EJERCICIOS DOM PARTE 2: PRÁCTICA AVANZADA

1.- RENDERIZADO DESDE ESTADO

- **Objetivo:** Separar estado y DOM.
- **Descripción:**
  Crea un objeto `gameState` con un array de 8 peones para un jugador.
  Escribe una función `renderPawns(state)` que pinte los peones en un `<div id="board"></div>` como texto " ♙ ".
- **Requisitos:**
  - No tocar el HTML más que el `<div id="board">`.
  - Cada vez que se llama a `renderPawns`, el tablero se actualiza completamente desde `gameState`.

2.- ACTUALIZACIÓN DE UNA PIEZA

- **Objetivo:** Actualizar el DOM a partir del estado.
- **Descripción:**
  Agrega un método `movePawn(fromIndex, toIndex)` que modifique el array en `gameState`.
  Actualiza la vista llamando a `renderPawns(state)`.
- **Requisitos:**
  - Validar que `toIndex` esté dentro de rango 0-7.
  - Mostrar la acción en consola: `"Peón movido de X a Y"`.

3.- AÑADIR Y ELIMINAR PIEZAS DINÁMICAMENTE

- **Objetivo:** Manipular nodos de forma dinámica.
- **Descripción:**
  Crea botones "Añadir Peón" y "Eliminar Peón".
  - Añadir → agrega un peón al final del array y actualiza DOM.
  - Eliminar → quita el último peón y actualiza DOM.
- **Requisitos:**
  - Actualizar contador de peones en pantalla con `.textContent`.
  - Evitar manipular directamente el HTML.

4.- TABLERO 8X8 CON CLASES

- **Objetivo:** Crear un tablero completo.
- **Descripción:**
  Genera un tablero 8x8 dinámicamente usando `createElement`.
  - Alterna clases `.light` y `.dark` automáticamente.
  - Cada celda debe tener un `data-pos="a1"...h8`.
- **Requisitos:**
  - Solo DOM, sin HTML inicial para las celdas.
  - Escribir una función `createBoard()`.

5.- INTERACCIÓN CON CELDAS

- **Objetivo:** Eventos dinámicos y delegación.
- **Descripción:**
  Cada celda del tablero debe tener un evento `click` que muestre un `alert` con la posición (`data-pos`).
- **Requisitos:**
  - Usar **delegación de eventos** sobre el contenedor `#board`.
  - No añadir `addEventListener` a cada celda individualmente.

6.- MOSTRAR ESTADO DE PIEZAS

- **Objetivo:** Refuerzo de estado vs DOM.
- **Descripción:**
  Añade un objeto `gameState` con `{pieces: [{type:"pawn", pos:"e2"}, ...]}`.
  - Función `renderPieces(state)` pinta las piezas en las celdas correctas.
- **Requisitos:**
  - Solo pintar desde el estado.
  - Evitar leer `innerText` del DOM para calcular el estado.

7.- RESALTAR POSIBLES MOVIMIENTOS

- **Objetivo:** Manipular clases y estilo según condiciones.
- **Descripción:**
  - Cuando se hace click en un peón, resalta las casillas donde puede moverse (`.highlight`).
  - Click en otra casilla → mover el peón si es legal y limpiar los highlights.
- **Requisitos:**
  - Solo modificar clases CSS (`classList.add/remove`).
  - Actualizar `gameState` al mover la pieza.

8.- PANEL DE INFORMACIÓN

- **Objetivo:** Refuerzo de modularidad y actualización del DOM.
- **Descripción:**
  Crear un `<div id="panel"></div>` que muestre:
  `"Turno actual: <blancas/negras>"` y `"Número de movimientos: X"`.
- **Requisitos:**
  - Función `updatePanel(state)` actualiza contenido con `.textContent`.
  - Llamar a esta función después de cada movimiento.

9.- CONTROL DE EVENTOS MÚLTIPLES

- **Objetivo:** Diferentes tipos de eventos.
- **Descripción:**
  - Tecla `r` → reinicia tablero.
  - Tecla `u` → deshacer último movimiento.
  - Click en un botón "Mostrar estado" → console.log del `gameState`.
- **Requisitos:**
  - Usar `keydown` y `click`.
  - Modularizar el código, cada función independiente.

10.- MINI-FORMULARIO DE CONFIGURACIÓN DE PARTIDA

- **Objetivo:** Formularios, validación y DOM.
- **Descripción:**
  - Inputs: `Tiempo por jugador`, `Incremento`, `Color inicial`.
  - Botón "Crear partida".
  - Validar que el tiempo > 0, incremento ≥ 0.
  - Mostrar mensaje de error en `<p>` si hay fallo.
  - Si todo OK → actualizar `gameState` y renderizar tablero inicial.
- **Requisitos:**
  - Validación en JS, no usar HTML5 `pattern`.
  - Modularizar validadores y funciones de actualización del DOM.