🧠 1. Estructura general del archivo

El código se organiza en bloques temáticos bien separados con comentarios /* ========== ... ========== */.

Cada bloque representa **una fase del flujo de ejecución**:

1. Variables y utilidades comunes
2. Carga del test (desde JSON)
3. Renderizado en pantalla
4. Gestión de respuestas y evaluación
5. Botones de navegación
6. Exportar, reiniciar y extras

⚙️ 2. Funciones básicas y estado global

```
const qs = s => document.querySelector(s);

const shuffle = arr => arr.sort(() => Math.random() - 0.5);
```

- qs() → simplifica la selección de elementos del DOM (en lugar de escribir document.querySelector todo el rato).
- shuffle() → mezcla aleatoriamente los elementos de un array (sirve para barajar preguntas o respuestas).

```
const KEY = "quiz_state_v1";

const saveState = s => localStorage.setItem(KEY, JSON.stringify(s));

const loadState = () => JSON.parse(localStorage.getItem(KEY) || "null");

const clearState = () => localStorage.removeItem(KEY);
```

- Se usan para **guardar el progreso** del test en el navegador.
- Así, si recargas la página, tus respuestas no se pierden (usa localStorage).

```
let state = {
  meta: {},
  questions: [],
  index: 0,
  answers: {},
```

```
  submitted: false
};
```

- state contiene **toda la información viva del test**:
  título, lista de preguntas, índice actual, respuestas del usuario, etc.

📥 3. Carga de un test desde un archivo JSON

```
async function cargarTest(ruta = "./Temas/Sostenibilidad.json") {
  try {
    const respuesta = await fetch(ruta);
    if (!respuesta.ok) throw new Error("Error al cargar el archivo JSON");

    const datos = await respuesta.json();
    console.log(`Test cargado: ${datos.meta.title} (${datos.questions.length} preguntas)`);
    inicializarTest(datos);
  } catch (err) {
    console.error("Error al cargar el test:", err);
    alert("No se pudo cargar el test. Revisa la ruta o el formato del JSON.");
  }
}
```

- Carga el JSON con las preguntas (usando fetch()).
- Si todo va bien, llama a inicializarTest(datos) para montar el test.
- Si hay un error (ruta incorrecta, JSON mal formado...), lo muestra en la consola.

🧩 4. Inicialización del test

```
function inicializarTest(datos) {
  state.meta = { ...datos.meta };
  state.questions = shuffle(datos.questions);
```

```
  const guardado = loadState();
  if (guardado && guardado.questions?.length === state.questions.length) {
    state.answers = guardado.answers || {};
    state.index = Math.min(guardado.index || 0, state.questions.length - 1);
  } else {
    clearState();
    state.answers = {};
    state.index = 0;
  }

  qs("#quiz-title").textContent = state.meta.title || "Test";
  qs("#quiz-desc").textContent = state.meta.description || "";
  renderizarPregunta();
}
```

💡 Esta función:

- Guarda en state el contenido del JSON.
- Si hay datos guardados en localStorage, los recupera (para continuar donde te quedaste).
- Luego actualiza el encabezado del test y llama a renderizarPregunta().

🧱 5. Renderizado de la pregunta actual

```
function renderizarPregunta() {
  const contenedor = qs("#quiz");
  contenedor.innerHTML = "";

  const pregunta = state.questions[state.index];
  if (!pregunta) {
```

```
    contenedor.innerHTML = "<p>No hay preguntas disponibles.</p>";
    return;
  }
```

- Limpia el contenido anterior.
- Busca la pregunta actual según state.index.

Luego genera el contenido dinámico:

```
const card = document.createElement("div");
card.className = "q-card";

const titulo = document.createElement("h3");
titulo.textContent = `${state.index + 1}. ${pregunta.question}`;
card.appendChild(titulo);

const bloque = document.createElement("div");
bloque.className = "options";

pregunta.options.forEach((opt, i) => {
  const label = document.createElement("label");
  label.className = "option";
  label.innerHTML = `
    <input type="radio" name="${pregunta.id}" value="${i}">
    <span>${opt.text}</span>`;
  bloque.appendChild(label);
});
```

- Crea dinámicamente los inputs <input type="radio"> o <input type="checkbox"> según el tipo de pregunta.
- Finalmente, lo inyecta en el DOM dentro de <main id="quiz">.

✍️ 6. Guardar la respuesta del usuario

```
function guardarRespuesta(id, bloque) {
  const seleccion = bloque.querySelector(`input[name="${id}"]:checked`);
  state.answers[id] = seleccion ? Number(seleccion.value) : undefined;
  saveState(state);
}
```

- Se ejecuta cada vez que el usuario selecciona una respuesta.
- Guarda la opción elegida en state.answers[id].
- Actualiza el localStorage.

🧮 7. Evaluar el test

```
function evaluarTest() {
  let puntos = 0;
  const total = state.questions.length;
  const resumen = [];

  for (const p of state.questions) {
    const correcta = p.options.findIndex(o => o.correct);
    const elegida = state.answers[p.id];
    const esCorrecta = elegida === correcta;
    if (esCorrecta) puntos++;
```

💡 Esta parte compara las respuestas del usuario con las correctas (o.correct) y:

- Suma los puntos obtenidos.
- Crea un **resumen detallado** con el resultado de cada pregunta.

```
qs("#score").textContent = `${puntos} / ${total} correctas`;
qs("#result").style.display = "flex";
```

👉 Muestra la puntuación final y un desglose en la sección #review.

🧭 8. Botones de navegación

```
qs("#prev").addEventListener("click", () => {
  if (state.index > 0) {
    state.index--;
    renderizarPregunta();
  }
});

qs("#next").addEventListener("click", () => {
  if (state.index < state.questions.length - 1) {
    state.index++;
    renderizarPregunta();
  }
});
```

Controlan el movimiento entre preguntas (anterior / siguiente).

📜 9. Enviar y reiniciar el test

```
qs("#submit").addEventListener("click", () => {
  evaluarTest();
  window.scrollTo({ top: 0, behavior: "smooth" });
});
```

- Corrige el test y muestra los resultados.

```
qs("#retry").addEventListener("click", () => {
  clearState();
  cargarTest();
});
```

- Reinicia el test desde cero.

🧰 10. Extras (shuffle, exportar, imprimir, CSV...)

🔄 Reiniciar completamente

```
qs("#reset").addEventListener("click", () => {
  if (confirm("¿Seguro que quieres reiniciar el test completo?")) {
    clearState();
    cargarTest();
  }
});
```

🔀 Barajar preguntas

```
qs("#shuffle").addEventListener("change", (e) => {
  if (e.target.checked) {
    state.questions = shuffle(state.questions);
    state.index = 0;
    renderizarPregunta();
    alert("Preguntas barajadas");
  }
});
```

💾 Exportar plantilla JSON

Guarda el test actual en un archivo descargable.

📄 Exportar CSV

Convierte tus respuestas a formato CSV (útil para subir a Moodle o Excel).

🖨️ Imprimir

Manda los resultados a la impresora.

🚀 11. Inicio automático

```
cargarTest(); // Carga por defecto "Temas/Sostenibilidad.json"
```

Cuando cargas la página, se ejecuta automáticamente y renderiza el primer test.

🧭 FLUJO GENERAL DEL PROGRAMA

1️⃣ Usuario entra en la página

2️⃣ JS ejecuta cargarTest() → lee el JSON

3️⃣ inicializarTest(datos) → carga meta y preguntas

4️⃣ renderizarPregunta() → muestra la primera pregunta

5️⃣ Usuario responde → guardarRespuesta()

6️⃣ Pulsa "Siguiente" → renderizarPregunta() avanza

7️⃣ Pulsa "Enviar" → evaluarTest()

8️⃣ Se muestran resultados + opciones de exportar / imprimir