EVIDENCIAS 6

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Curso JavaScript - Desarrollo Personal</title>
    <style>
        body {
            font-family: 'Roboto', sans-serif; /* Fuente moderna y limpia */
            margin: 20px;
            background-color: #f0f2f5; /* Fondo gris claro y suave */
            color: #333; /* Texto principal oscuro */
        }
        h1 {
            color: #2c3e50; /* Azul oscuro para el título principal */
            text-align: center;
            font-weight: 700; /* Negrita */
            margin-bottom: 30px;
        }
        h2 {
            color: #34495e; /* Azul grisáceo para los subtítulos */
            border-bottom: 2px solid #3498db; /* Línea azul para destacar */
            padding-bottom: 8px;
            margin-top: 35px;
            font-weight: 600;
        }
        h3 {
            color: #3498db; /* Azul medio para los encabezados de sección */
            margin-top: 25px;
            font-weight: 500;
        }
        section {
            background-color: #ffffff; /* Fondo blanco para las secciones */
            margin-bottom: 25px;
            padding: 20px 25px;
            border-radius: 10px; /* Bordes más suaves */
            box-shadow: 0 4px 10px rgba(0,0,0,0.08); /* Sombra más pronunciada pero sutil */
```

```
Lín. 25, col. 30  Espacios: 4  UT
```

```
// main.js - Código del curso JavaScript.info

// TEMA: 6.1 Recursión y pila — 24/10/2025
// Demuestra recursión con factorial y suma de números.
function factorial(n) {
    if (n === 0 || n === 1) return 1;
    return n * factorial(n - 1);
}
function sumNumbers(n) {
    if (n === 1) return 1;
    return n + sumNumbers(n - 1);
}
const numFactorial = 5, resFactorial = factorial(numFactorial);
const numSum = 4, resSum = sumNumbers(numSum);
console.log(`Factorial de ${numFactorial}: ${resFactorial}`);
console.log(`Suma hasta ${numSum}: ${resSum}`);
document.getElementById('output-6-1').innerHTML = `Factorial de ${numFactorial}: ${resFactorial

// TEMA: 6.2 Parámetros de descanso y sintaxis de propagación — 24/10/2025
// Explora rest parameters (...args) y spread syntax (...array).
function showNames(firstName, lastName, ...titles) {
    return `Hola, ${firstName} ${lastName}, tus títulos son: ${titles.join(', ')}`;
}
const showNamesResult = showNames("Julio", "Verne",  let arr1: number[]  a");
let arr1 = [1, 2], arr2 = [3, 4], combinedArr = [...arr1, ...arr2, 5];
function multiply(a, b, c) { return a * b * c; }
let numbersToMultiply = [2, 3, 4], multiplyResult = multiply(...numbersToMultiply);
console.log(showNamesResult);
console.log("Array combinado:", combinedArr);
console.log("Multiplicación con spread:", multiplyResult);
document.getElementById('output-6-2').innerHTML = `Rest: ${showNamesResult}<br>Spread (arrays):

// TEMA: 6.3 Alcance variable, cierre — 24/10/2025
// Alcance (global, función, bloque) y closures (funciones que recuerdan su entorno).
let globalVar = "Global";
function outerFunction() {
```

```
bajo avanzado 6 > JS main.js > ...
// TEMA: 6.4 El viejo "var" — 24/10/2025
// `var` tiene alcance de función y hoisting (declaración elevada).
if (true) { var varInIf = "var en if"; }
console.log("Fuera del if (var):", varInIf); // Accesible
console.log("Hoisting de var:", hoistedVar); // undefined
var hoistedVar = "Hoisted";
console.log("Después de var:", hoistedVar);
document.getElementById('output-6-4').innerHTML = `'varInIf': ${varInIf}. 'hoistedVar' accesible antes

// TEMA: 6.5 Objeto global — 24/10/2025
// `window` (navegador) o `global` (Node.js), estandarizado por `globalThis`.
window.myGlobalVar = "Desde window";
globalThis.anotherGlobalVar = "Desde globalThis";
var varOnGlobal = "Var es global";
let letOnGlobal = "Let no es global"; // No es propiedad de window
console.log("window.myGlobalVar:", window.myGlobalVar);
console.log("globalThis.anotherGlobalVar:", globalThis.anotherGlobalVar);
console.log("varOnGlobal es propiedad de window:", window.varOnGlobal === varOnGlobal);
console.log("letOnGlobal es propiedad de window:", window.letOnGlobal === undefined);
document.getElementById('output-6-5').innerHTML = `window.myGlobalVar: ${myGlobalVar}<br>globalThis.ar

// TEMA: 6.6 Objeto de función, NFE — 24/10/2025
// Las funciones son objetos (`.name`, `.length`). NFE (Named Function Expression)
// para nombres internos en recursión/depuración.
function greet(who, greeting = "Hola") { console.log(`${greeting}, ${who}!`); }
greet.custom = "Propiedad custom";
console.log(`greet.name: ${greet.name}, greet.length: ${greet.length}, greet.custom: ${greet.custom}`)
let sayHi = function func(name) { // NFE
    if (name) console.log(`Hola, ${name}`);
    else func("Invitado"); // Uso interno
};
sayHi("Alice"); sayHi();
document.getElementById('output-6-6').innerHTML = `greet.name: ${greet.name}, greet.length: ${greet.le

// TEMA: 6.7 La sintaxis de la "nueva función" — 24/10/2025
// `new Function('args', 'body')` crea funciones dinámicamente; no captura alcance léxico.
let sum = new Function('a', 'b', 'return a + b');
```

```
Lín. 125, col. 90  Espacios: 4  UTF-8  CR
```

```
avanzado 6 > JS main.js > ...
// TEMA: 6.7 La sintaxis de la "nueva función" — 24/10/2025
// `new Function('args', 'body')` crea funciones dinámicamente; no captura alcance léxico.
let sum = new Function('a', 'b', 'return a + b');
console.log("new Function 'sum(2,3)':", sum(2, 3));
let globalValue = 10;
let dynamicFunc = new Function('return globalValue * 2;');
console.log("new Function acceso global:", dynamicFunc());
try { new Function('return localValue')(); } catch (e) { console.error("new Function NO accede a local:"
document.getElementById('output-6-7').innerHTML = `'sum(2,3)': ${sum(2,3)}. Acceso a globalValue: ${dyna

// TEMA: 6.8 Programación: setTimeout y setInterval — 24/10/2025
// Funciones para ejecutar código asincrónicamente (`setTimeout` una vez, `setInterval` repetidamente).
setTimeout(() => console.log("setTimeout (2s) ejecutado."), 2000);
let count = 0;
const intervalId = setInterval(() => {
    count++;
    console.log(`setInterval: ${count}`);
    if (count >= 3) clearInterval(intervalId);
}, 1500);
document.getElementById('output-6-8').innerHTML = `setTimeout (2s) y setInterval (1.5s x 3) se ejecutan.

// TEMA: 6.9 Decoradores y reenvío, llamar/aplicar — 24/10/2025
// Decoradores añaden funcionalidad. `call` y `apply` reenvían `this` y argumentos.
function sayHiTo(user) { return `Hola, ${user}!`; }
function loggingDecorator(func) {
    return function(...args) {
        console.log(`Llamando a '${func.name}' con args: ${args}`);
        return func.apply(this, args); // Reenvía this y args
    };
}
let sayHiLogged = loggingDecorator(sayHiTo);
sayHiLogged("Juan");
function showDetails(age, city) { console.log(`${this.name}: ${age}, ${city}`); }
const person1 = { name: "Maria" };
showDetails.call(person1, 30, "Madrid"); // this=person1, args=30, "Madrid"
showDetails.apply({ name: "Pedro" }, [25, "Barcelona"]); // this=Pedro, args=[25, "Barcelona"]
document.getElementById('output-6-9').innerHTML = `sayHiLogged("Juan") decorado. Call/Apply para Maria (
```

JavaScript - Conceptos Avanzados

Presiona <F12> para ver la mayoría de los resultados en la consola del navegador.

6.1 Recursión y Pila de Llamadas

Demostración de la recursión, con ejemplos de cálculo factorial y la suma de una serie numérica, ilustrando la gestión de la pila de llamadas.

```
Factorial de 5: 120

Suma de 1 a 4: 10


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

> Consulta la consola para observar la traza de pila durante la ejecución recursiva.

6.2 Parámetros Rest y Sintaxis Spread