Evidencias de trabajo avanzado

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prácticas del Curso de JavaScript</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-color: #fce4ec; /* Rosa claro, similar a tu imagen */
            color: #333;
        }
        h1 {
            color: #c2185b; /* Un rosa más oscuro para el título */
            text-align: center;
            margin-bottom: 30px;
        }
        .section-container {
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
            padding: 20px;
            margin-bottom: 20px;
        }
        h2 {
            color: #880e4f; /* Un rosa aún más oscuro para los subtítulos */
            border-bottom: 2px solid #f8bbd0;
            padding-bottom: 10px;
            margin-top: 0;
        }
        p {
            line-height: 1.6;
        }
        .code-output {
```

```html
/head>
body>
    <h1>Prácticas del Curso de JavaScript</h1>
    <p style="text-align: center; margin-bottom: 30px;">(F12 para ver la mayoría de los resu

    <div id="content-area">
        <!-- Las secciones de los temas se generarán aquí o se mostrarán sus resultados -->

        <!-- Ejemplo de cómo se vería una sección, luego main.js interactuará con ella -->
        <section id="tema-6-1" class="section-container">
            <h2>6.1 Recursión y pila</h2>
            <p>Aquí se demostrará la recursión, con un ejemplo de cálculo factorial y la sum
            <div class="code-output" id="output-6-1"></div>
            <p class="console-note">Ver la consola para la traza de pila de la recursión.</p
        </section>

        <section id="tema-6-2" class="section-container">
            <h2>6.2 Parámetros de descanso y sintaxis de propagación</h2>
            <p>Demostración de cómo los parámetros "rest" recopilan argumentos restantes en
            <div class="code-output" id="output-6-2"></div>
        </section>

        <section id="tema-6-3" class="section-container">
            <h2>6.3 Alcance variable, cierre</h2>
```

```
main.js > ...
1   // main.js - Código del curso JavaScript.info
2   // TEMA: 6.1 Recursión y pila — 24/10/2025
3   // Este bloque demuestra el concepto de recursión mediante ejemplos de cálculo factorial
4   // y la suma de una serie de números. La recursión es una función que se llama a sí misma
5   // hasta que se cumple una condición base. Es crucial entender cómo funciona la pila de ll
6   // Función recursiva para calcular el factorial de un número
7   function factorial(n) {
8       // Caso base: si n es 0 o 1, el factorial es 1
9       if (n === 0 || n === 1) {
10          console.log(`Factorial(${n}): Caso base alcanzado, retornando 1.`);
11          return 1;
12      } else {
13          // Paso recursivo: n * factorial(n-1)
14          console.log(`Factorial(${n}): Llamando a Factorial(${n-1}).`);
15          return n * factorial(n - 1);
16      }
17  }
18  // Función recursiva para sumar una serie de números de 1 a n
19  function sumNumbers(n) {
20      if (n === 1) {
21          console.log(`sumNumbers(${n}): Caso base, retornando 1.`);
22          return 1;
23      } else {
24          console.log(`sumNumbers(${n}): Sumando ${n} + sumNumbers(${n-1}).`);
25          return n + sumNumbers(n - 1);
26      }
27  }
28  // Practicando e imprimiendo resultados
29  console.log("--- TEMA 6.1: Recursión y pila ---");
30  const numFactorial = 5;
31  const resFactorial = factorial(numFactorial);
32  console.log(`El factorial de ${numFactorial} es: ${resFactorial}`); // Esperado: 120
33  const numSum = 4;
34  const resSum = sumNumbers(numSum);
35  console.log(`La suma de números hasta ${numSum} es: ${resSum}`); // Esperado: 10 (1+2+3+4)
36  // Actualizar el HTML para mostrar un resumen del resultad
37  document.getElementById('output-6-1').innerHTML = `
```

```
// TEMA: 6.6 Objeto de función, NFE — 24/10/2025
// Este bloque explora el hecho de que las funciones en JavaScript son ob
// clase. Tienen propiedades y métodos, y pueden ser pasadas como argumen
// por otras funciones, etc.
// También se introduce el concepto de NFE (Named Function Expression), q
// expresión de función con un nombre interno, útil para la recursión y d
console.log("\n--- TEMA 6.6: Objeto de función, NFE ---");
// 1. Funciones como objetos: Propiedades `name`, `length`
function greet(who, greeting = "Hola") {
    console.log(`${greeting}, ${who}!`);
}
console.log("Nombre de la función greet:", greet.name); // "greet"
console.log("Número de argumentos esperados (sin valores por defecto) en
// También podemos añadir propiedades personalizadas a las funciones
greet.customProperty = "Esta es una propiedad personalizada.";
console.log("Propiedad personalizada de greet:", greet.customProperty);

// 2. Named Function Expression (NFE)
let sayHi = function func(name) { // 'func' es el nombre interno, solo vi
    if (name) {
        console.log(`Hola, ${name}`);
    } else {
        console.log("No hay nombre, llamando a la propia función internam
        func("Invitado"); // Podemos usar 'func' internamente para recurs
    }
};
sayHi("Alice"); // Llamada normal a través de la variable externa
sayHi();        // Llamada interna usando el nombre 'func'
// console.log(func); // Error: func no está definida fuera de la expresi
console.log("Nombre de la función 'sayHi' (vista desde fuera):", sayHi.na
// Otro ejemplo NFE: Contador en una función recursiva anónima
let countdown = function doCountdown(n) {
    if (n > 0) {
        console.log(`Contando: ${n}...`);
        setTimeout(() => doCountdown(n - 1), 1000); // Uso de 'doCountdow
```

```
// TEMA: 6.4 El viejo "var" — 24/10/2025
// Este bloque se enfoca en las características de la palabra clave `var`, que fue
// la forma principal de declarar variables antes de ES6 (ES2015).
// A diferencia de `let` y `const`, `var` tiene alcance de función y hoisting.
// - Alcance de función: Las variables `var` solo son accesibles dentro de la función
//   en la que fueron declaradas, ignorando los bloques `if`, `for`, etc.
// - Hoisting: Las declaraciones `var` son "elevadas" al principio de su alcance de
//   función (o global), aunque su asignación permanezca en su lugar original. Esto
//   puede llevar a comportamientos inesperados (acceder a una variable antes de su
//   declaración aparente, pero con valor `undefined`).
console.log("\n--- TEMA 6.4: El viejo 'var' ---");
// 1. Alcance de función vs. Alcance de bloque
if (true) {
    var varInIf = "Soy var dentro de un if";
    let letInIf = "Soy let dentro de un if"; // Alcance de bloque
    const constInIf = "Soy const dentro de un if"; // Alcance de bloque
    console.log("Dentro del if (var):", varInIf);
    console.log("Dentro del if (let):", letInIf);
    console.log("Dentro del if (const):", constInIf);
}
console.log("Fuera del if (var):", varInIf); // varInIf es accesible aquí (alcance de fu
// console.log("Fuera del if (let):", letInIf); // Error: letInIf no está definida
// console.log("Fuera del if (const):", constInIf); // Error: constInIf no está definida
function showVarScope() {
    var funcVar = "Soy var dentro de una función";
    if (true) {
        var anotherVar = "Otra var dentro de un bloque en la función";
        console.log("Dentro del bloque en función (var):", anotherVar);
    }
    console.log("Fuera del bloque, pero dentro de función (var):", anotherVar); // accesi
    return funcVar;
}
console.log("Llamando showVarScope:", showVarScope());
// console.log(funcVar); // Error: funcVar no está definida fuera de la función
// 2. Hoisting con var
```

```
// TEMA: 6.10 Vinculación de funciones — 24/10/2025
// Este bloque explora el método `bind()`, que permite crear una nueva funci
// cuando es llamada, tiene su palabra clave `this` establecida en un valor
// `bind()` también puede preestablecer argumentos a la función. Es muy útil
// asegurar que el contexto de `this` se mantenga en callbacks o en eventos.
console.log("\n--- TEMA 6.10: Vinculación de funciones ---");

const user = {
    firstName: "John",
    sayHi() {
        console.log(`Hola, ${this.firstName}!`);
        return `Hola, ${this.firstName}!`;
    }
};
// Problema: Si pasamos user.sayHi como un callback, pierde su 'this'
// setTimeout(user.sayHi, 1000); // Esto daría "Hola, undefined!" en un nave
// Solución 1: Usar un envoltorio (wrapper)
const wrapperResult = "Hola desde wrapper!";
setTimeout(() => {
    user.sayHi();
    console.log("Solución 1 (wrapper): 'user.sayHi()' se llama correctamente
    document.getElementById('output-6-10').innerHTML += `
        <p><i>Solución 1 (wrapper): 'user.sayHi()' se llama correctamente.</
    `;
}, 1200);
// Solución 2: Usar `bind()` para fijar `this`
const sayHiBound = user.sayHi.bind(user);
const sayHiBoundResult = "Hola desde bind!";
setTimeout(() => {
    sayHiBound(); // Ahora `this` dentro de sayHi es `user`
    console.log("Solución 2 (bind): 'sayHiBound()' se llama correctamente.")
    document.getElementById('output-6-10').innerHTML += `
```

Evidencias de trabajo avanzado

```
// Este bloque repasa las funciones de flecha (arrow functions), que son una sintaxis
// concisa para escribir funciones. Su principal característica es cómo manejan `this`:
// no tienen su propio `this`, sino que lo toman del contexto léxico envolvente.
// Esto las hace ideales para callbacks o métodos que necesitan mantener el `this` del padre.
console.log("\n--- TEMA 6.11: Funciones de flecha revisadas ---");
// 1. Sintaxis concisa
const add = (a, b) => a + b;
console.log("Función de flecha 'add(2, 3)':", add(2, 3)); // 5

const greetUser = name => `Saludos, ${name}!`; // Un solo argumento sin paréntesis
console.log("Función de flecha 'greetUser(\"Ana\")':", greetUser("Ana"));

const sayHello = () => console.log("Hola mundo con flecha!"); // Sin argumentos con paréntesis
sayHello();
// 2. Ausencia de 'this' propio (this léxico)
const group = {
    title: "Nuestro Grupo",
    members: ["Pedro", "Laura"],

    showList() {
        this.members.forEach(member => {
            // Aquí, 'this' en la función de flecha se refiere a 'group', no a 'member' o glob
            console.log(`${this.title}: ${member}`);
            document.getElementById('output-6-11').innerHTML += `
                <p><i>${this.title}: ${member}</i></p>
```

trabajo avanzado

(F12 para ver la mayoría de los resultados en la consola)

6.1 Recursión y pila

Aquí se demostrará la recursión, con un ejemplo de cálculo factorial y la suma de una serie de números.

```
Factorial de 5: 120
Suma de 1 a 4: 10

Resultados detallados y traza de pila en la consola (F12).
```

Ver la consola para la traza de pila de la recursión.

6.2 Parámetros de descanso y sintaxis de propagación

Demostración de cómo los parámetros "rest" recopilan argumentos restantes en un array y cómo el operador "spread" expande un iterable.

```
Parámetros de descanso:
Hola, Julio Verne, tus títulos son: Escritor, Novelista, Aventurero

Sintaxis de propagación (arrays):
Array combinado: [1, 2, 3, 4, 5, 6, 7, 8]
```