EVIDENCIAS 5

```js
// TEMA: 4.1 Objetos
// Creación, acceso, modificación, adición y eliminación de propiedades de objetos.
// Iteración sobre propiedades.
console.log("--- TEMA 4.1: Objetos ---");
let usuario = { nombre: "linda", edad: 15, "es Admin": true };
console.log("Nombre:", usuario.nombre, "Edad:", usuario["edad"], "¿Es Admin?", usuario["es Admin"]);
usuario.edad = 31; // Modificar
usuario.email = "juan@example.com"; // Añadir
delete usuario.edad; // Eliminar
console.log("¿Tiene nombre?", "nombre" in usuario, "Usuario:", usuario);
for (let key in usuario) console.log(`${key}: ${usuario[key]}`);
document.getElementById('output-4-1').innerText = `Nombre: ${usuario.nombre}, Email: ${usuario.email}`
// Fin TEMA 4.1

// TEMA: 4.2 Referencias de objetos y copia
// Los objetos se asignan por referencia. Métodos para copia superficial (Object.assign, spread).
console.log("\n--- TEMA 4.2: Referencias de objetos y copia ---");
let user = { name: "sofia", age: 25 };
let admin = user; // Referencia
admin.age = 26;
console.log("Edad de user (modificado por admin):", user.age);
console.log("¿Son user y admin el mismo objeto?", user === admin); // true

let clon = Object.assign({}, user); // Copia superficial con Object.assign
clon.name = "Clonado";
console.log("Nombre user original:", user.name, "Nombre clon:", clon.name);

let clon3 = { ...user }; // Copia superficial con spread
clon3.age = 50;
console.log("Edad user original:", user.age, "Edad clon3:", clon3.age);
document.getElementById('output-4-2').innerText = `user.age: ${user.age}, clon.name: ${clon.name}`;
// Fin TEMA 4.2

// TEMA: 4.3 Recolección de basura
// JavaScript gestiona automáticamente la memoria, liberando objetos inalcanzables.
console.log("\n--- TEMA 4.3: Recolección de basura ---");
let obj1 = { data: "datos" };
```

```js
// main.js - Resumen de Prácticas del Curso de JavaScript
// Este archivo contiene un resumen conciso de cada tema de "Tipos de Datos",
// con sus principales conceptos y aprendizajes, tal como se implementó.

// ========================================================================
// TEMA: 5.1 Métodos de primitivos — Resumen
// Aprendizaje clave: JavaScript permite que los tipos de datos primitivos (ej. strings, numbers)
// usen métodos como si fueran objetos, convirtiéndolos temporalmente.
// Ejemplos: `str.toUpperCase()`, `num.toFixed(2)`.
// ========================================================================
console.log("5.1 Métodos de primitivos: Primitivos usan métodos de objeto temporalmente.");

// ========================================================================
// TEMA: 5.2 Números — Resumen
// Aprendizaje clave: Manejo de enteros y flotantes. Conversiones con `Number()`.
// Propiedades `NaN` (Not a Number) e `Infinity`. Funciones globales `isNaN()` y `isFinite()`.
// Uso del objeto `Math` para operaciones como `floor()`, `ceil()`, `round()`, `random()`.
// Importancia de la precisión de punto flotante (`0.1 + 0.2` no siempre es `0.3`).
// ========================================================================
console.log("5.2 Números: Conversiones, NaN/Infinity, Math object y precisión.");

// ========================================================================
// TEMA: 5.3 Cuerdas (Strings) — Resumen
// Aprendizaje clave: Las cadenas son inmutables (los métodos devuelven nuevas cadenas).
// Propiedad `length`. Acceso a caracteres (`[0]`, `charAt()`).
// Búsqueda: `indexOf()`, `includes()`, `startsWith()`, `endsWith()`.
// Extracción: `slice()`, `substring()` (preferidos).
// Modificación: `replace()`. Transformación: `toUpperCase()`, `toLowerCase()`.
// Limpieza: `trim()`.
// ========================================================================
console.log("5.3 Cuerdas: Inmutabilidad, longitud, acceso, búsqueda, extracción y transformación.");

// ========================================================================
// TEMA: 5.4 Matrices (Arrays) — Resumen
// Aprendizaje clave: Arrays como colecciones ordenadas, mutables y con índices base 0.
// Acceso y modificación por índice.
```

```js
// ========================================================================
// `filter()`: Crea un nuevo array con elementos que pasan una prueba.
// `find()`: Devuelve el primer elemento que cumple una condición.
// `findIndex()`: Devuelve el índice del primer elemento que cumple una condición.
// `some()`, `every()`: Verifican si algunos/todos cumplen una condición.
// `reduce()`: Reduce el array a un único valor.
// `sort()`: Ordena el array (modifica el original, necesita comparador para números).
// `concat()`: Une arrays, creando uno nuevo.
// `join()`: Une elementos de array en una cadena.
// ========================================================================
console.log("5.5 Métodos de matriz: forEach, map, filter, find, reduce, sort, concat, join para manip

// ========================================================================
// TEMA: 5.6 Iterables — Resumen
// Aprendizaje clave: Objetos que pueden ser recorridos (iterados) con `for...of`.
// Cadenas y arrays son iterables.
// Objetos literales NO son iterables directamente con `for...of` (se necesitan `Object.keys()`, etc.
// `Array.from()`: Convierte un iterable (o array-like) en un array real.
// ========================================================================
console.log("5.6 Iterables: for...of para strings/arrays. Objetos no iterables directamente. Array.fr

// ========================================================================
// TEMA: 5.7 Mapa y conjunto (Map y Set) — Resumen
// Aprendizaje clave:
// `Map`: Colección de clave-valor. Las claves pueden ser de CUALQUIER tipo (a diferencia de objetos)
//   Métodos: `set()`, `get()`, `has()`, `delete()`, `size`. Es iterable.
// `Set`: Colección de valores ÚNICOS. Ignora duplicados.
//   Métodos: `add()`, `has()`, `delete()`, `size`. Es iterable.
// ========================================================================
console.log("5.7 Mapa y conjunto: Map (clave-valor con claves de cualquier tipo), Set (valores únicos

// ========================================================================
// TEMA: 5.8 Mapa débil y conjunto débil (WeakMap y WeakSet) — Resumen
// Aprendizaje clave: Versiones "débiles" de Map y Set.
// Claves (`WeakMap`) o valores (`WeakSet`) deben ser OBJETOS.
// No impiden la recolección de basura de esos objetos si no hay otras referencias fuertes.
// No son iterables y no tienen `size`. Usos para metadatos sin evitar limpieza de memoria.
```

```js
// ========================================================================
// TEMA: 5.8 Mapa débil y conjunto débil (WeakMap y WeakSet) — Resumen
// Aprendizaje clave: Versiones "débiles" de Map y Set.
// Claves (`WeakMap`) o valores (`WeakSet`) deben ser OBJETOS.
// No impiden la recolección de basura de esos objetos si no hay otras referencias fuertes.
// No son iterables y no tienen `size`. Usos para metadatos sin evitar limpieza de memoria.
// ========================================================================
console.log("5.8 Mapa débil y conjunto débil: WeakMap/WeakSet (claves/valores objetos), no evitan recole

// ========================================================================
// TEMA: 5.9 Objeto.claves, valores, entradas (Object.keys, values, entries) — Resumen
// Aprendizaje clave: Métodos estáticos para extraer partes de objetos literales.
// `Object.keys()`: Devuelve un array con las claves (propiedades).
// `Object.values()`: Devuelve un array con los valores.
// `Object.entries()`: Devuelve un array de `[clave, valor]` pares.
// Esenciales para iterar y manipular propiedades de objetos.
// ========================================================================
console.log("5.9 Objeto.claves, valores, entradas: Object.keys(), Object.values(), Object.entries() para

// ========================================================================
// TEMA: 5.10 Asignación de desestructuración (Destructuring assignment) — Resumen
// Aprendizaje clave: Sintaxis para "desempaquetar" valores de arrays o propiedades de objetos
// directamente en variables separadas, de forma más concisa.
// Desestructuración de arrays: `[a, b] = [1, 2]`.
// Desestructuración de objetos: `{ nombre, edad } = usuario`.
// Soporta valores por defecto, renombrado de variables y parámetros de función.
// ========================================================================
console.log("5.10 Asignación de desestructuración: Desempaquetar valores de arrays/objetos en variables

// ========================================================================
// TEMA: 5.11 Fecha y hora (Date and time) — Resumen
// Aprendizaje clave: Uso del objeto `Date` para manejar fechas y horas.
// Crear fechas: `new Date()`, `new Date(string)`, `new Date(año, mes, ...)`.
// Métodos para obtener componentes: `getFullYear()`, `getMonth()`, `getDate()`, `getHours()`, `getMinut
// Métodos para establecer componentes: `setFullYear()`, `setMonth()`, etc.
// Obtener timestamp: `getTime()`.
// ========================================================================
console.log("5.11 Fecha y hora: Objeto Date para crear, obtener y manipular fechas/horas.");
```