister()) {
    e.preventDefault();
  } else {
    alert("Registro correcto.");
  }
});
```

Nota el ? en registerForm")?.addEventListener("submit" el efecto es:

Si existe registerForm se llama a addEventListener, si no existe, no hace nada y no lanza error.

3.- Crear lo métodos validateRegiter() y validateLogin() que se llaman en el "submit" que le corresponde.

4.- Persiste los datos de registro en localStorage y el login en sessionStorage.

Sesión 3

Ejercicio propuesto: calculadora genérica

Ejemplo guiado: clculadora genérica

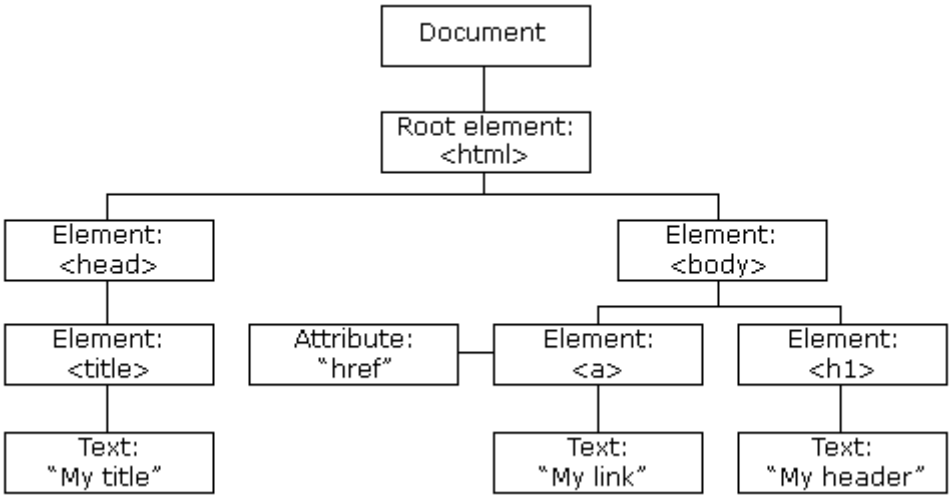
Crear un calculadora como la que vimos en el ejercicio pasado usando HTML, CSS y JS.

Se adjunta código HTML, CSS y JS.

ANEXOS

Modelo de Objetos de Documento (DOM)

El árbol de objetos HTML DOM



El Modelo de Objetos de Documento (DOM) del W3C es una plataforma y una interfaz neutral en cuanto al idioma que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento. En definitiva, HTML DOM es un estándar sobre cómo obtener, cambiar, agregar o eliminar elementos HTML. O dicho de una manera más simplificada: es una estructura jerárquica que representa todos los elementos HTML como objetos en JavaScript. Nos permite:

- Acceder a elementos
- Cambiar su contenido o estilos
- Crear o eliminar nodos
- Escuchar y responder a eventos

El estándar W3C DOM se divide en 3 partes diferentes:

Core DOM: modelo estándar para todo tipo de documentos
XML DOM: modelo estándar para documentos XML
HTML DOM: modelo estándar para documentos HTML

La interfaz de programación son las propiedades y métodos de cada objeto.

- Una **propiedad** es un valor que puede obtener o establecer (como cambiar el contenido de un elemento HTML).
- Un **método** es una acción que puedes realizar (como agregar o eliminar un elemento HTML).

El siguiente ejemplo cambia el contenido (el innerHTML) del <p>elemento con id="demo":

```
<html>
  <body>
    <p id="demo"></p>
      <script>
        document.getElementById("demo").innerHTML = "Hello World!";
      </script>
  </body>
</html>
```

En el ejemplo anterior, getElementByIdes un **método** , mientras que innerHTML es una **propiedad**.

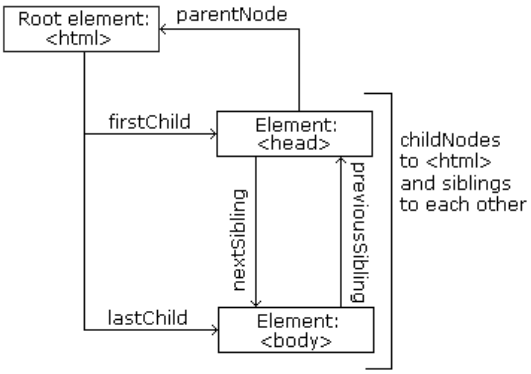
Volvamos a la navegabilidad. Tenemos el documento:

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



En el HTML anterior puedes leer:

<html> es el nodo raíz
<html> no tiene padres
<html> es el padre de <head> y <body>
<head> es el primer hijo de <html>
<body> es el último hijo de <html>

y:

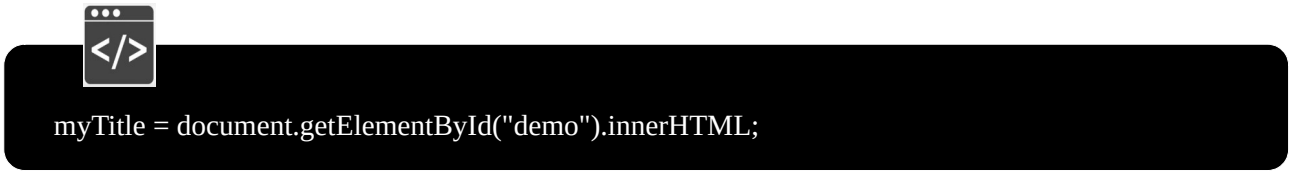
<head> tiene un hijo: <title>
<title> tiene un hijo (un nodo de texto): "DOM Tutorial"
<body> tiene dos hijos: <h1> y <p>
<h1> tiene un hijo: "DOM Lesson one"
<p> tiene un hijo: "¡Hello world!"
<h1> y <p> son hermanos

Puede utilizar las siguientes propiedades de nodo para navegar entre nodos con JavaScript:

- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling
- previousSibling

Por ejemplo el título puede ser accedido de esta manera:

o de esta;



o esta:

Formas de acceder al DOM

Por id, tag o clase

Las formas de acceder a un elemento son:

- document.**getElementById(id)** → Busca elemento por id
- document.getElementsByTagName(name) → Busca elemento por tag
- document.getElementsByClassName(name) → Busca elemento por clase

Ejemplo: dado el párrafo:

```
// index.html

<p id="main_section" class="main_paragraph">Hola</p>
```

```
// script.js

    document.getElementById(main_section)
    document.getElementsByTagName(p)
    document.getElementsByClassName(main_paragraph)
```

Por querySelector y querySelectorAll

querySelector() encuentra la primera coincidencia

```
const titulo = document.querySelector("h1");  // Encuentra el primer h1 y lo almacena en titulo
```

querySelectorAll() encuentra toda las coincidencia en formato array.

```
// Encuentra todas las clases boton y lo almacena en botone
const botones = document.querySelectorAll(".boton");
```

Otras formas de acceder

Modificar HTML, atributo y estilo de un elemento

Propiedades

element.innerHTML = nuevo valor → añadir código HTML.
element.attribute = nuevo valor → cambia el valor del atributo
element.style.property = nuevo estilo → cambia el estilo
element.textContent = nuevo valor → cambia el contenido

Método	
element.setAttribute(attribute, value)	Cambia el valor del atributo de un elemento HTML

Crear, borrar, agregar, reemplazar y escribir elementos

document.createElement(element) → **Crea** un elemento en un documento HTML
document.appendChild(element) → **Agrega** un elemento en un documento HTML

Un ejemplo práctico: añadir texto a un párrafo.

```
const p = document.createElement("p");
p.textContent = "Este es un nuevo párrafo";
document.body.appendChild(p);
```

document.removeChild(element) → **Borra** un elemento en un documento HTML
document.replaceChild(new, old) → **Reemplaza** un elemento en un doc. HTML
document.write(text) → **Escribir** en el flujo de salida HTML

Lectura del valor de un input

Accedemos a los valores introducidos por el usuario

```
const nombre = document.querySelector("#nombre").value;
```

Eventos

Un evento es cualquier suceso que le ocurra a un elemento HtML Javascript puede darse cuenta de ese evento y reaccionar a este ejecutando el código programado.

Algunos eventos que pueden ocurrir podrían ser:

- Pulsar un botón
- Modificar un campo de texto
- Pulsar una tecla
- La página ha terminado de cargarse
- Pasar sobre un elemento HTML

La forma de programar en el elemento HTML el evento sería la siguiente:

```
<elemento_HTML evento=’código_Javascript’>
```

También podemos crear el evento desde código javascript con el método addEventListener():

```
elemento.addEventListener("evento", () => {
  alert("Has hecho clic");
});
```

elemento es cualquiera del DOM.

Eventos comunes:

- click
- input
- submit
- mouseover, mouseout
- keydown, keyup

Para conocer todos los eventos visita w3School.

Gestión de la clases CSS

Crear clase

Añadir una clase CSS.

```
let elemento = document.getElementById(“idAlumno”);
elemento.classList.add("activo");
```

Borrar clase

Borramos una clase de la memoria del navegador, pero no del fichero.

```
let elemento = document.getElementById(“idAlumno”);
elemento.classList.remove("activo");
```

Ocultar clase

Para alternar a otro clase del CSS. Útil para modificar estilo, por ejemplo al para el ratón sobre un elemento.

```
let elemento = document.getElementById(“idAlumno”);
elemento.classList.toggle("oculto");
```

División de código en varios ficheros

¿cómo puedo llamar desde main.js a métodos de otro fichero metodos.js? Hay varias formas según el entorno de trabajo:

1) En navegador con ES Modules

index.html

```
<script type="module" src="./main.js"></script>
```

metodos.js

```
// exportaciones con nombre
export function sumar(a, b) { return a + b; }
export const PI = 3.1416;

// exportación por defecto (opcional)
export default function saludar(nombre) { return `Hola ${nombre}`; }
```

main.js

```
// importar con nombre
import { sumar, PI } from "./metodos.js";

// importar la exportación por defecto
import saludar from "./metodos.js";

console.log(sumar(2, 3), PI);
console.log(saludar("David"));
```

Notas:

- type="module" es obligatorio. Carga en modo estricto y con ámbito propio.
- Rutas relativas con extensión .js.
- Servir por HTTP evita problemas de CORS.

2) En Node.js (CommonJS)

metodos.js

```
function sumar(a, b) { return a + b; }
const PI = 3.1416;

module.exports = { sumar, PI };
```

main.js

```
const { sumar, PI } = require("./metodos.js");
console.log(sumar(2, 3), PI);
```

Notas:

- Válido si tu package.json **NO** tiene "type": "module".

3) En Node.js (ESM)

package.json

```
{ "type": "module" }
```

metodos.js

```
export function sumar(a, b) { return a + b; }
export const PI = 3.1416;
```

main.js

```
import { sumar, PI } from "./metodos.js";
console.log(sumar(2, 3), PI);
```

Diferencias ESM vs CommonJS

- ESM: import/export, evaluación diferida (los módulos se **analizan primero** y se **ejecutan después**), enlaces vivos (i el módulo exportador cambia el valor, los importadores ven el cambio).
- CJS: require/module.exports, carga dinámica en tiempo de ejecución, valores copiados.

Cuándo usar ESM:

- Proyectos modernos en navegador o Node.
- Cuando necesitas tree-shaking, tooling moderno y código isomórfico.

Cuándo evitarlo:

- Proyectos legacy que dependen de require() sin bundler.