Actividades: Javascript + HTML

Ejercicio 1: Modificar el Contenido de un Elemento

- **Consigna:**
  Crea una página HTML con un `<div>` que tenga el id `"mensaje"`. Mediante JavaScript, modifica el contenido de dicho `<div>` a "¡Bienvenido a la práctica de Asincronismo y Eventos!".

- **Resultado esperado:**
  Al cargar la página, el `<div>` mostrará el texto indicado.

- **Conceptos tratados:**
  - Acceso al DOM
  - Uso de `document.getElementById` y propiedad `innerHTML`

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 1</title>
</head>
<body>
  <div id="mensaje">Contenido original</div>
  <script>
    document.getElementById("mensaje").innerHTML =
"¡Bienvenido a la práctica de Asincronismo y Eventos!";
  </script>
</body>
</html>
```

Ejercicio 2: Agregar un Elemento a una Lista con Evento de Click

- **Consigna:**
  En una página HTML, crea una lista no ordenada (`<ul>`) con algunos elementos `<li>` y un botón. Al pulsar el botón, añade un nuevo elemento `<li>` con el texto "Ítem agregado".

- **Resultado esperado:**
  Al hacer clic en el botón, se agregará un nuevo ítem al final de la lista.

- **Conceptos tratados:**
  - Manipulación de DOM (crear elementos y agregarlos)
  - Manejo de eventos con el método `onclick` o `addEventListener`

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 2</title>
</head>
<body>
  <ul id="lista">
    <li>Ítem 1</li>
    <li>Ítem 2</li>
  </ul>
  <button id="agregarBtn">Agregar Ítem</button>

  <script>

document.getElementById("agregarBtn").addEventListener("click"
, function() {
      const nuevoItem = document.createElement("li");
      nuevoItem.textContent = "Ítem agregado";
```

```
      document.getElementById("lista").appendChild(nuevoItem);
    });
  </script>
</body>
</html>
```

Ejercicio 3: Cambiar la Fuente de una Imagen con un Evento

- **Consigna:**
  Inserta en la página una imagen con `src="img1.jpg"`. Crea un botón que, al ser pulsado, cambie el atributo `src` de la imagen a `"img2.jpg"`.

- **Resultado esperado:**
  Al hacer clic en el botón, la imagen mostrada se actualizará a la nueva ruta.

- **Conceptos tratados:**
  - Acceso y manipulación de atributos del DOM
  - Uso de `setAttribute` junto con eventos

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 3</title>
</head>
<body>
  <img id="foto" src="img1.jpg" alt="Imagen inicial">
  <button id="cambiarFoto">Cambiar Imagen</button>

  <script>
```

```
document.getElementById("cambiarFoto").addEventListener("click
", function() {
      document.getElementById("foto").setAttribute("src",
"img2.jpg");
    });
  </script>
</body>
</html>
```

Ejercicio 4: Manejar un Evento con addEventListener

- **Consigna:**
  Crea un `<div>` con un fondo de color y texto "Haz Clic Aquí". Usa `addEventListener` para que, al hacer clic, se imprima en la consola "Div activado".

- **Resultado esperado:**
  Al hacer clic en el `<div>`, se visualizará el mensaje en la consola del navegador.

- **Conceptos tratados:**
  - Manejo de eventos con `addEventListener`
  - Uso de la consola para depuración

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 4</title>
  <style>
    #divClickeable {
      width: 200px;
```

```
      height: 100px;
      background-color: lightcoral;
      line-height: 100px;
      text-align: center;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <div id="divClickeable">Haz Clic Aquí</div>
  <script>
    const divElem = document.getElementById("divClickeable");
    divElem.addEventListener("click", function() {
      console.log("Div activado");
    });
  </script>
</body>
</html>
```

Ejercicio 5: Actualizar Contenido en Tiempo Real con Eventos de Entrada

- **Consigna:**
  Crea un `<input type="text">` y un `<p>` vacío. Cada vez que el usuario escriba en el campo, el párrafo se actualizará mostrando lo que se escribe en tiempo real.

- **Resultado esperado:**
  El `<p>` mostrará inmediatamente el contenido ingresado en el `<input>`.

- **Conceptos tratados:**
  - Uso del evento `input`
  - Actualización dinámica del DOM

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 5</title>
</head>
<body>
  <input type="text" id="campoEntrada" placeholder="Escribe
algo...">
  <p id="salidaTexto"></p>

  <script>
    const campo = document.getElementById("campoEntrada");
    const salida = document.getElementById("salidaTexto");

    campo.addEventListener("input", function() {
      salida.textContent = campo.value;
    });
  </script>
</body>
</html>
```

Ejercicio 6: Consumo de una API con fetch y Promesas para Mostrar Datos en una Tabla

- **Consigna:**
  Utiliza la API de JSONPlaceholder para obtener una lista de publicaciones (posts). Muestra los primeros 10 posts en una tabla HTML con las columnas ID, Título y Contenido.
  Usa Bootstrap para estilizar la tabla.

- **Resultado esperado:**
  Al cargar la página, se realizará la consulta asíncrona y se mostrará una tabla con los datos obtenidos de la API.

- **Conceptos tratados:**
    - Uso de `fetch` y manejo de promesas
    - Manipulación del DOM para generar una tabla
    - Integración de Bootstrap para la presentación visual

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 6</title>
  <!-- Bootstrap CSS CDN -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css">
</head>
<body>
  <div class="container mt-4">
    <h1 class="mb-4">Publicaciones</h1>
    <table class="table table-bordered table-striped">
      <thead class="thead-dark">
        <tr>
          <th>ID</th>
          <th>Título</th>
          <th>Contenido</th>
        </tr>
      </thead>
      <tbody id="tablaPosts">
        <!-- Aquí se insertarán los posts -->
      </tbody>
    </table>
  </div>
```

```
  <script>
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(response => response.json())
      .then(data => {
        const tabla = document.getElementById("tablaPosts");
        // Tomamos los 10 primeros posts
        data.slice(0, 10).forEach(post => {
          const fila = document.createElement("tr");
          fila.innerHTML = `<td>${post.id}</td>
                            <td>${post.title}</td>
                            <td>${post.body}</td>`;
          tabla.appendChild(fila);
        });
      })
      .catch(error => console.error("Error:", error));
  </script>
</body>
</html>
```

Ejercicio 7: Consumo de API con Async/Await y Refresco de Datos

- **Consigna:**
  Crea un botón "Refrescar Datos" que, al hacer clic, haga una consulta asíncrona a la API de JSONPlaceholder (usando async/await) para obtener los posts y actualice la tabla de datos creada en el ejercicio 6.

- **Resultado esperado:**
  Al pulsar el botón, la tabla se actualiza con los datos actuales de la API (por ejemplo, los primeros 10 posts). Se deberá manejar errores de forma apropiada.

- **Conceptos tratados:**
  - Uso de async/await para manejo asíncrono
  - Refresco y actualización dinámica de datos en el DOM

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 7</title>
  <!-- Bootstrap CSS CDN -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css">
</head>
<body>
  <div class="container mt-4">
    <h1 class="mb-4">Publicaciones (Refrescables)</h1>
    <button id="refrescarBtn" class="btn btn-primary
mb-3">Refrescar Datos</button>
    <table class="table table-bordered table-striped">
      <thead class="thead-dark">
        <tr>
          <th>ID</th>
          <th>Título</th>
          <th>Contenido</th>
        </tr>
      </thead>
      <tbody id="tablaPosts">
        <!-- Se actualizarán los posts -->
      </tbody>
    </table>
  </div>

  <script>
    async function cargarPosts() {
      try {
        const respuesta = await
fetch("https://jsonplaceholder.typicode.com/posts");
        const datos = await respuesta.json();
```

```
        const tabla = document.getElementById("tablaPosts");
        tabla.innerHTML = ""; // Vaciar tabla antes de
actualizar
        datos.slice(0, 10).forEach(post => {
          const fila = document.createElement("tr");
          fila.innerHTML = `<td>${post.id}</td>
                            <td>${post.title}</td>
                            <td>${post.body}</td>`;
          tabla.appendChild(fila);
        });
      } catch (error) {
        console.error("Error al cargar datos:", error);
      }
    }


document.getElementById("refrescarBtn").addEventListener("clic
k", cargarPosts);

    // Cargar los datos cuando se carga la página
    cargarPosts();
  </script>
</body>
</html>
```

Ejercicio 8: Actualizar Vista con Evento y Datos Asíncronos

- **Consigna:**
  Crea una página HTML que contenga dos secciones: una con un botón "Mostrar Posts" y otra vacía que actuará como contenedor para mostrar los posts obtenidos desde JSONPlaceholder mediante fetch. Al pulsar el botón se hará la consulta asíncrona y se desplegarán los resultados en la sección correspondiente.

- **Resultado esperado:**
  Al pulsar el botón "Mostrar Posts", se realiza la consulta a la API y se renderiza una lista de posts (por ejemplo, solo títulos) en el contenedor.

- **Conceptos tratados:**
  - Eventos y asincronismo con fetch
  - Manipulación condicional del DOM para cambiar vistas

Solución propuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 8</title>
  <!-- Bootstrap CSS CDN -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css">
</head>
<body>
  <div class="container mt-4">
    <button id="mostrarPosts" class="btn btn-success
mb-3">Mostrar Posts</button>
    <div id="contenedorPosts" class="list-group">
      <!-- Se mostrarán los posts aquí -->
    </div>
  </div>
```

```
  <script>

document.getElementById("mostrarPosts").addEventListener("clic
k", async function() {
      try {
        const res = await
fetch("https://jsonplaceholder.typicode.com/posts");
        const posts = await res.json();
        const contenedor =
document.getElementById("contenedorPosts");
        contenedor.innerHTML = "";  // Limpiar el contenedor
        posts.slice(0, 10).forEach(post => {
          const item = document.createElement("a");
          item.href = "#";
          item.className = "list-group-item
list-group-item-action";
          item.textContent = post.title;
          contenedor.appendChild(item);
        });
      } catch (error) {
        console.error("Error al obtener posts:", error);
      }
    });
  </script>
</body>
</html>
```

Ejercicio 9 (Desafío): SPA Simple sin Hash Routing – Cambio de Vistas con Eventos

- **Consigna:**
  Crea una aplicación de página única (SPA) que contenga dos secciones principales (por ejemplo, "Inicio" y "Acerca de"). La navegación entre secciones se realizará mediante botones (o enlaces) que, al pulsarlos, cambian dinámicamente el contenido principal sin recargar la página. La consulta de datos de la API (por ejemplo, mostrar una lista de posts en la vista "Inicio") debe hacerse de forma asíncrona al cambiar a esa vista.

- **Resultado esperado:**
  Al pulsar los botones de navegación se intercambia el contenido en la misma página. Al ingresar a la vista "Inicio", se muestra el resultado de una consulta a JSONPlaceholder; al cambiar a "Acerca de", se muestra contenido estático.

- **Conceptos tratados:**
  - SPA (single page application) sin cambiar la URL
  - Eventos para cambiar vistas
  - Consumo de datos asíncronos con fetch

- **Solución:**
  Este ejercicio es un desafío. Se sugiere:
  - Crear botones para la navegación.
  - Usar eventos de click para ocultar/mostrar secciones.
  - En la vista "Inicio", realizar una consulta asíncrona similar a los ejercicios anteriores para mostrar posts.

Ejercicio 10 (Desafío): Gestor de Tareas Asíncrono en una SPA

- **Consigna:**
  Desarrolla una aplicación de "Lista de Tareas" (Todo List) como SPA. La aplicación debe permitir:
    - Agregar nuevas tareas a través de un formulario.
    - Marcar las tareas como completadas (cambiando su estilo).
    - Eliminar tareas.
    - (Opcional) Simular la persistencia de datos usando una API externa o almacenamiento local de forma asíncrona (por ejemplo, mediante fetch con un endpoint simulado o utilizando localStorage en conjunto con promesas).
- **Resultado esperado:**
  La aplicación muestra una lista interactiva de tareas en la misma página. Cada acción (agregar, completar, eliminar) se realiza sin recargar la página, y se incorpora asincronismo en la simulación de persistencia de datos.
- **Conceptos tratados:**
    - SPA y manejo de estado
    - Consumo asíncrono de datos (simulado o real)
    - Gestión de eventos en formularios y elementos dinámicos
    - Organización modular del código
- **Solución:**
  Este ejercicio es un desafío para que integres los conocimientos aprendidos. Se sugiere:
    - Crear una estructura HTML con un formulario y una lista.
    - Usar eventos para manejar la adición y eliminación de tareas.

- Incorporar async/await para simular la persistencia (por ejemplo, con `setTimeout` o utilizando localStorage).