Logotipo

El contenido generado por IA puede ser incorrecto.

Santiago Yosa González

**Ingeniero Sistemas de Información**



Logotipo

El contenido generado por IA puede ser incorrecto.}

**Introducción a JavaScript:**

Tipo de dato

* Number
* String

Variables en JavaScript

* let
* var
* const

Consola de JavaScript

Operadores aritméticos, operadores lógicos y unarios.

**Unidad I**

**Estructuración del sitio Web**

# Situación

Imagina que estás pidiendo un café desde una app.

1. La app te saluda: "¡Hola, [Tu Nombre]!" -> Para "recordar" tu nombre, usa una **variable** y almacena un tipo de dato **String** (texto).
2. Eliges la cantidad: "2" cafés. -> Esto es un **Number** (número).
3. Ves el precio de cada café: "5$". -> Esto también es un **Number**.
4. La app calcula tu total: 2 \* 5 = 10. -> Para hacer esto, usa un **operador aritmético** (\*).
5. La app te pregunta si quieres añadir un croissant por 2$ extra. Tú decides si true (sí) o false (no). -> Para manejar esta decisión, se usan **operadores lógicos**.

JavaScript es el "cerebro" detrás de esa aplicación. Es el lenguaje que toma estos datos, los almacena en "cajas" (variables) y realiza cálculos y decisiones (operadores) para que la aplicación funcione. Hoy, vamos a aprender a usar esas cajas y a realizar esos cálculos.

# Variables (Las Cajas para Guardar Datos)

Una variable es un contenedor con un nombre donde almacenamos un valor para usarlo más tarde. JavaScript es un lenguaje de **tipado dinámico**, lo que significa que no necesitas decirle a la variable qué tipo de dato va a contener (a diferencia de Java o C#).

* **let (La Caja Estándar):** Es la forma moderna y preferida de declarar una variable cuyo valor **puede cambiar**. Solo vive dentro de las llaves { ... } donde fue creada.
* **const (La Caja Fuerte):** Declara una variable cuyo valor **no puede ser reasignado**. Es una "constante". Se debe usar por defecto, a menos que sepas que el valor cambiará.
* **var (La Caja Vieja y "Con Fugas"):** Es la forma antigua de declarar variables. Tiene un comportamiento problemático con el *alcance (scope)*, lo que puede causar errores difíciles de rastrear. **Práctica moderna: Evitar var por completo.**

# var vs. let (Scope y Hoisting)

Este es el "por qué" evitamos var.

* **Scope (Alcance):** Define dónde es visible una variable.
  + var tiene **alcance de función**. Si la declaras dentro de un bloque {} (como un if o un for), ¡sigue existiendo fuera de ese bloque! Esto es peligroso.
  + let y const tienen **alcance de bloque**. Solo existen dentro del bloque {} donde fueron declaradas. Esto es predecible y seguro.
* **Hoisting (Elevación):** JavaScript "eleva" las declaraciones.
  + var es elevado y se inicializa con undefined. Puedes usar la variable *antes* de declararla y su valor será undefined.
  + let y const también son elevadas, pero no se inicializan. Si intentas acceder a ellas antes de su declaración, entras en la "Zona Muerta Temporal" (Temporal Dead Zone - TDZ) y obtienes un ReferenceError. Esto es bueno, ya que te obliga a escribir código más limpio.

# Tipos de Datos Fundamentales

1. **Number (Números)** JavaScript tiene un solo tipo para todos los números, ya sean enteros (10) o decimales (10.5).

* **Operaciones (Aritméticos):** + (suma), - (resta), \* (multiplicación), / (división), \*\* (exponenciación - ES6), % (módulo o residuo de una división).
* **Valores Especiales (Avanzado):**
  + **Infinity:** El resultado de 1 / 0.
  + **NaN (Not a Number):** El resultado de una operación matemática inválida, como 0 / 0 o 'texto' \* 2. Es el único valor en JavaScript que no es igual a sí mismo (NaN === NaN es false).

1. **String (Cadenas de Texto)** Cualquier cosa entre comillas ('') o comillas dobles ("").

* **Concatenación:** Se pueden "sumar" (unir) strings usando el operador +.
* **Template Literals (La Forma Moderna):** Usando backticks (comillas invertidas `), podemos crear strings de manera más limpia e insertar variables directamente. Esto se llama interpolación.

1. **Boolean**

* true (verdadero) o false (falso). Base de toda la lógica computacional.

# Anatomía del Código (Strings)

let nombre = 'Juan';

let edad = 25;

// Forma Antigua (Concatenación)

let saludoViejo = 'Hola, mi nombre es ' + nombre + ' y tengo ' + edad + ' años.';

// Forma Moderna (Template Literals) - ¡Mucho mejor!

let saludoModerno = `Hola, mi nombre es ${nombre} y tengo ${edad} años.`;

console.log(saludoModerno); // "Hola, mi nombre es Juan y tengo 25 años."

Evidencia los tipos con typeof

# Recomendaciones para nombrar variables en JavaScript

* Usa nombres descriptivos
* Usa camelCase
* Evita abreviaciones innecesarias
* No uses palabras reservadas
* No empieces con números o símbolos (se permite $ y \_)
* Evita poner el tipo de dato en el nombre
* Usa nombres afirmativos o con sentido lógico para booleanos
* No uses tildes ni eñes
* Recuerda que JavaScript distingue mayúsculas y minúsculas
* Sé coherente y constante con tu estilo
* Usa la misma convención en todo tu proyecto

# Consola de JavaScript

El navegador (Chrome, Edge, Firefox) tiene un motor de JavaScript integrado (V8 en el caso de Chrome). La consola es nuestra línea directa con ese motor. Comenzaremos abriendo la herramienta más importante para un desarrollador: la **Consola del Navegador** (F12).

**El Laboratorio del Científico:** La consola es nuestro laboratorio. El archivo .js es donde escribimos nuestra teoría, pero la consola es donde la probamos en tiempo real. Usaremos console.log() como nuestro "microscopio" para ver qué está pasando dentro de nuestro código.

**Comando clave:** console.log(...) es la forma de "imprimir" mensajes para nosotros los humanos (desarrolladores), no para el usuario final.

console.log("Hola Mundo");

console.log("Viajando a " + ciudad + ", en " + pais + ".");

console.log("El producto es:", producto, "y cuesta:", precio, "dólares.");

Template Literals

console.log(`Mi banda favorita es ${artista} y su mejor canción es ${cancion}.`);

# Operadores (Los "Verbos")

Los operadores nos permiten manipular nuestras variables.

* **Aritméticos:** (+, -, \*, /, %, \*\*). + (suma), - (resta), \* (multiplicación), / (división), \*\* (exponenciación - ES6), % (módulo o residuo de una división).
* **Unarios:** Operan sobre un solo valor.
  + ++ (Incremento): x++ (post-incremento) o ++x (pre-incremento). Aumenta el valor en 1.
  + -- (Decremento): x-- o --x. Disminuye el valor en 1.
  + ! (Negación Lógica - NOT): Invierte un valor booleano (!true es false).
* **Lógicos:** Se usan para tomar decisiones. JS evalúa de izquierda a derecha.
  + && (AND): Devuelve true si **ambos** lados son verdaderos.
  + || (OR): Devuelve true si **al menos uno** de los lados es verdadero.
  + **Short-Circuit:** En A || B, si A es verdadero, JS ni siquiera mira B. Esto se usa para valores por defecto: let nombre = inputUsuario || "Anónimo";
* **Comparación:**
  + > (mayor que), < (menor que), >= (mayor o igual), <= (menor o igual).
  + === (Igualdad Estricta): Comprueba si el valor **Y** el tipo son iguales. **Esta es la que debes usar siempre.**
  + == (Igualdad Débil): Intenta "convertir" los tipos antes de comparar. Es una fuente común de errores.

# Anatomía del Código (Pre vs. Post Incremento)

let a = 5;

let b = 5;

// Post-incremento: usa el valor (5) y LUEGO lo incrementa (a=6)

console.log(a++); // Muestra: 5

// Pre-incremento: incrementa el valor (b=6) y LUEGO lo usa

console.log(++b); // Muestra: 6

# Preguntas

**Acabas de decir que const no se puede cambiar, pero ¿por qué const persona = { nombre: 'Ana' }; persona.nombre = 'Laura'; SÍ funciona?**

Esta es una confusión fundamental. const significa que la variable no puede ser reasignada. No puedes hacer persona = ... (apuntar la "caja" a un objeto totalmente nuevo). Sin embargo, const no hace que el valor dentro de la caja sea inmutable. Si el valor es un objeto (como un {} o un []), sus propiedades internas sí pueden ser modificadas.

**¿Por qué 0.1 + 0.2 no es igual a 0.3?**

¡Bienvenido al mundo de la precisión de punto flotante! Las computadoras usan un sistema binario (base 2) para almacenar números. Los decimales como 0.1 y 0.2 no pueden representarse de forma exacta en binario, de la misma forma que 1/3 no se puede escribir de forma exacta en decimal (0.333...). El resultado de la suma es algo como 0.30000000000000004. Es un problema común a casi todos los lenguajes de programación.

# Ejemplo 1: El Calculador de Total

// Almacenamos valores base

const precioProducto = 150;

const impuesto = 0.19; // 19%

let cantidad = 2;

// Calculamos el subtotal

let subtotal = precioProducto \* cantidad;

// Calculamos el impuesto a pagar

let impuestoTotal = subtotal \* impuesto;

// Calculamos el gran total

let totalAPagar = subtotal + impuestoTotal;

console.log(`Subtotal: $${subtotal}`);

console.log(`Impuesto (19%): $${impuestoTotal}`);

console.log(`Total a Pagar: $${totalAPagar}`);

# Ejemplo 2: Verificador de Acceso a un Bar

let edad = 22;

let tieneIdentificacion = true;

let estaEnListaVIP = false;

// && (AND): Debe cumplir ambas

let puedeBeber = (edad >= 18) && tieneIdentificacion;

console.log(`¿Puede beber? ${puedeBeber}`); // true

// || (OR): Debe cumplir al menos una

let puedeEntrarGratis = estaEnListaVIP || (edad > 25);

console.log(`¿Entra gratis? ${puedeEntrarGratis}`); // false

# Ejemplos Demostrativos (Paso a Paso)

Vamos a simular el sistema de puntos de vida de un videojuego RPG en la consola.

// --- PASO 1: Definición de Constantes (Configuración) ---

// El nombre del juego no cambia, ni la vida máxima inicial.

const NOMBRE\_JUEGO = "Zelda Adventure";

const VIDA\_MAXIMA = 100;

// --- PASO 2: Definición de Variables de Estado (Lo que cambia) ---

let vidaActual = 100;

let tieneEscudo = false; // Booleano

let nombreJugador = "Link";

// --- PASO 3: Operaciones y Lógica ---

// El jugador recibe un golpe de 25 de daño

console.log("¡Jugador recibe daño!");

vidaActual = vidaActual - 25;

// O la forma corta: vidaActual -= 25;

// Verificamos estado

console.log(`Vida actual de ${nombreJugador}: ${vidaActual}`); // 75

// El jugador encuentra una poción (+40 vida)

console.log("¡Jugador encuentra poción!");

vidaActual = vidaActual + 40;

// --- PASO 4: Problema de Lógica (Superar el máximo) ---

// Ahora tiene 115 de vida, pero el máximo es 100. ¿Cómo lo arreglamos?

// Usamos operador ternario (un 'if' en una línea) o Math.min (Avanzado simple)

// "Toma el menor valor entre vidaActual y VIDA\_MAXIMA"

vidaActual = Math.min(vidaActual, VIDA\_MAXIMA);

console.log(`Vida corregida: ${vidaActual}`); // 100

// --- PASO 5: Verificación de Estado (Operadores Lógicos) ---

// ¿Está vivo el jugador? (Vida mayor a 0)

const estaVivo = vidaActual > 0;

// ¿Está en peligro crítico? (Vida menor a 20 Y está vivo)

const enPeligro = vidaActual < 20 && estaVivo;

console.log("¿Está vivo?:", estaVivo);

console.log("¿En peligro?:", enPeligro);

console.log("Tipo de dato de vidaActual:", typeof vidaActual); // number

# Práctica Acompañada "Calculadora de Propinas"

Estás creando una app para un restaurante. Necesitas calcular cuánto debe pagar un cliente.

**Paso a paso:**

1. **Abre tu consola de desarrollador.**
2. **Crea una variable const** llamada montoBoleta y asígnale un valor, por ejemplo: 80.
3. **Crea una variable const** llamada porcentajePropina y asígnale el valor decimal de la propina, por ejemplo: 0.15 (para el 15%).
4. **Calcula el total de la propina.** Crea una variable const llamada totalPropina y guarda ahí el resultado de montoBoleta \* porcentajePropina.
5. **Calcula el gran total.** Crea una variable const llamada totalPagar y guarda la suma de montoBoleta + totalPropina.
6. **Muestra los resultados** de forma clara en la consola usando console.log() y template literals.

# Solución

// 1 y 2. Crear variables base

const montoBoleta = 80;

const porcentajePropina = 0.15;

// 3. Calcular la propina

const totalPropina = montoBoleta \* porcentajePropina;

// 4. Calcular el total

const totalPagar = montoBoleta + totalPropina;

// 5. Mostrar resultados

console.log('--- Resumen de la Cuenta ---');

console.log(`Monto de la Boleta: $${montoBoleta}`);

console.log(`Propina (15%): $${totalPropina}`);

console.log(`Total a Pagar: $${totalPagar}`);

# Práctica B: "Conversor de Temperatura"

**Misión:** Convertir una temperatura de grados Celsius a Fahrenheit.

**Paso a paso para el estudiante:**

1. La fórmula para convertir Celsius (C) a Fahrenheit (F) es: (C \* 9/5) + 32.
2. **Crea una variable const** llamada tempCelsius y asígnale un valor (ej. 25).
3. **Crea una variable const** llamada tempFahrenheit.
4. **Usa los operadores aritméticos** para aplicar la fórmula y guardar el resultado en tempFahrenheit. (Recuerda el orden de las operaciones: \* y / se hacen antes que +).
5. **Muestra el resultado** en la consola con un mensaje claro, por ejemplo: "25°C equivalen a 77°F."

# Ejercicio Independiente Aplicado

**Misión: "Calculadora de Perfil de Gimnasio"**

Vas a crear un script que almacene la información de un usuario y calcule su Índice de Masa Corporal (IMC).

**Instrucciones:**

1. **Almacena la información del usuario:**
   * Crea una variable const para el nombre del usuario.
   * Crea una variable let para la edad del usuario.
   * Crea una variable const para el peso del usuario en kilogramos (ej. 70).
   * Crea una variable const para la altura del usuario en metros (ej. 1.75).
2. **Calcula el IMC:**
   * La fórmula del IMC es: peso / (altura \* altura).
   * Puedes usar el operador \*\* para elevar al cuadrado: peso / (altura \*\* 2).
   * Guarda el resultado en una const llamada imc.
3. **Calcula el estado (lógica):**
   * Crea una variable let llamada esMayorDeEdad y usa un operador de comparación para que guarde true o false si la edad es mayor o igual a 18.
4. **Muestra el Reporte:**
   * Imprime en la consola un reporte completo usando console.log() y template literals, que se vea así:

--- Reporte de Usuario ---

Nombre: [Nombre del usuario]

Edad: [Edad del usuario]

¿Es mayor de edad?: [true/false]

--- Reporte de Salud ---

Peso: [Peso] kg

Altura: [Altura] m

IMC: [Resultado del IMC]

**Criterios de Logro Esperados:**

* Se diferencia correctamente entre let (para edad, que puede cambiar) y const (para valores fijos).
* El cálculo del IMC es matemáticamente correcto (uso de paréntesis u orden de operaciones).
* El cálculo de esMayorDeEdad usa un operador de comparación (>=).
* La salida en consola es limpia y usa template literals para mostrar las variables

# ¿Debería Saber Algo Más?

Has aprendido los "átomos", pero hay dos conceptos "moleculares" cruciales que surgen de ellos:

1. **Type Coercion (Coerción de Tipos):** Este es el concepto "oculto" más importante. JavaScript es un lenguaje de "tipado débil", lo que significa que intenta "ayudarte" convirtiendo tipos automáticamente. A veces es útil, pero a menudo es confuso.
   * '5' - 1 da 4 (JavaScript convierte el string '5' a un Number).
   * '5' + 1 da '51' (El operador + prefiere la concatenación de strings).
   * '5' \* '2' da 10 (Convierte ambos a Number).
   * Este comportamiento es la razón por la que **siempre usamos ===** y evitamos ==, para no ser víctimas de la coerción.
2. **Precedencia de Operadores:** Al igual que en matemáticas (PEMDAS), JavaScript tiene un orden estricto de operaciones. \* y / se ejecutan antes que + y -. Los operadores lógicos también tienen un orden (! primero, luego &&, luego ||). Cuando tengas dudas, **usa paréntesis ()** para forzar el orden que deseas. El código claro siempre es mejor que el código "inteligente".

Logotipo

El contenido generado por IA puede ser incorrecto.