



UNIVERSIDAD DE  
DISEÑO, INNOVACIÓN  
Y TECNOLOGÍA

Lenguaje de Marcas

**PE1 – (Parte B) To-Do App - Guia del  
alumno**

*Primer Curso Grado de Formación Profesional*

Profesor: Luis Rubio

# Índice

Descripción de la práctica.....	3
Panorama general.....	3
Fase 0 – Preparación del proyecto (≈ 10 min).....	4
Estructura mínima en index.html.....	4
Checklist Fase 0.....	5
Fase 1 – Estilos básicos y estructura visual (≈ 20 min).....	6
Checklist Fase 1.....	9
Fase 2 – Lógica base en JavaScript (≈ 25 min).....	9
Checklist Fase 2.....	11
Fase 3 – Añadir tareas nuevas (≈ 35 min).....	11
3.1 Manejar el envío del formulario.....	11
Checklist Fase 3.....	12
Fase 4 – Eliminar tareas (≈ 30 min).....	13
4.1 Escuchar clics en la lista.....	13
Checklist Fase 4.....	14
Fase 5 – (Opcional) Marcar tareas como completadas (≈ 30 min).....	14
5.1 Marcar/desmarcar desde el mismo listener.....	14
Checklist Fase 5.....	15
Checklist final de la práctica.....	16

# Descripción de la práctica

## Objetivo de la sesión:

- Añadir un formulario con un campo de texto y un botón “**Añadir tarea**”.
- Cada vez que se pulse el botón, crear un nuevo elemento de lista con el texto introducido.
- Incluir un botón “**Eliminar**” por tarea que permita borrarla de la lista.
- (Opcional) Permitir marcar una tarea como completada cambiando su estilo.

Sin guardar nada en localStorage (al recargar la página se pierde todo, y está bien así para esta práctica).

## Panorama general

- **Duración estimada:** 150 minutos (2 h 30 min)
- **Tecnologías:** HTML, CSS, JavaScript (DOM + eventos + arrays).
- **Producto final:** una pequeña aplicación de lista de tareas donde puedes:
  - Añadir tareas nuevas.

- Verlas listadas.
- Eliminar tareas que ya no quieras.
- (Opcional) Marcar tareas como completadas.

## Fase 0 - Preparación del proyecto (≈ 10 min)

Crea una carpeta, por ejemplo:

to-do-app

1. Dentro, crea estos archivos:

- index.html
- styles.css
- script.js

2. Abre la carpeta en VS Code.

### Estructura mínima en index.html

Pon esta estructura base:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
```

```

initial-scale=1.0" />
<title>To-Do App</title>
<link rel="stylesheet" href="styles.css" />
</head>
<body>
  <header>
    <h1>To-Do App</h1>
    <p>Gestiona una lista sencilla de tareas con JavaScript.</p>
  </header>

  <main>
    <section class="contenedor">
      <h2>Mis tareas</h2>

      <!-- Formulario para añadir tareas -->
      <form id="form-tarea">
        <input
          type="text"
          id="input-tarea"
          placeholder="Escribe una nueva tarea..." required
        />
        <button type="submit">Añadir tarea</button>
      </form>

      <!-- Lista donde aparecerán las tareas -->
      <ul id="lista-tareas">
        <!-- Las tareas se generarán dinámicamente -->
      </ul>
    </section>
  </main>

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

```

## Checklist Fase 0

- Carpeta de proyecto creada con `index.html`, `styles.css`, `script.js`.

- index.html enlaza bien styles.css y script.js.
- Al abrir index.html en el navegador ves el título, el formulario y la lista vacía.
- No hay errores en la consola del navegador.

## Fase 1 - Estilos básicos y estructura visual (≈ 20 min)

Objetivo: que la app sea agradable y clara de usar.

En styles.css, añade:

```
* {  
    box-sizing: border-box;  
}  
  
body {  
    margin: 0;  
    font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe  
UI", sans-serif;  
    background-color: #f5f5f5;  
    color: #222;  
}  
  
header {  
    background-color: #ffffff;  
    border-bottom: 1px solid #ddd;  
    padding: 1rem 2rem;  
}  
  
header h1 {  
    margin: 0;  
    font-size: 1.8rem;  
}  
  
header p {  
    margin: 0.3rem 0 0;  
    font-size: 0.95rem;
```

```
    color: #555;
}

main {
    max-width: 600px;
    margin: 2rem auto;
    padding: 0 1rem 2rem;
}

.contenedor {
    background-color: #ffffff;
    border-radius: 6px;
    padding: 1.5rem;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.06);
}

.contenedor h2 {
    margin-top: 0;
}

/* Formulario */
#form-tarea {
    display: flex;
    gap: 0.5rem;
    margin: 1rem 0 1.5rem;
}

#input-tarea {
    flex: 1;
    padding: 0.5rem 0.7rem;
    border-radius: 4px;
    border: 1px solid #ccc;
    font-size: 1rem;
}

#form-tarea button {
    padding: 0.5rem 0.9rem;
    border-radius: 4px;
    border: none;
    background-color: #222;
    color: #f5f5f5;
    font-weight: 600;
```

```
    cursor: pointer;
}

/* Lista de tareas */
#lista-tareas {
    list-style: none;
    padding-left: 0;
    margin: 0;
}

.tarea {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 0.5rem;
    padding: 0.5rem 0.7rem;
    border: 1px solid #ddd;
    border-radius: 4px;
    margin-bottom: 0.4rem;
    background-color: #fafafa;
}

/* Texto de la tarea */
.tarea-texto {
    flex: 1;
}

/* Botones de cada tarea */
.tarea button {
    border: none;
    border-radius: 4px;
    padding: 0.25rem 0.5rem;
    font-size: 0.8rem;
    cursor: pointer;
}

.btn-eliminar {
    background-color: #e53935;
    color: #fff;
}

/* Opcional: estilo para tareas completadas */
```

```
.tarea-completada .tarea-texto {  
    text-decoration: line-through;  
    opacity: 0.6;  
}
```

Puedes ajustar colores y detalles si te apetece, pero mantén las clases e ids que se usan en el JS.

## Checklist Fase 1

- El formulario se ve horizontal, con input y botón al lado.
- La tarjeta “Mis tareas” se ve con un fondo blanco y una ligera sombra.
- La lista de tareas aparece vacía, pero con espacio listo para elementos.
- No hay errores nuevos en consola.

## Fase 2 - Lógica base en JavaScript (≈ 25 min)

Objetivo: preparar las **variables** y la estructura que usaremos para la To-Do App.

En `script.js`, empieza con:

```
// FASE 2: Configuración base de la To-Do App  
  
// 1. Array donde guardaremos las tareas (cada tarea será un objeto)  
const tareas = [];  
  
// 2. Referencias a elementos del DOM  
const formulario = document.querySelector("#form-tarea");  
const inputTarea = document.querySelector("#input-tarea");  
const listaTareas = document.querySelector("#lista-tareas");  
  
// 3. Función para renderizar (dibujar) todas las tareas en el HTML  
function renderizarTareas() {  
    // Limpiar la lista antes de volver a dibujar
```

```

listaTareas.innerHTML = "";

// Recorrer el array tareas y crear un <li> por cada una
tareas.forEach(function (tarea, indice) {
    // Crear el <li> principal
    const li = document.createElement("li");
    li.classList.add("tarea");

    // Añadir una clase si está completada
    if (tarea.completada) {
        li.classList.add("tarea-completada");
    }

    // Elemento para el texto
    const spanTexto = document.createElement("span");
    spanTexto.classList.add("tarea-texto");
    spanTexto.textContent = tarea.texto;

    // Botón eliminar
    const botonEliminar = document.createElement("button");
    botonEliminar.textContent = "Eliminar";
    botonEliminar.classList.add("btn-eliminar");
    // Guardamos el índice en un atributo data
    botonEliminar.dataset.indice = indice;

    // (Opcional) Botón completar
    const botonCompletar = document.createElement("button");
    botonCompletar.textContent = tarea.completada ? "Desmarcar" :
    "Completar";
    botonCompletar.dataset.indice = indice;

    // Añadir elementos al <li>
    li.appendChild(spanTexto);
    li.appendChild(botonCompletar);
    li.appendChild(botonEliminar);

    // Añadir el <li> a la lista <ul>
    listaTareas.appendChild(li);
});
}

```

De momento no añadimos tareas aún; solo hemos preparado:

- el array `tareas`,
- las referencias al DOM,
- la función `renderizarTareas()`.

## Checklist Fase 2

- Existe un `const tareas = []` en `script.js`.
- Has seleccionado `#form-tarea`, `#input-tarea` y `#lista-tareas`.
- Tienes una función `renderizarTareas()` que:
  - limpia la lista,
  - recorre el array `tareas`,
  - crea elementos `<li>` con el texto de cada tarea.
- No ves errores en la consola al recargar la página (aunque todavía no haya tareas).

## Fase 3 – Añadir tareas nuevas (≈ 35 min)

Objetivo: cada vez que el usuario envíe el formulario, se añadirá una tarea al array y se actualizará la lista.

### 3.1 Manejar el envío del formulario

Debajo del código anterior, añade:

```
// FASE 3: Añadir nuevas tareas
```

```

formulario.addEventListener("submit", function (evento) {
  evento.preventDefault(); // Evita recargar la página

  // 1. Obtener el texto de la tarea
  const texto = inputTarea.value.trim();

  // 2. Si el input está vacío, no hacemos nada
  if (texto === "") {
    alert("Por favor, escribe una tarea antes de añadir.");
    return;
  }

  // 3. Crear un objeto tarea y añadirla al array
  const nuevaTarea = {
    texto,
    completada: false
  };

  tareas.push(nuevaTarea);

  // 4. Limpiar el input y volver a dibujar la lista
  inputTarea.value = "";
  inputTarea.focus();

  renderizarTareas();
});

```

Prueba el formulario:

1. Escribe una tarea en el campo de texto.
2. Pulsa “Añadir tarea”.
3. Comprueba que aparece un <li> con la tarea y los botones correspondientes.

### Checklist Fase 3

- Al enviar el formulario NO se recarga la página.
- Si el input está vacío, la app avisa y no añade nada.

- ✓ Cada nueva tarea se guarda como objeto en el array `tareas`.
- ✓ Después de añadir la tarea, el input se vacía y mantiene el foco.
- ✓ `renderizarTareas()` muestra la lista actualizada en el HTML (1 elemento, luego 2, etc.).

## Fase 4 – Eliminar tareas ( $\approx 30$ min)

Objetivo: que el botón “Eliminar” borre la tarea correspondiente del array y de la lista.

Hay varias formas de hacerlo; usaremos una sencilla con **evento delegado** en la lista.

### 4.1 Escuchar clics en la lista

Debajo del código anterior:

```
// FASE 4: Eliminar tareas

listaTareas.addEventListener("click", function (evento) {
  const elemento = evento.target;

  // ¿Han hecho clic en un botón "Eliminar"?
  if (elemento.classList.contains("btn-eliminar")) {
    const indice = elemento.dataset.indice; // string
    // Convertir a número por claridad
    const indiceNumero = Number(indice);

    // Eliminar ese elemento del array
    tareas.splice(indiceNumero, 1);

    // Volver a dibujar la lista
    renderizarTareas();
  }
});
```

Importante: la posición de cada tarea en el array (índice) está guardada en `data-indice`, por eso sabemos cuál borrar.

Prueba:

1. Crea 3 tareas.
2. Elimina la segunda.
3. Comprueba que el array se actualiza y la lista también (no hay huecos raros).

## Checklist Fase 4

- Cada tarea muestra un botón “Eliminar”.
- Al pulsar “Eliminar” en una tarea concreta, solo se borra esa.
- No aparecen errores en consola al eliminar.
- Si borras todas, la lista queda vacía otra vez.

## Fase 5 - (Opcional) Marcar tareas como completadas (≈ 30 min)

Objetivo: permitir marcar una tarea como completada visualmente cambiando su estilo.

### 5.1 Marcar/desmarcar desde el mismo listener

Aprovechamos el mismo addEventListener de la Fase 4, añadiendo lógica para el botón “Completar”:

```
// FASE 5: Completar tareas (opcional)

// Seguimos en el mismo listener de listaTareas
listaTareas.addEventListener("click", function (evento) {
  const elemento = evento.target;

  // Eliminar tarea
  if (elemento.classList.contains("btn-eliminar")) {
```

```

        const indice = Number(elemento.dataset.indice);
        tareas.splice(indice, 1);
        renderizarTareas();
    }

    // Completar / desmarcar tarea
    else if (elemento.textContent === "Completar" || elemento.textContent === "Desmarcar") {
        const indice = Number(elemento.dataset.indice);
        // Cambiamos el estado completada
        tareas[indice].completada = !tareas[indice].completada;
        // Volvemos a dibujar la lista para actualizar estilos y texto del botón
        renderizarTareas();
    }
});

```

Y recuerda que:

- En `renderizarTareas()` ya se estaba añadiendo la clase `.tarea-completada` si `tarea.completada` es `true`.
- La clase `.tarea-completada` en CSS pone el texto tachado y con menos opacidad.

## Checklist Fase 5

- Cada tarea muestra también un botón “Completar” (o “Desmarcar”).
- Al pulsar “Completar”, el texto de la tarea aparece tachado (o con el estilo que hayas definido).
- Al pulsar “Desmarcar”, el texto vuelve a la normalidad.
- El comportamiento sigue funcionando bien aunque añadas y borres tareas varias veces.

## Checklist final de la práctica

Antes de terminar, revisa:

- Has creado un proyecto con HTML, CSS y JS separados.
- Has añadido un formulario para escribir nuevas tareas y un botón “Añadir tarea”.
- Cada envío del formulario crea una nueva tarea en un **array de objetos** y la muestra en la lista.
- Cada tarea tiene un botón “Eliminar” que la borra de la lista y del array.
- (Opcional) Puedes marcar tareas como completadas y el estilo cambia.
- No hay errores grandes en consola o, si los hay, sabes decir qué significan.

Si todo esto se cumple, ya tienes una **To-Do App funcional** usando exactamente los conceptos vistos en clase: variables, arrays, DOM, eventos...