| | UNIVERSIDAD AUTONOMA "TOMAS FRIAS" | NOTA: |
|---|---|---|
| | FACULTAD DE VICERRECTORADO | |
| | CARRERA INGENIERIA DE SISTEMAS | |

| | |
|---|---|
| ESTUDIANTE: Moises Villegas Vidaurre | PRACTICA No 2 |
| DOCENTE: Ing. Javier Choque Matos | MATERIA: SIS - 414 |
| AUXILIAR: Univ. Kevin Alexis Rodríguez Condori | GRUPO: 2 |

# Parte Teórica

## 1. Preguntas Conceptuales

### a) ¿Qué es el DOM y cómo se relaciona con HTML?

El **DOM** (Documento Object Model) es una interfaz de programación que representa la estructura de documentos HTML y XML como un árbol de nodos. Cada nodo del árbol corresponde a una parte del documento, como elementos, atributos y texto.

### Relación entre DOM y HTML:

1. **Estructura**: El DOM permite que los lenguajes de programación, como JavaScript, interactúen con el contenido y la estructura de un documento HTML. Cada etiqueta HTML se convierte en un nodo en el árbol del DOM.
2. **Manipulación**: A través del DOM, se pueden agregar, eliminar o modificar elementos y atributos de un documento HTML de manera dinámica. Esto permite crear interfaces interactivas y responder a eventos del usuario.
3. **Acceso**: Los desarrolladores pueden acceder a los nodos del DOM utilizando métodos y propiedades proporcionados por JavaScript, lo que facilita la manipulación del contenido de la página web.

**b)** Explica la diferencia entre:

- `document. getElementById ()` vs `document. querySelector ()`.
- `textContent` vs `innerHTML`.

**1. `document. getElementById ()` VS `document. querySelector ()`**

- **`document. getElementById ()`:**
  - **Uso**: Se utiliza para seleccionar un único elemento del DOM basado en su atributo `id`.
  - **Sintaxis**: `document. getElementById('miId')`
  - **Retorno**: Devuelve el primer elemento que coincide con el `id` especificado o `null` si no hay coincidencias.
  - **Rendimiento**: Generalmente más rápido, ya que busca directamente por `id`.
- **`document. querySelector ()`:**

- **Uso**: Permite seleccionar el primer elemento que coincide con un selector CSS proporcionado (puede ser un `id`, clase, etiqueta, etc.).
- **Sintaxis**: `document. querySelector ('. mi Clase')` o `document. querySelector('div')`
- **Retorno**: Devuelve el primer elemento que coincide con el selector o `null` si no hay coincidencias.
- **Flexibilidad**: Más versátil, ya que se pueden utilizar selectores CSS más complejos.

### 2. `textContent` VS `innerHTML`

- **`textContent`**:
  - **Uso**: Se utiliza para obtener o establecer el contenido textual de un nodo. Ignora cualquier etiqueta HTML.
  - **Ejemplo**: `element. textContent` devuelve solo el texto, sin HTML.
  - **Seguridad**: Más seguro para evitar la inyección de HTML, ya que solo trata el texto.
- **`innerHTML`**:
  - **Uso**: Se utiliza para obtener o establecer el contenido HTML de un nodo, lo que incluye etiquetas y contenido.
  - **Ejemplo**: `element.innerHTML` devuelve el contenido HTML completo, permitiendo el uso de etiquetas.
  - **Funcionalidad**: Permite insertar HTML dinámicamente, pero puede ser menos seguro si se inserta contenido no confiable, ya que puede introducir vulnerabilidades como XSS (Cross-Site Scripting).

## c) ¿Para qué sirve `addEventListener`? Proporciona un ejemplo.

El método **`addEventListener`** se utiliza para registrar un evento en un elemento del DOM, permitiendo que se ejecute una función específica cuando ocurre ese evento. Es una forma de hacer que los elementos respondan a interacciones del usuario, como clics, movimientos del mouse, pulsaciones de teclas, entre otros.

### Ventajas de **`addEventListener`**:

- Permite agregar múltiples manejadores para el mismo evento en un mismo elemento.
- Facilita el control sobre el comportamiento de los eventos, como la opción de detener la propagación del evento.

### Ejemplo:

Aquí tienes un ejemplo simple donde se utiliza `addEventListener` para manejar un clic en un botón:

html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Ejemplo de addEventListener</title>
</head>
<body>
    <button id="miBoton">¡Haz clic en mí!</button>
    <p id="mensaje"></p>

    <script>

        const boton =
document.getElementById('miBoton');
        const mensaje = document.
getElementById('mensaje');


        boton.addEventListener('click', function() {
            mensaje.textContent = '¡Gracias por
hacer clic!';
        });
    </script>
</body>
</html>
```

html

Abrir en lienzo

**¡Haz clic en mí! - Pesquisa Google**

**d) ¿Qué métodos del DOM se usan para capturar valores de un formulario?**

Para capturar valores de un formulario en el DOM, se utilizan varios métodos y propiedades.

### 1. `document.getElementById()`

- Se utiliza para obtener un elemento específico del formulario mediante su atributo `id`.
- Ejemplo:

  JavaScript

  Copiar

```
const valor = document.getElementById('miCampo').value;
```

### 2. `document.querySelector()`

- Permite seleccionar el primer elemento que coincide con un selector CSS, lo que incluye campos de entrada.
- Ejemplo:

  JavaScript

  Copiar

```
const valor = document.querySelector('input[name="miCampo"]').value;
```

### 3. `elements`

- Se puede acceder a los elementos de un formulario directamente a través de la propiedad `elements` del formulario.
- Ejemplo:

  JavaScript

```
const formulario =
document.getElementById('miFormulario');
const valor =
formulario.elements['miCampo'].value;
```

**4. `value`**

- Esta propiedad se utiliza para obtener o establecer el valor de un campo de entrada.
- Ejemplo:

  JavaScript

```
const valor =
document.getElementById('miCampo').value;
```

**Ejemplo**

Html

```
<form id="miFormulario">
    <input type="text" id="miCampo" name="miCampo"
placeholder="Escribe algo aquí">
    <button type="submit">Enviar</button>
</form>

<script>
    const formulario =
document.getElementById('miFormulario');
    formulario.addEventListener('submit',
function(event) {
        event.preventDefault(); // Evita el envío
del formulario
        const valor =
formulario.elements['miCampo'].value;
        console.log('Valor capturado:', valor);
    });
</script>
```

## e) Explica cómo prevenir el envío por defecto de un formulario con JavaScript.

Para prevenir el envío por defecto de un formulario en JavaScript, se utiliza el método preventDefault() del objeto del evento. Esto evita que el formulario se envíe y la página se recargue, permitiendo manejar el envío de manera personalizada.

**Pasos para prevenir el envío por defecto de un formulario:**

1. **Agregar un manejador de eventos**: Usa addEventListener para escuchar el evento submit del formulario.
2. **Llamar a preventDefault()**: Dentro de la función del manejador, llama a event.preventDefault() para evitar el comportamiento por defecto.

**Ejemplo:**

html

```
<form id="miFormulario">
  <input type="text" id="miCampo" name="miCampo"
placeholder="Escribe algo aquí">
  <button type="submit">Enviar</button>
</form>

<script>
  const formulario = document.getElementById('miFormulario');

  formulario.addEventListener('submit', function(event) {
    event.preventDefault(); // Previene el envío del formulario
    const valor = document.getElementById('miCampo').value;
    console.log('Valor capturado:', valor);

  });
</script>
```

**Descripción del ejemplo:**

- Se selecciona el formulario y se agrega un evento submit.

- Al enviarlo, se llama a preventDefault(), lo que evita que la página se recargue.
- Se puede procesar el valor del campo de entrada sin enviar el formulario.

## f) ¿Qué es el "almacenamiento en memoria" y en qué se diferencia de `localStorage`?

El "almacenamiento en memoria" y localStorage son dos formas de manejar datos en aplicaciones web, pero tienen características y propósitos diferentes.

### Almacenamiento en Memoria

- **Definición**: Se refiere a la capacidad de almacenar datos en variables dentro de la memoria del navegador mientras la página está abierta. Estos datos se pierden al cerrar o actualizar la página.
- **Uso**: Ideal para almacenar información temporal que no necesita persistir entre sesiones, como el estado de una interfaz de usuario o datos de formularios que se están completando.
- **Acceso**: Los datos se pueden acceder directamente a través de variables en el código JavaScript.

### localStorage

- **Definición**: Es una parte de la Web Storage API que permite almacenar datos en el navegador de manera persistente. Los datos guardados en localStorage permanecen disponibles incluso después de cerrar el navegador o actualizar la página.
- **Uso**: Útil para almacenar configuraciones de usuario, preferencias, o cualquier dato que deba persistir entre sesiones.
- **Acceso**: Se accede a localStorage mediante métodos como setItem(), getItem(), y removeItem(). Los datos se almacenan como pares clave-valor.

### Diferencias Clave

| Característica | Almacenamiento en Memoria | localStorage |
|---|---|---|

| | | |
|---|---|---|
| Persistencia | No persiste al cerrar la página | Persiste entre sesiones |
| **Almacenamiento** | Solo en variables en memoria | Almacenamiento clave-valor |
| **Capacidad** | Limitada a la memoria del navegador | Aproximadamente 5-10 MB por dominio |
| **Métodos de acceso** | Variables directamente | setItem(), getItem(), removeItem() |

**Ejemplo de uso de localStorage:**

JavaScript

```
localStorage.setItem('nombre', 'Juan');

const nombre = localStorage.getItem('nombre');
console.log(nombre);

localStorage.removeItem('nombre');
```

### 2. Análisis de Código

Dado el siguiente código:

```
<button id="btn">Haz clic</button>
<p id="mensaje"></p>
document.getElementById("btn").addEventListener("click", () => {
  document.getElementById("mensaje").textContent = "¡Botón presionado!";
});
```

**Preguntas:**

- ¿Qué hace el código?
- ¿Qué pasaría si cambiamos `textContent` por `innerHTML`?

## Análisis del Código

1. **¿Qué hace el código?**
   - El código agrega un evento de clic a un botón con el ID btn. Cuando el botón es presionado, se cambia el contenido de un párrafo con el ID mensaje a "¡Botón presionado!".
   - En resumen, al hacer clic en el botón, se muestra un mensaje en el párrafo.
2. **¿Qué pasaría si cambiamos textContent por innerHTML?**
   - Si cambias textContent por innerHTML, el código todavía funcionará de manera similar, pero con una diferencia clave:
     - **textContent**: Solo establece el contenido textual del párrafo, ignorando cualquier etiqueta HTML. En este caso, el texto "¡Botón presionado!" se mostrará tal cual.
     - **innerHTML**: Permite insertar contenido HTML. Si se usara innerHTML, podrías agregar etiquetas HTML al contenido. Por ejemplo:

       JavaScript

       ```
       document.getElementById("mensaje").innerHTML = "<strong>¡Botón presionado!</strong>";
       ```

       Esto haría que el texto se muestre en negrita.

## Resumen

- El código muestra un mensaje en un párrafo al hacer clic en un botón.
- Cambiar a innerHTML permitiría insertar HTML, lo que puede ser útil, pero también presenta riesgos de seguridad si se insertan datos no confiables (p. ej., inyección de HTML).

## 3. Análisis de Código

Dado el siguiente formulario:

```
<form id="form-usuario">
  <input type="text" id="nombre" placeholder="Nombre">
  <button type="submit">Guardar</button>
</form>
<ul id="lista-usuarios"></ul>
const usuarios = [];
document. getElementById("form-usuario").
addEventListener ("submit", (e) => {
  e. preventDefault ();
  const nombre = document.getElementById("nombre").value;
  usuarios.push(nombre);
  actualizarListaUsuarios();
});

function actualizarListaUsuarios() {
  const lista = document.getElementById("lista-usuarios");
  lista.innerHTML = usuarios.map(user =>
`<li>${user}</li>`).join("");
}
```

**Preguntas:**

- ¿Qué hace el código al enviar el formulario?
- ¿Cómo se simula la "persistencia de datos" aquí?

**Análisis del Código**

1. **¿Qué hace el código al enviar el formulario?**
    - Cuando se envía el formulario, se ejecuta el manejador de eventos asociado al evento submit.
    - **Pasos que se llevan a cabo:**
        1. **e.preventDefault();**: Esta línea evita que el formulario se envíe de la manera predeterminada, lo que normalmente recargaría la página.

2. Se obtiene el valor del campo de texto con id="nombre".
3. El nombre se agrega al array usuarios, que simula un almacenamiento temporal en memoria.
4. Se llama a la función actualizarListaUsuarios(), que actualiza la lista en la interfaz mostrando todos los nombres almacenados.

2. **¿Cómo se simula la "persistencia de datos" aquí?**
    - La "persistencia de datos" en este caso se simula mediante el uso de un array (usuarios) que almacena los nombres ingresados en la sesión actual. Aunque este almacenamiento es volátil y se pierde al recargar la página, simula la idea de mantener datos durante la vida de la aplicación.
    - Al agregar nombres al array y mostrarlos en la lista, se da la impresión de que los datos se están guardando y persistiendo, aunque en realidad solo se mantienen en memoria mientras la página no se recargue.

**Resumen**

- Al enviar el formulario, se evita el comportamiento predeterminado, se almacena el nombre en un array y se actualiza la lista de usuarios en la interfaz.
- La persistencia de datos se simula usando un array en memoria, lo que permite mantener los datos durante la sesión actual del navegador.