



Conceptos Básicos JavaScript

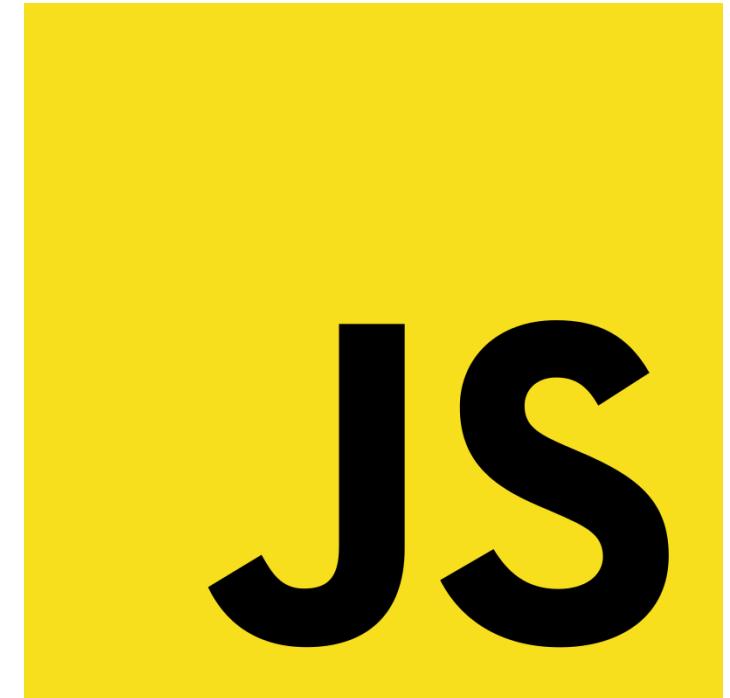
Centro de Servicios y Gestión Empresarial
SENA Regional Antioquia

Introducción a JS

¿Qué es JavaScript?



JavaScript (JS) es un **lenguaje de programación interpretado** utilizado principalmente en el desarrollo web para hacer que las páginas sean interactivas. Funciona en el navegador del usuario y permite manipular elementos HTML y CSS en tiempo real.



Características principales:



Multiparadigma (soporta programación orientada a objetos, funcional e imperativa).

Basado en eventos (ejecuta código en respuesta a acciones del usuario).

Asíncrono (maneja operaciones como peticiones a servidores sin bloquear el flujo principal).

Dinámico y débilmente tipado (no es necesario definir el tipo de datos al declarar una variable).

¿Para qué sirve JavaScript?



Interactividad en sitios web: Botones dinámicos, efectos, validaciones de formularios, etc.



Manipulación del DOM: Cambiar contenido, estilos y estructura HTML en tiempo real.



Peticiones a servidores: Cargar datos sin necesidad de recargar la página (AJAX, Fetch API).



Desarrollo de aplicaciones web completas: Frontend con React/Vue y Backend con Node.js.



Creación de juegos y animaciones: Con librerías como Three.js o Pixi.js.



Aplicaciones móviles y de escritorio: React Native (móvil), Electron.js (escritorio).

Como trabajar los script de JS?

Dependiendo del contexto y los requerimientos del proyecto, JavaScript se puede incluir en diferentes partes de un archivo HTML.

Colocar JavaScript en el <head>



Cuándo usarlo:

Cuando el script debe ejecutarse antes de que la página cargue completamente.

Para cargar configuraciones iniciales o definir funciones globales.

Para evitar bloqueos visuales cuando el código no depende de los elementos HTML.



Desventaja:

Si el script intenta manipular elementos del DOM antes de que se cargue el body, puede fallar.

Colocar JavaScript en el <head>



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo JS</title>
  <script>
    console.log("Este script se ejecuta antes de que el cuerpo (body) se
                cargue.");
  </script>
</head>
<body>
  <h1>Este es un ejemplo de JavaScript</h1>
</body>
</html>
```



index.html

Alvaro Perez N - Instructor

Colocar JavaScript al final del <body>



Cuándo usarlo:

Cuando el script necesita interactuar con el DOM.

Para mejorar el rendimiento (el HTML y CSS se cargan primero, evitando bloqueos).

Para scripts que no requieren ejecutarse de inmediato.



Ventajas:

La página carga más rápido.

Se asegura de que los elementos del DOM existan antes de ejecutar el código.

Colocar JavaScript al final del <body>



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo JS</title>
</head>
<body>
    <h1>Este es un ejemplo de JavaScript</h1>
    <button onclick="saludar()">Haz clic</button>
    <script>
        function saludar() {
            alert("Hola, JavaScript cargado al final del body");
        }
    </script>
</body>
</html>
```



Usar un archivo JavaScript externo (script.js)



Cuándo usarlo:

Para mantener un código más organizado y modular.

Cuando el mismo código JavaScript se usa en varias páginas.

Para facilitar la depuración y reutilización del código.



Ventajas:

Mantiene el código más limpio en el HTML

Facilita la reutilización en múltiples páginas

Mejora la organización y mantenimiento del proyecto.

Usar un archivo JavaScript externo (script.js)



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo JS</title>
</head>
<body>
  <h1>Este es un ejemplo de JavaScript</h1>
  <button onclick="mostrarMensaje()">Click aquí</button>
  <script src="script.js"></script>
</body>
</html>
```



index.html



```
function mostrarMensaje() {
  alert("¡Hola desde un archivo externo!");
}
```

Sintaxis

Comentarios

Ejemplo:

1. Se usan // para agregar notas rápidas o descripciones breves.

```
// Esto es un comentario de una línea
let nombre = "Juan"; // Variable que almacena el nombre
```

2. Se usan /* ... */ para explicaciones más largas.

```
/*
    Este script calcula la suma de dos números
    y muestra el resultado en la consola.
*/
let num1 = 10;
let num2 = 20;
console.log(num1 + num2); // Muestra 30 en la consola
```

Comentarios

Ejemplo:

3. **JSDoc** es el estándar de documentación en JavaScript. Se usa con `/** ... */` antes de funciones, clases o métodos.

```
/**
 * Calcula la suma de dos números.
 * @param {number} a - Primer número.
 * @param {number} b - Segundo número.
 * @returns {number} La suma de los dos números.
 */
function sumar(a, b) {
    return a + b;
}
```

<https://jsdoc.app/>

Variables y constantes

Variables y Constantes en JavaScript

En JavaScript, existen tres formas de declarar variables: **var**, **let** y **const**. Cada una tiene un comportamiento diferente en términos de alcance (scope), reasignación y seguridad.

Ejemplo:

```
let nombre = "Juan"; // Variable modificable
const PI = 3.1416; // Constante, no cambia
var edad = 25; // Variable modificable (antiguo)
```

Variabile var

1. **var** (*No recomendado*)

- **Ámbito (Scope)**: Funcional (visible en toda la función).
- Se puede **reasignar** y **redeclarar** dentro del mismo ámbito.
- No respeta el **bloque {}** (se filtra fuera de if, for, etc.).
- Problemas con **hoisting** (se mueve a la parte superior del código pero sin inicializar).

Ejemplo:

```
var nombre = "Carlos";
console.log(nombre); // Carlos
```

```
var nombre = "Ana"; // Se permite redeclarar
console.log(nombre); // Ana
```

Variable let

2. **let** (*Recomendada para variables cambiantes*)

- Ámbito (Scope): Bloque {} (solo existe dentro del bloque donde se declara).
- Se puede reasignar, pero no redeclarar en el mismo ámbito.
- Evita problemas de hoisting y filtrado de variables

Ejemplo:

```
let nombre = "Carlos";
console.log(nombre); // Carlos

nombre = "Ana"; // ✅ Se puede reasignar
console.log(nombre); // Ana

// let nombre = "Pedro"; ❌ Error: no se puede redeclarar en el mismo bloque
```

Variable let

2. **let** (*Recomendada para variables cambiantes*)

- Ámbito (Scope): Bloque {} (solo existe dentro del bloque donde se declara).
- Se puede reasignar, pero no redeclarar en el mismo ámbito.
- Evita problemas de hoisting y filtrado de variables

Ejemplo:

```
if (true) {  
    let edad = 30;  
    console.log(edad); // 30  
}  
console.log(edad); // ✗ Error: edad no está definida fuera del bloque
```

Constante const

3. **const** (*Recomendada para valores fijos*)

- Ámbito (Scope): Bloque {}. No permite reasignación.
- Obligatorio asignarle un valor al declararla.
- Los objetos y arreglos declarados con const pueden cambiar su contenido (pero no ser reasignados).

Ejemplo:

```
const PI = 3.1416;
console.log(PI); // 3.1416

// PI = 3.15; X Error: No se puede reasignar una constante
```

Variables y constantes

Característica	var	let	const
Ámbito	Función	Bloque	Bloque
Reasignable	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No
Redeclarable en el mismo ámbito	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No	<input type="checkbox"/> No
Hoisting	<input checked="" type="checkbox"/> Sí (pero sin inicialización)	<input type="checkbox"/> No	<input type="checkbox"/> No
Se recomienda?	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí (si el valor no cambia)

Tipos de Datos en JavaScript

Primitivos:

(Valores
inmutables y
almacenados
por valor)

Compuestos (o de referencia):

(Valores
mutables
almacenados
en memoria
por
referencia)

Especiales:

(Representan
ausencia o
falta de
valor)

Tipos de Datos Primitivos

Los **primitivos** son inmutables (no se pueden modificar) y se almacenan por **valor** en la memoria.

Ejemplo:

```
let nombre = "Ana"; // string
let edad = 30; // number
let estatura = 1.75; // number
let esEstudiante = true; // boolean
let saldoGrande = 9007199254740991n; // bigint
let id = Symbol("user_id"); // symbol
```

Tipos de Datos Compuestos (Referenciados)

Los compuestos almacenan referencias en memoria, lo que significa que si se modifica un objeto, el cambio se refleja en todas sus referencias.

Ejemplo:

```
// Objeto
let persona = { nombre: "Carlos", edad: 30 };
persona.edad = 31; //  Se puede modificar

// Arreglo
let numeros = [1, 2, 3, 4];
numeros.push(5); //  Se puede modificar el contenido
```

Tipos de Datos Compuestos (Referenciados)

Tipo	Descripción	Ejemplo
Object	Colección de datos clave-valor	{ nombre: "Carlos", edad: 25 }
Array	Lista ordenada de valores	[1, 2, 3]
Function	Bloque de código reutilizable	function sumar(a, b) { return a + b; }

Tipos de Datos Especiales

Estos representan ausencia de valor o errores de tipo.

- ◊ **undefined** → Se asigna automáticamente a variables no inicializadas.

```
let x;  
console.log(x); // undefined
```

- ◊ **null** → Se usa para indicar que una variable no tiene valor intencionalmente.

```
let y = null;  
console.log(y); // null
```

Tipos de Datos



Categoría	Tipo	Descripción
Primitivos	string	Cadenas de texto
	number	Números enteros y decimales
	boolean	true o false
	bigint	Números muy grandes
	symbol	Identificadores únicos
	undefined	Variable sin valor definido
	null	Ausencia intencional de valor
Compuestos	object	Conjunto de propiedades clave-valor
	array	Listas ordenadas de datos
	function	Código reutilizable
Especiales	undefined, null	Representan valores vacíos

Operadores

Operadores Aritméticos

Se utilizan para realizar operaciones matemáticas.

Operador	Descripción	Ejemplo	Resultado
+	Suma	$5 + 3$	8
-	Resta	$5 - 3$	2
*	Multiplicación	$5 * 3$	15
/	División	$10 / 2$	5
%	Módulo (resto)	$10 \% 3$	1
**	Potencia (exponente)	$2 ** 3$	8

Operadores de Asignación

Se usan para asignar valores a variables.

Operador	Ejemplo	Equivalente a
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 2$	$x = x - 2$
*=	$x *= 4$	$x = x * 4$
/=	$x /= 2$	$x = x / 2$
%=	$x \%= 3$	$x = x \% 3$
**=	$x **= 2$	$x = x ** 2$

Operadores de Comparación

Se usan para asignar valores a variables.

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	Igualdad (compara solo el valor)	<code>5 == "5"</code>	<code>true</code>
<code>===</code>	Estrictamente igual (compara valor y tipo)	<code>5 === "5"</code>	<code>false</code>
<code>!=</code>	Diferente (compara solo el valor)	<code>5 != "5"</code>	<code>false</code>
<code>!==</code>	Estrictamente diferente (compara valor y tipo)	<code>5 !== "5"</code>	<code>true</code>
<code>></code>	Mayor que	<code>10 > 5</code>	<code>true</code>
<code><</code>	Menor que	<code>10 < 5</code>	<code>false</code>
<code>>=</code>	Mayor o igual que	<code>5 >= 5</code>	<code>true</code>
<code><=</code>	Menor o igual que	<code>4 <= 5</code>	<code>true</code>

Operadores Lógicos

Se usan para combinar condiciones.

Operador	Descripción	Ejemplo	Resultado	Observacion
&&	AND (y)	true && false	false	&& devuelve true solo si ambas condiciones son verdaderas.
	OR (o)	true false	true	devuelve true si al menos una de las condiciones es verdadera.
!	NOT (negación)	!true	false	! invierte el valor booleano.

Operadores de Incremento y Decremento

Se usan para aumentar o disminuir valores numéricos en 1.

Operador	Ejemplo	Equivalente a
++	x++	$x = x + 1$
--	x--	$x = x - 1$

Entradas y Salidas

Input y Output en JavaScript

En JavaScript, podemos interactuar con el usuario de dos formas principales:

- 1 Input (Entrada de datos):** Capturar datos del usuario.

- 2 Output (Salida de datos):** Mostrar información en pantalla o en la consola.

Métodos de Entrada (Input) en JavaScript

Estos métodos permiten obtener datos del usuario.

- 💡 1. **prompt()** → Pedir datos al usuario
 - Abre una ventana emergente donde el usuario ingresa un valor.
 - Siempre devuelve un string (se debe convertir a número si es necesario).

- 💡 2. **confirm()** → Solicitar confirmación al usuario
 - Muestra un cuadro de diálogo con Aceptar y Cancelar.
 - Devuelve true si el usuario acepta y false si cancela.

Métodos de Salida (Output) en JavaScript

Estos métodos permiten mostrar el resultado de las acciones al usuario.

- 1. **console.log()** → Mostrar en la consola

Se usa para depurar código o mostrar resultados en la consola del navegador.

- 2. **alert()** → Mostrar mensaje emergente

Muestra una ventana emergente con un mensaje.

Ejemplo de Entradas / Salidas

```
// Ejemplo 01
```

```
let nombre = prompt("¿Cuál es tu nombre?");  
console.log(nombre); // Muestra el nombre ingresado
```

```
// Ejemplo 02
```

```
let edad = prompt("¿Cuántos años tienes?");  
edad = Number(edad); // Convertir a número  
alert("Tienes " + edad + " años.");
```

```
// Ejemplo 03
```

```
let respuesta = confirm("¿Deseas continuar?");  
console.log(respuesta); // true o false
```

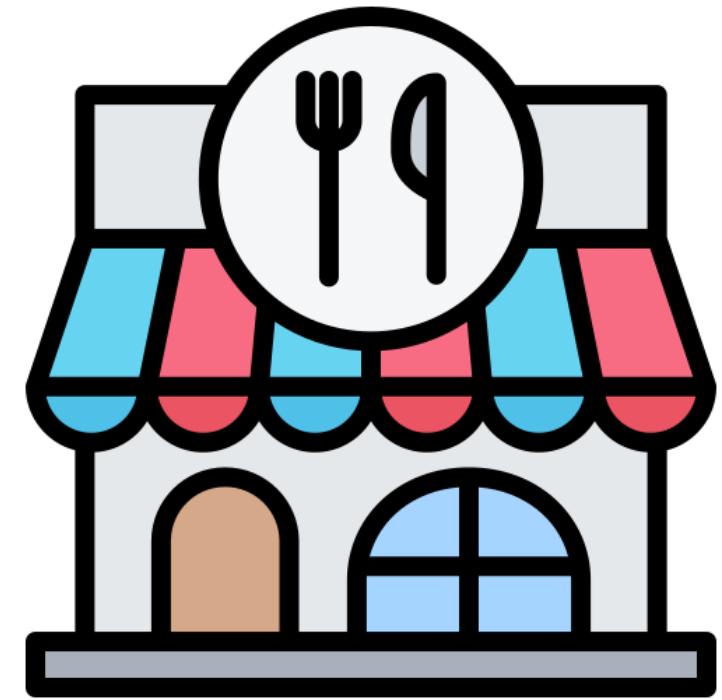
Ejercicios de Afianciamiento

Ejemplo No.01

Cálculo del Total a Pagar en un Restaurante

Escenario:

Un restaurante desea calcular el **total a pagar** por un cliente después de aplicar el **IVA (19%)** y una **propina del 10%**. El usuario ingresará el **precio de su consumo**, y el programa mostrará el total final.



Ejemplo No.01



```
// 1 Entrada de datos
let consumo = Number(prompt("Ingrese el valor total de su consumo:"));
const IVA = 0.19;
const PROPINA = 0.10;

// 2 Cálculos
let totalIVA = consumo * IVA;
let totalPropina = consumo * PROPINA;
let totalPagar = consumo + totalIVA + totalPropina;

// 3 Salida de datos
console.log("Consumo: $" + consumo);
console.log("IVA (19%): $" + totalIVA);
console.log("Propina (10%): $" + totalPropina);
console.log("Total a pagar: $" + totalPagar);
alert("El total a pagar es: $" + totalPagar);
```

Ejemplo No.02

Conversor de Unidades de Temperatura



Escenario:

Un usuario necesita convertir temperaturas de **Celsius a Fahrenheit** y de **Celsius a Kelvin**. El programa pedirá la temperatura en grados Celsius y devolverá los valores convertidos.



Ejemplo No.02



```
// 1 Entrada de datos
let celsius = Number(prompt("Ingrese la temperatura en grados Celsius:"));

// 2 Cálculo de conversiones
let fahrenheit = (celsius * 9/5) + 32;
let kelvin = celsius + 273.15;

// 3 Salida de datos
console.log(celsius + "°C equivalen a " + fahrenheit + "°F");
console.log(celsius + "°C equivalen a " + kelvin + "K");
alert(celsius + "°C equivalen a " + fahrenheit + "°F y " + kelvin + "K");
```

Ejemplo No.03

Cálculo del Costo de Llamadas Telefónicas ☎

💡 Escenario:

Una compañía de telefonía cobra las llamadas en función de su duración en minutos. El costo por minuto es \$0.50 USD y se requiere calcular el total a pagar por una llamada dada su duración en minutos.



Ejemplo No.03



```
// 1 Entrada de datos
let duracionMinutos = Number(prompt("Ingrese la duración de la llamada
en minutos:"));
const COSTO_POR_MINUTO = 0.50;

// 2 Cálculo del costo total
let costoTotal = duracionMinutos * COSTO_POR_MINUTO;

// 3 Salida de datos
console.log("Duración de la llamada: " + duracionMinutos + " minutos");
console.log("Costo por minuto: $" + COSTO_POR_MINUTO);
console.log("Total a pagar: $" + costoTotal);
alert("El total a pagar por la llamada es: $" + costoTotal);
```



G R A C I A S

Presentó: Alvaro Pérez Niño

Instructor Técnico

Correo: aperezn@sena.edu.co

<http://centrodesserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



www.sena.edu.co