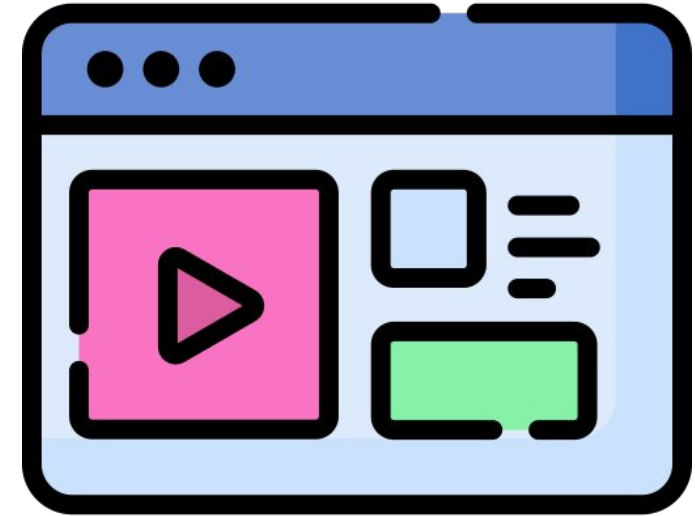


Desarrollo de Software 2024

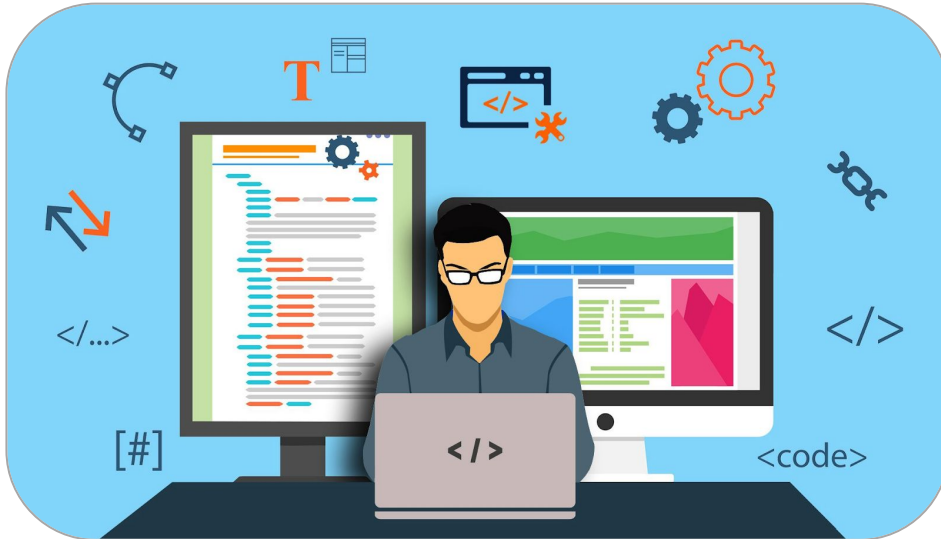
Contenido

Introducción a JavaScript

- ¿Qué es JavaScript y por qué es importante?
- Sintaxis básica de JavaScript.
- Variables, tipos de datos y operadores.
- Estructuras de control de flujo.



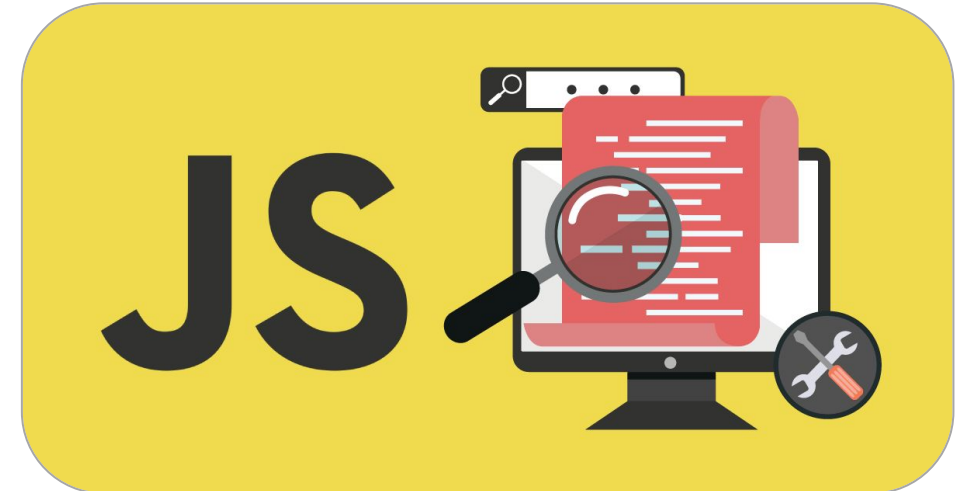
Objetivos de la Clase



- Entenderemos la importancia de los fundamentos de JavaScript en el desarrollo frontend.
- Aprenderemos los conceptos básicos de la sintaxis de JavaScript para construir una base sólida en programación.
- Exploraremos diversos tipos de datos y cómo operar con ellos eficientemente.
- Dominaremos el uso de estructuras de control de flujo para crear lógica y tomar decisiones en nuestras aplicaciones.
- Desarrollaremos la habilidad de crear y utilizar funciones, comprendiendo su alcance y la gestión de argumentos.
- Conoceremos cómo trabajar con arreglos y objetos para organizar y manipular datos de manera efectiva.
- Exploraremos conceptos avanzados, como funciones flecha, para mejorar nuestras habilidades de programación.
- Comprenderemos el papel esencial de npm (Node Package Manager) en la gestión de dependencias en proyectos frontend.

¿Qué es JavaScript (js)?

- JavaScript es un **lenguaje de programación** que se utiliza para conectar el frontend con el backend.
- JavaScript **funciona en el navegador** web del usuario y puede interactuar con el contenido de una página web.



Actividad: Javascript

- Utilizando la herramienta javascript-tester, ingresando al siguiente sitio:
<https://codebeautify.org/javascript-tester>
- Escribe una función llamada calcularPromedio que tome como parámetro un arreglo de números y calcule el promedio de esos números. La función debe retornar el promedio calculado:
- Instrucciones:
 1. Crea una función llamada calcularPromedio con un parámetro llamado **numeros**.
 2. Dentro de la función, declara una variable llamada suma y asígnale el valor inicial de 0.
 3. Utiliza la sentencia iterativa **for** para recorrer cada número en el arreglo **numeros**.
 4. En cada iteración, suma el número actual al valor de suma.
 5. Después de recorrer todos los números, calcula el promedio dividiendo suma entre la longitud del arreglo numeros.
 6. Retorna el promedio calculado.
 7. Entrega el código fuente resultante en la actividad del aula virtual.

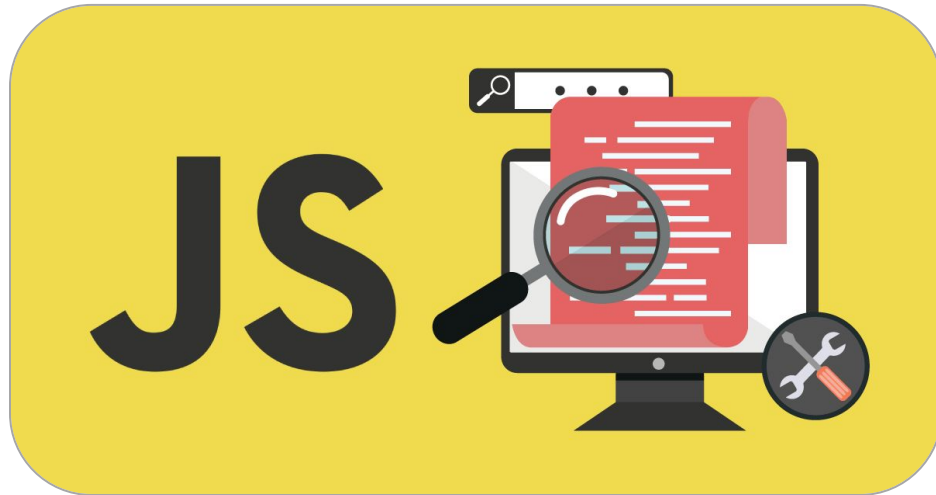
```
1  const calcularPromedio = (numeros) => {  
2    let suma = 0;  
3  
4    for (let i = 0; i < numeros.length; i++) {  
5      suma += numeros[i];  
6    }  
7  
8    let promedio = suma / numeros.length;  
9    return promedio;  
10 }  
11  
12 // Prueba de la función  
13 let numeros = [5, 8, 2, 10, 4];  
14 let resultado = calcularPromedio(numeros);  
15 console.log(resultado);
```

¿Por qué es importante JavaScript?

- Lenguaje de programación versátil y dinámico.
- Ampliamente utilizado en el desarrollo de aplicaciones web.
- Permite conectar frontend con backend y crear efectos visuales en el frontend.
- Utilizado tanto en el frontend como en el backend (con Node.js para crear aplicaciones escalables).



Importancia de JavaScript en el desarrollo Frontend

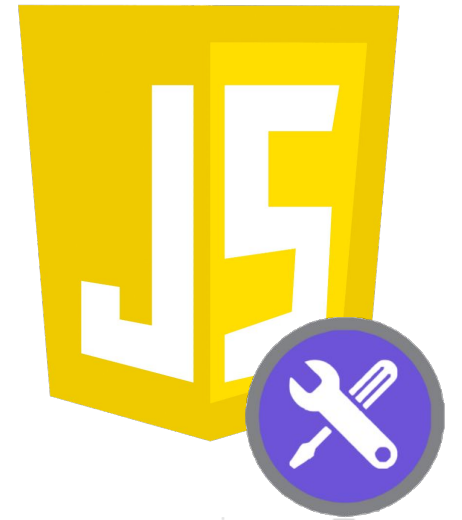


- **Interactividad:** Permite crear elementos interactivos y experiencias dinámicas en las páginas web.
- **Lenguaje del navegador:** Interpretado por los navegadores para ejecutar acciones en el lado del cliente.
- **Manipulación del DOM:** Facilita la modificación y actualización de elementos en tiempo real.
- **Mejora la UX:** Contribuye a interfaces reactivas y respuestas instantáneas.
- **Comunicación asíncrona:** Posibilita la carga y envío de datos al servidor sin recargar la página.
- **Fundamental en el frontend:** Imprescindible para el desarrollo moderno de aplicaciones web.

Herramientas y Entorno de Desarrollo para JavaScript

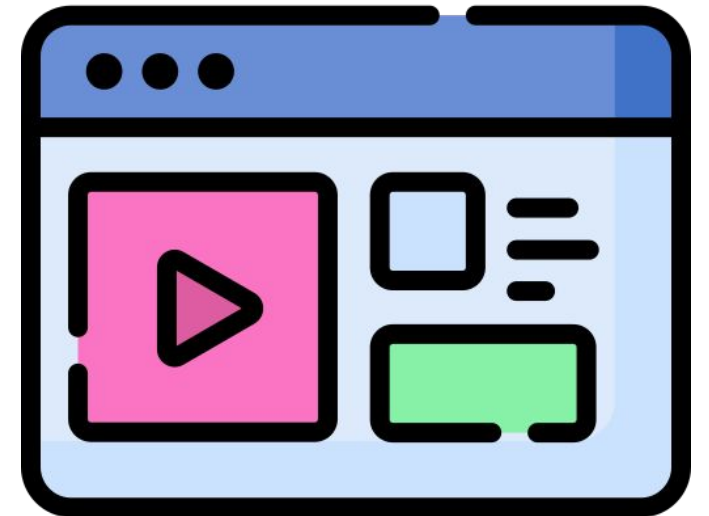
- **Editores de Código:** Utilizamos editores como Visual Studio Code para escribir y organizar el código.
- **Navegadores:** Chrome, Firefox y otros para probar y depurar aplicaciones web.
- **Consola del Navegador:** Inspeccionamos y depuramos el código con la consola integrada.
- **Node.js:** Entorno de ejecución que permite ejecutar JavaScript fuera del navegador.
- **npm (Node Package Manager):** Gestiona bibliotecas y paquetes de código para proyectos.
- **Control de Versiones:** Utilizamos herramientas como Git y plataformas como GitHub para colaborar y controlar cambios.
- **Herramientas de Construcción:** Webpack, Parcel, Gulp para optimizar y empaquetar recursos.
- **Frameworks y Bibliotecas:** React, Vue.js, Angular para agilizar el desarrollo frontend.

JavaScript



Node.js en el Frontend

- **¿Qué hace?:** Imaginemos que es como una caja de herramientas especial para los desarrolladores web.
- **Servidor de Desarrollo:** Es como un pequeño servidor que muestra nuestra página web en nuestro propio computador.
- **Actualizaciones Rápidas:** Nos ayuda a ver los cambios que hacemos instantáneamente, como magia.
- **Compilación:** También nos ayuda a arreglar nuestros archivos CSS y JavaScript automáticamente.
- **Imágenes Encantadas:** Puede hacer que nuestras imágenes se vuelvan más ligeras para que la página cargue más rápido.
- **Hace que Google nos Ame:** Node.js nos ayuda a crear sitios web que Google y otros motores de búsqueda adoran.
- **Amigo de los Desarrolladores:** Nos hace la vida más fácil al automatizar tareas aburridas.



¿Qué es Node.js?

- Node.js es un entorno de tiempo de ejecución de JavaScript de código abierto y multiplataforma.
- Características principales de Node.js:
 - Basado en el motor V8 de Google Chrome, lo que proporciona un rendimiento rápido y eficiente.
 - Utiliza un modelo de operaciones de entrada/salida sin bloqueo y orientado a eventos, lo que lo hace altamente escalable y adecuado para aplicaciones en tiempo real y de alto rendimiento.
 - Posee un ecosistema rico y activo de paquetes y módulos npm (Node Package Manager) que permiten a los desarrolladores reutilizar y compartir código fácilmente.



NodeJs: Instalación

Node.js® es un entorno de ejecución para JavaScript construido con V8, motor de JavaScript de Chrome.

Descargar para macOS

18.16.0 LTS

Recomendado para la mayoría

20.2.0 Actual

Últimas características

[Otras Descargas](#) | [Cambios](#) | [Documentación de la API](#)

[Otras Descargas](#) | [Cambios](#) | [Documentación de la API](#)

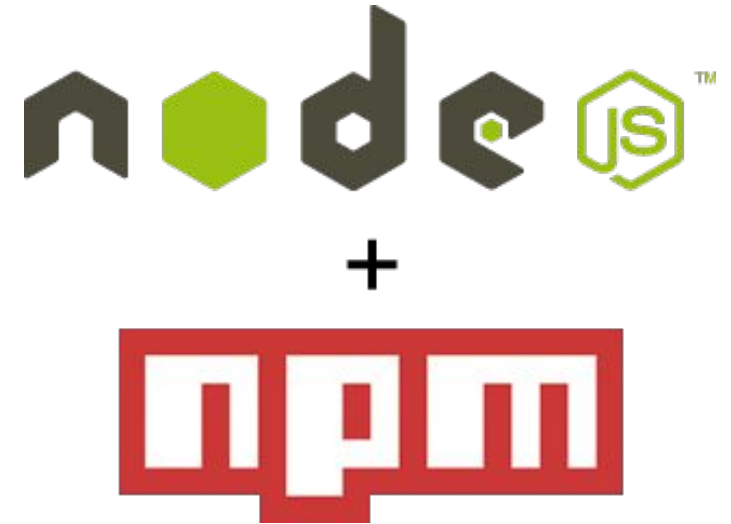
O eche un vistazo al [Programa de soporte a largo plazo \(LTS\)](#).

Instalar Nodejs

- Descargar archivo de instalación:
 - <https://nodejs.org/es>
- Seguir las instrucciones de instalación.

Introducción a npm

- **npm (Node Package Manager)** es el gestor de paquetes predeterminado para Node.js.
- Es una herramienta que nos **permite descargar, instalar y administrar fácilmente paquetes de código reutilizable** para nuestros proyectos de Node.js.
- Características principales:
 - **Repositorio de paquetes:** npm cuenta con un repositorio en línea que alberga miles de paquetes de código abierto. <https://www.npmjs.com/>
 - **Instalación sencilla:** Permite instalar paquetes específicos o dependencias de proyectos mediante comandos simples.
 - **Manejo de dependencias:** Administra automáticamente las dependencias de un proyecto y asegura que las versiones correctas estén instaladas.
 - **Scripts personalizados:** Permite ejecutar scripts personalizados definidos en el archivo "package.json" de un proyecto.
 - **Actualizaciones regulares:** npm se actualiza periódicamente para agregar nuevas funciones y mejoras.



¿Cómo usamos npm?

- Esta herramienta se utiliza con la línea de comandos.
- Para iniciar el uso de npm en un proyecto, debemos ejecutar el comando

> *npm init -y*

para generar un archivo "package.json" que almacena información sobre el proyecto y sus dependencias.

- Luego, puedes utilizar comandos como:

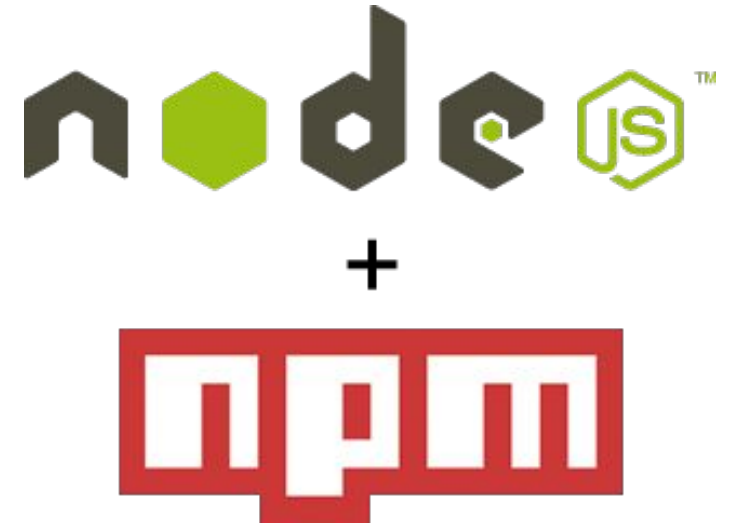
> *npm install*

para instalar las dependencias necesarias.

- Además, puedes buscar paquetes en el repositorio de npm y utilizar

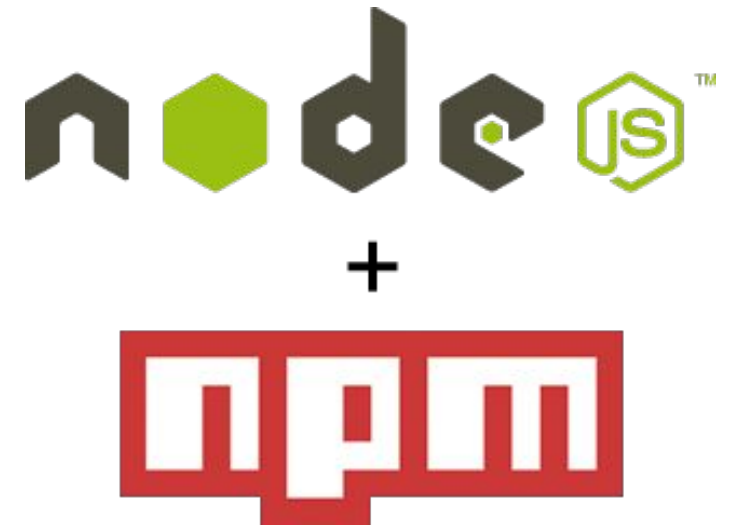
> *npm install <nombre_del_paquete>*

para descargar e instalarlos en tu proyecto.



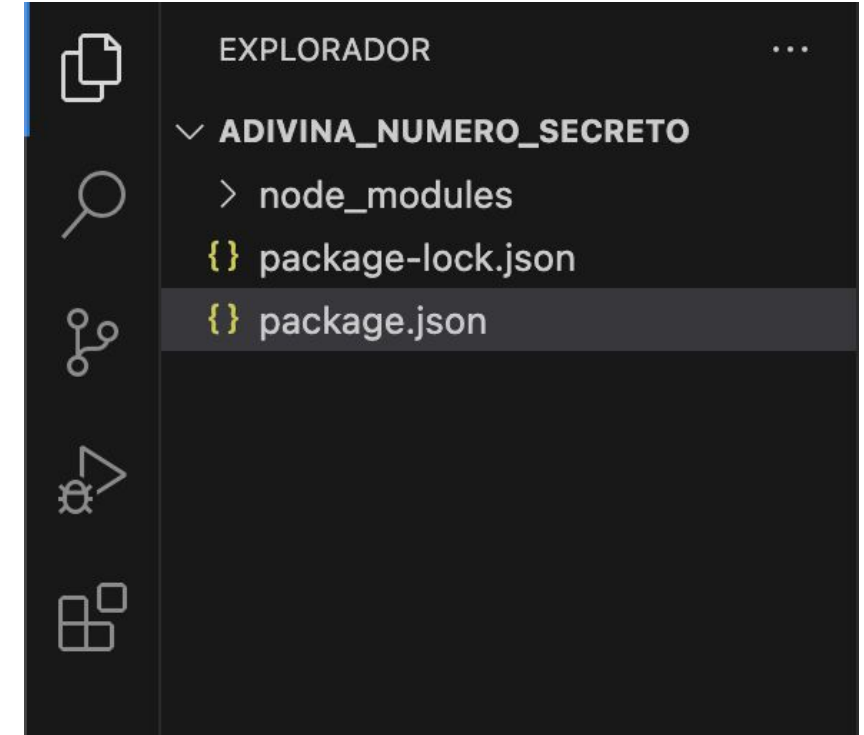
npm - package.json

```
{  
  "name": "nodejs_npm",  
  "version": "1.0.0",  
  "main": "app.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "description": ""  
}
```



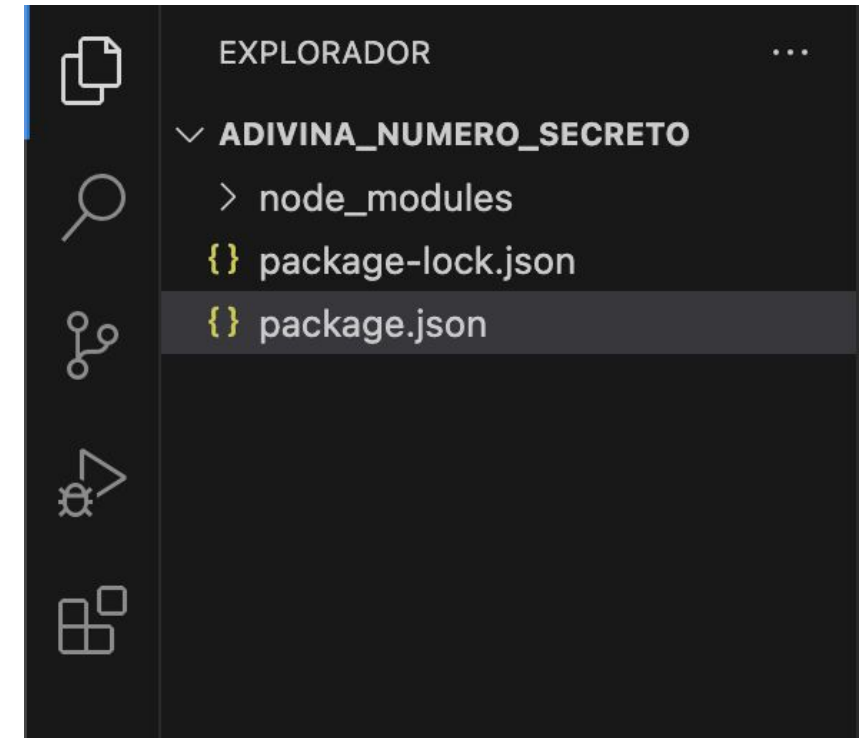
Instalación paquetes npm

- Como resultado de instalar un paquete npm sucede lo siguiente:
 - Genera una carpeta dentro de **node_modules** con el nombre del paquete. (La carpeta node_modules se genera la primera vez que instalamos un paquete o si ejecutar el comando npm install).
 - Genera un archivo **package-lock.json**, donde indica los paquetes que están en la carpeta node_modules.json.
 - Modifica el archivo **package.json** para agregar una dependencia.
- Tanto el archivo **package-lock.json** como la carpeta **node_modules** se generan automáticamente por npm, por lo tanto podemos borrarlas en cualquier momento y que se generen nuevamente con el comando npm install.



NPM - devDependencies y Herramientas Globales

- **devDependencies:**
 - Herramientas para desarrollo, como pinturas y pinceles para construir un proyecto.
 - Ejemplos: Compiladores, pruebas, empaquetado de archivos.
 - Agregamos con
> npm install <nombre-de-la-herramienta> --save-dev.
 - Estas herramientas ayudan a construir, pero no afectan a los usuarios finales.
- **Herramientas Globales:**
 - Herramientas que queremos usar en todos nuestros proyectos.
 - Instalamos con
> npm install -g <nombre-de-la-herramienta>.
 - Ejemplo: Herramientas para crear proyectos, como "create-react-app".
 - Accedemos a comandos directamente desde la terminal.
- **Cuándo usar cada una?**
 - Usa devDependencies para herramientas de construcción y prueba.
 - Usa herramientas globales para tareas comunes en múltiples proyectos.



Sintaxis básica de JavaScript

- Javascript utiliza **variables** para almacenar datos y operadores para realizar operaciones matemáticas y lógicas.
- Las declaraciones de control de flujo, como "if" y "else", permiten tomar decisiones y ejecutar diferentes acciones en función de las condiciones.



La **sintaxis** se refiere a las reglas y convenciones que se utilizan para escribir un código en un lenguaje de programación.

Ejemplo de código JavaScript

Aquí se muestra un ejemplo básico de código JavaScript para sumar dos números:

```
1  let numero1 = 5;  
2  let numero2 = 7;  
3  const resultado = numero1 + numero2;  
4  console.log(resultado);
```

Explicación:

Este código define dos variables, "numero1" y "numero2", suma sus valores y almacena el resultado en la variable "resultado". Finalmente, se imprime el resultado en la consola con el comando "console.log()".

Uso de Comentarios para Documentar el Código

- **Comentarios en el Código:**
 - Notas para explicar partes del código.
 - Aclaran el propósito y la lógica.
- **Beneficios:**
 - Facilitan la colaboración entre desarrolladores.
 - Ayudan a mantener el recuerdo de decisiones.
- **Cómo se usan:**
 - `//` para comentarios de una línea.
 - `/* ... */` para comentarios de varias líneas.
- **Dónde se aplican:**
 - Partes confusas o decisiones críticas.
 - Funciones y variables importantes.
- **Consejos para Recordar:**
 - Comentarios son para humanos, no máquinas.
 - Mantén comentarios útiles y relevantes.

```
// Ejemplo de comentario de una línea
let nombre = "Juan"; // Aquí estamos asignando el
valor "Juan" a la variable nombre

/*
    Ejemplo de comentario
    de varias líneas
*/
function saludar(nombre) {
    // Comentario para aclarar el proceso de saludo
    console.log("¡Hola, " + nombre + "!");
}
```

Variables / Constantes

- Una variable es un **contenedor** que se utiliza para **almacenar datos**. Se puede pensar en una variable como una caja con un nombre en la que puedes guardar un valor. Estos valores pueden ser números, cadenas de texto, objetos, arreglos u otros tipos de datos.

- Las **Variables** se crean con **let**:

```
let apellidoYNombre = "Juan Lopez";  
let cantidadDisponible = 15;  
cantidadDisponible = 30;
```

Buena práctica:

- Siempre utilizaremos **let** en vez de **var**.
- Usar camelCase para los nombres.

- Las **Constantes** se crean con **const** y son variables de sólo lectura (no se pueden modificar).

```
const PI = 3.141592;  
const EDAD = 23;
```

Buena práctica:

- Usar mayúscula para los nombres.

- Las variables creadas sin un valor aún contienen el valor **undefined**.

Javascript: Tipos de datos



Javascript: Operadores aritméticos

+	suma
-	resta
*	producto
%	módulo
++	incremento
--	decremento

```
// Operadores de suma
let suma = 5 + 3;    // 8

// Operadores de resta
let resta = 10 - 4;  // 6

// Operadores de multiplicación
let multiplicacion = 2 * 6;  // 12

// Operadores de división
let division = 20 / 5;  // 4

// Operador de módulo (resto de la división)
let modulo = 15 % 4;    // 3 (15 dividido por 4 da 3 con un residuo de 3)

// Operador de incremento
let contador = 10;
contador++;    // contador es ahora 11

// Operador de decremento
let numero = 8;
numero--;     // numero es ahora 7
```

Javascript: Operadores lógicos

&&	and
	or
!	not
?:	operador ternario

```
var edad = 25;
var tieneLicencia = true;
var esFeriado = true;
var tienePermiso = false;
var esNoche = false;

// Operador && (and): Ambas condiciones deben ser verdaderas para que se cumpla
if (edad >= 18 && tieneLicencia) {
  console.log("Puede conducir un automóvil.");
} else {
  console.log("No cumple los requisitos para conducir un automóvil.");
}

// Operador || (or): Al menos una de las condiciones debe ser verdadera para que se cumpla
if (esFeriado || tienePermiso) {
  console.log("Puede salir de vacaciones.");
} else {
  console.log("No puede salir de vacaciones.");
}

// Operador ! (not): Invierte el valor de verdad de una condición
if (!esNoche) {
  console.log("Es de día.");
} else {
  console.log("Es de noche.");
}

// Operador ternario
var mensaje = (edad >= 18) ? "Es mayor de edad" : "Es menor de edad";
console.log(mensaje);
```

Operadores de comparación

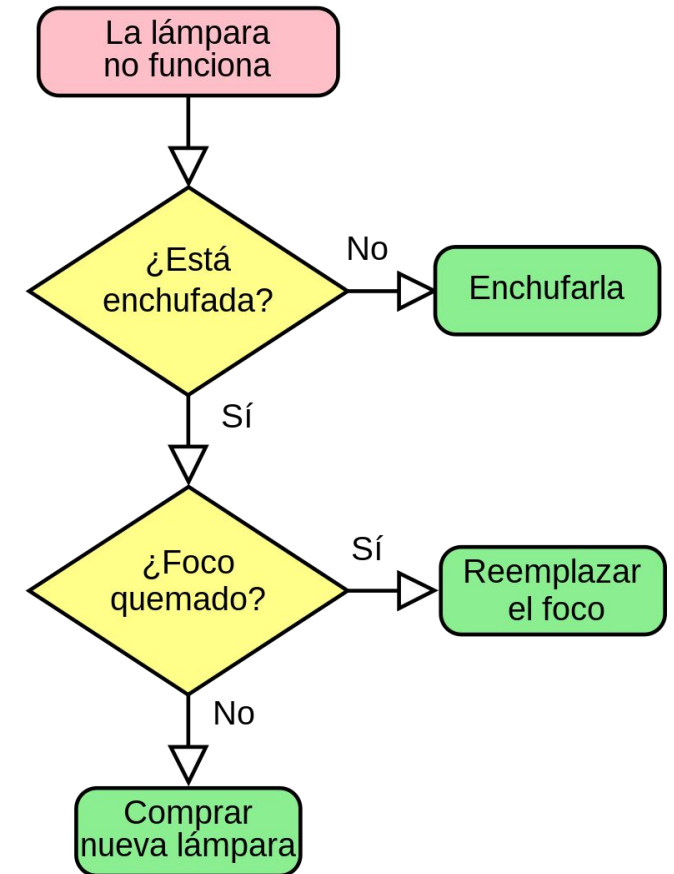
==	igualdad
!=	desigualdad
>	mayor
<	menor
>=	mayor o igual
<=	menor o igual
===	estrictamente iguales
!==	no estrictamente iguales

```
1  var numero1 = 10;
2  var numero2 = 5;
3
4  // Operador de igualdad (==)
5  console.log(numero1 == numero2); // Resultado: false
6
7  // Operador de desigualdad (!=)
8  console.log(numero1 != numero2); // Resultado: true
9
10 // Operador estrictamente igual (===)
11 console.log(numero1 === numero2); // Resultado: false
12
13 // Operador estrictamente desigual (!==)
14 console.log(numero1 !== numero2); // Resultado: true
15
16 // Operador mayor que (>)
17 console.log(numero1 > numero2); // Resultado: true
18
19 // Operador menor que (<)
20 console.log(numero1 < numero2); // Resultado: false
21
22 // Operador mayor o igual que (>=)
23 console.log(numero1 >= numero2); // Resultado: true
24
25 // Operador menor o igual que (<=)
26 console.log(numero1 <= numero2); // Resultado: false
```


Sentencias condicionales: if / if..else

```
if (condicion) {  
    código que se ejecuta si la condición es verdadera  
}
```

```
if (condicion) {  
    código que se ejecuta si la condición es verdadera  
} else {  
    código que se ejecuta si la condición es falsa  
}
```

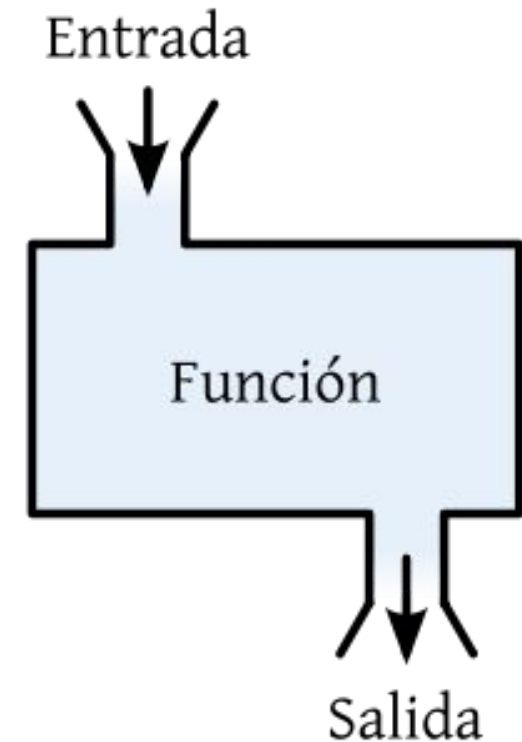


Javascript: Funciones

Una función es un bloque de código reutilizable que realiza una tarea específica cuando se invoca o se llama. Una función puede recibir datos de entrada, llamados argumentos o parámetros, realizar operaciones en esos datos y devolver un resultado.

```
function nombre(par1, par2, par3) {  
    /* código */  
}
```

```
function sumar(numero1,numero2){  
    return numero1+numero2;  
}
```



Javascript: Funciones

- Para invocar una función se utiliza el operador ()

```
let resultadoSuma = sumar(2,3);
```

- Se puede almacenar una función en una variable sin invocarla.

```
let c = sumar;  
d = c(5,6); /* d contendrá 11 */
```

- Funciones flecha (arrow function)

```
sumar = (a, b) => { return a + b; }  
  
sumar = (a, b) => a + b;
```

Ámbito de Variables: Alcance Global y Local

- **Ámbito de Variables:** Donde una variable es accesible y puede ser utilizada.

Alcance Global:

- Variables declaradas fuera de funciones.
- Accesibles en cualquier parte del código.

```
let globalVar = "Soy global";

function mostrarGlobal() {
  console.log(globalVar);
}

mostrarGlobal();
// Resultado: "Soy global"
```

Alcance Local:

- Variables declaradas dentro de funciones.
- Solo accesibles dentro de esa función.

```
function mostrarLocal() {
  let localVar = "Soy local";
  console.log(localVar);
}

mostrarLocal();
// Resultado: "Soy local"

console.log(localVar);
// Error: localVar no está definida aquí
```

Importante:

- Variables globales pueden afectar múltiples partes del código.
- Variables locales son más seguras y aisladas, evitando conflictos.

Javascript: Objetos

palabra clave nombre objeto

propiedades objeto {

```
var miObjeto = {  
  propiedad1 : valor,  
  propiedad2 : valor  
};
```

nombre propiedad

Nombre del objeto

```
const miAuto = {  
  color: "rojo",  
  marca: "Toyota",  
  modelo: "Prius",  
}
```

Propiedades

llave valor

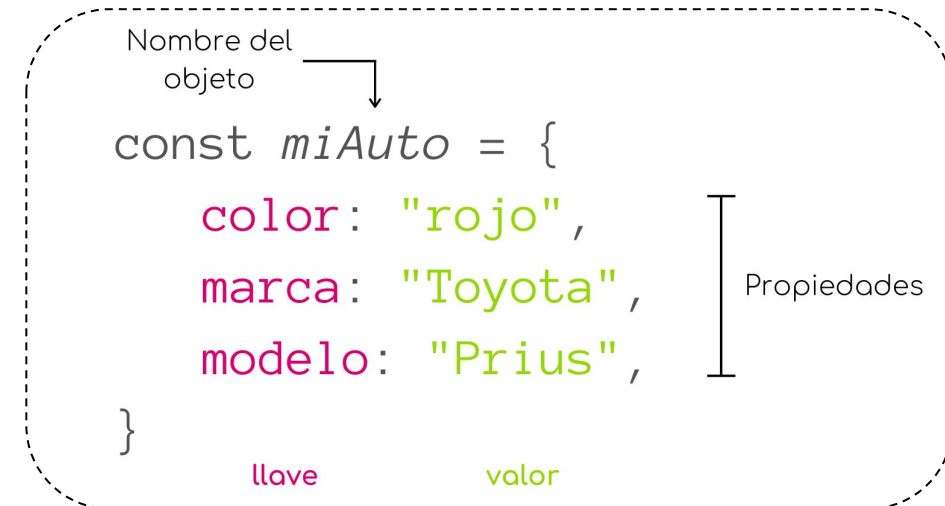
Javascript: Objetos

Se puede acceder a las propiedades mediante la **notación punto**:

```
let m = miAuto.marca;  
auto.marca = "Fiat";
```

O usando corchetes []

```
let m = miAuto["marca"];  
auto["marca"] = "Fiat";
```



Javascript: Arreglos

Sintaxis:

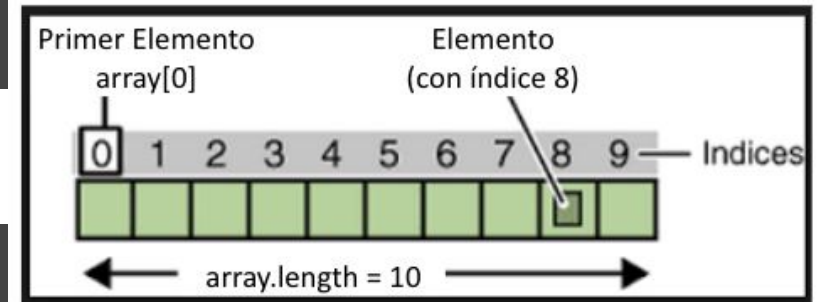
```
let nombre-array = [item1, item2, ...];
```

Ejemplo:

```
let colores = ["Verde", "Rojo", "Azul"];
```

Acceder a los elementos de un array:

```
let color1 = colores[0];  
colores[2] = "Amarillo";
```



Los **arreglos**, también conocidos como arrays, son estructuras de datos que permiten almacenar y organizar múltiples valores en una sola variable. Los arreglos son colecciones ordenadas, lo que significa que los elementos se almacenan en una secuencia y se accede a ellos mediante un índice numérico.

Trabajando con Arreglos (Arrays)

Creando Arreglos:

- Se crean con corchetes [].
- Pueden contener diferentes tipos de datos.

```
let colores = ["rojo", "verde", "azul"];  
let numeros = [1, 2, 3, 4, 5];  
let mezcla = ["hola", 42, true];
```

Acceso a Elementos:

- Se accede mediante índices (posición).
- Índices empiezan en 0.

```
console.log(colores[0]);  
// Resultado: "rojo"  
console.log(numeros[2]);  
// Resultado: 3
```

Longitud del Arreglo: Propiedad **length** para saber cuántos elementos hay.

```
console.log(mezcla.length);  
// Resultado: 3
```

Importante:

- Los arreglos nos ayudan a almacenar y manipular colecciones de datos.
- Accedemos a elementos usando índices, ¡recuerda que comienzan en 0!

Trabajo con Arreglos de Objetos

Creación de Arreglos de Objetos:

- Cada objeto tiene propiedades clave-valor.
- Pueden representar diferentes elementos con detalles.

```
let personas = [  
  { nombre: "Ana", edad: 25 },  
  { nombre: "Juan", edad: 30 },  
  { nombre: "María", edad: 28 }  
];
```

Acceso a Propiedades:

- Usamos la notación de punto o corchetes.

```
console.log(personas[0].nombre);  
// Resultado: "Ana"  
  
console.log(personas[1]["edad"]);  
// Resultado: 30
```

Iteración en Arreglos de Objetos:

Podemos usar bucles para recorrer y trabajar con cada objeto.

```
for (let persona of personas) {  
  console.log(persona.nombre + " tiene "  
+ persona.edad + " años.");  
}  
// Resultado:  
// Ana tiene 25 años.  
// Juan tiene 30 años.  
// María tiene 28 años.
```

Importante:

- Arreglos de objetos nos permiten almacenar y manejar múltiples datos con estructura.
- Accedemos a propiedades para obtener información específica de cada objeto.

Métodos de Arreglo: push, pop, shift, unshift

Funciones útiles para manipular arreglos.

push(): Agrega elementos al final del arreglo.

```
let frutas = ["manzana", "pera"];
frutas.push("banana");

// frutas: ["manzana", "pera", "banana"]
```

shift(): Elimina el primer elemento del arreglo.

```
let colores = ["rojo", "verde", "azul"];
colores.shift();

// colores: ["verde", "azul"]
```

pop(): Elimina el último elemento del arreglo.

```
let numeros = [1, 2, 3, 4, 5];
numeros.pop();

// numeros: [1, 2, 3, 4]
```

unshift(): Agrega elementos al principio del arreglo.

```
let letras = ["b", "c"];
letras.unshift("a");

// letras: ["a", "b", "c"]
```

Importante:

- Estos métodos nos permiten agregar, quitar y reorganizar elementos en arreglos.
- Pueden ser muy útiles para manejar datos de manera dinámica.

Bucles en JavaScript: while, for y for...of

- **Bucles:** Herramientas para repetir tareas en el código.

Bucle while:

- Repite mientras se cumpla una condición.
- Útil cuando no sabemos cuántas veces se repetirá.

```
let contador = 0;
while (contador < 5) {
  console.log("Número: " + contador);
  contador++;
}
// Resultado:
// Número: 0
// Número: 1
// Número: 2
// Número: 3
// Número: 4
```

Bucle for:

- Repite un número específico de veces.
- Más conciso para contadores.

```
for (let i = 0; i < 5; i++) {
  console.log("Iteración: " + i);
}
// Resultado:
// Iteración: 0
// Iteración: 1
// Iteración: 2
// Iteración: 3
// Iteración: 4
```

Bucle for...of:

- Itera sobre elementos de un arreglo.
- Perfecto para trabajar con listas de datos.

```
const colores = ["rojo", "verde", "azul"];
for (let color of colores) {
  console.log("Color: " + color);
}
// Resultado:
// Color: rojo
// Color: verde
// Color: azul
```

Instrucciones break y continue en Bucles

- **Instrucciones Break y Continue:** Herramientas para controlar bucles y saltar pasos.

Break:

- Detiene el bucle de inmediato.
- Útil cuando se alcanza una condición específica.

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    break;  
  }  
  console.log("Número: " + i);  
}  
// Resultado:  
// Número: 0  
// Número: 1  
// Número: 2
```

Continue:

- Salta a la siguiente iteración sin finalizar el bucle.
- Útil cuando queremos evitar ciertas condiciones.

```
for (let i = 0; i < 5; i++) {  
  if (i === 2) {  
    continue;  
  }  
  console.log("Número: " + i);  
}  
// Resultado:  
// Número: 0  
// Número: 1  
// Número: 3  
// Número: 4
```

Cuándo usar cada uno?:

- Utiliza break para salir del bucle en condiciones específicas.
- Usa continue para saltar iteraciones cuando sea necesario.

Javascript: JSON

JSON (se pronuncia Yeison) viene de JavaScript **O**bject **N**otation

- Es un formato de intercambio de datos entre aplicaciones independiente del lenguaje
- Es auto-descripto



```
{
  "clientes":[
    {"nombre":"Juan", "apellido":"Perez"},
    {"nombre":"Pedro", "apellido":"Garcia"},
    {"nombre":"Maria", "apellido":"Sanchez"}
  ]
}
```

Manejo de Errores y Uso de try...catch

- **Manejo de Errores:** Estrategias para lidiar con errores en el código.
- **¿Por qué es Importante?:**
 - Los errores son inevitables en la programación.
 - El manejo adecuado evita que los errores detengan la ejecución.

Uso de try...catch:

- **Bloque try:** Contiene el código propenso a errores.
- **Bloque catch:** Captura y maneja los errores.

```
try {  
  // Código propenso a errores  
  let resultado = 10 / undefined;  
} catch (error) {  
  console.error("Hubo un error: " + error.message);  
}
```

Ventajas del try...catch:

- Evita que el programa se detenga abruptamente.
- Proporciona información útil sobre los errores.

Importante:

- Aunque se manejen errores, es mejor prevenirlos con buena práctica.
- El manejo de errores debe ser específico para el contexto.

Cómo Ejecutar JavaScript con HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de JavaScript</title>
  <script>
    function saludar() {
      alert("¡Hola desde JavaScript!");
    }
  </script>
</head>
<body>
  <button onclick="saludar()">Haz clic</button>
</body>
</html>
```



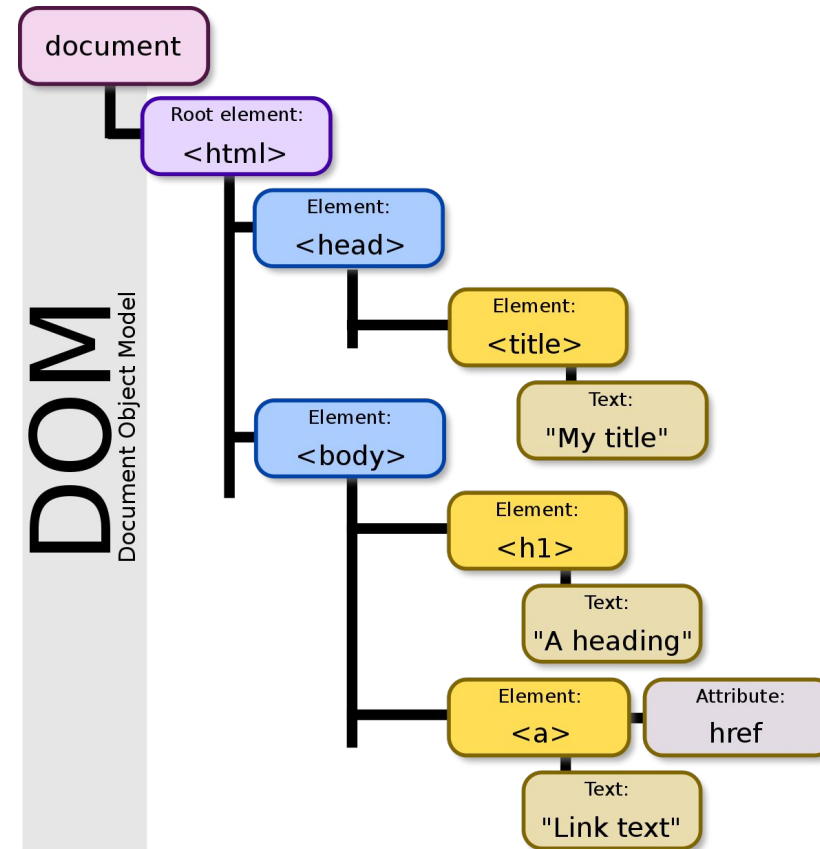
Importar un Archivo JavaScript en HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de Importación</title>
  <script src="mi-script.js"></script>
</head>
<body>
  <button onclick="saludar()">Haz
  clic</button>
</body>
</html>
```

```
function saludar() {
  alert("¡Hola desde mi-script.js!");
}
```


El Modelo de Objetos del Documento (DOM) y su Relación con HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo DOM</title>
</head>
<body>
  <h1 id="titulo">Título Original</h1>
  <p id="parrafo">Este es un párrafo.</p>
  <button onclick="cambiarTexto()">Cambiar
Título</button>
  <script>
    function cambiarTexto() {
      const tituloElemento =
document.getElementById("titulo");
      tituloElemento.textContent = "Nuevo
Título";
    }
  </script>
</body>
</html>
```



Bueno, Vamo a Codea!!!!



Calculadora Simple en HTML y JavaScript

Creemos un archivo HTML llamado **calculadora.html**:

```
<!DOCTYPE html>
<html>
<head>
  <title>Calculadora</title>
  <script src="calculadora.js"></script>
</head>
<body>
  <h1>Calculadora Simple</h1>
  <input type="number" id="num1" placeholder="Número 1">
  <input type="number" id="num2" placeholder="Número 2">
  <button onclick="sumar()">Sumar</button>
  <button onclick="restar()">Restar</button>
  <button onclick="multiplicar()">Multiplicar</button>
  <button onclick="dividir()">Dividir</button>
  <p id="resultado"></p>
</body>
</html>
```

Crea un archivo JavaScript llamado **calculadora.js** en la misma carpeta:

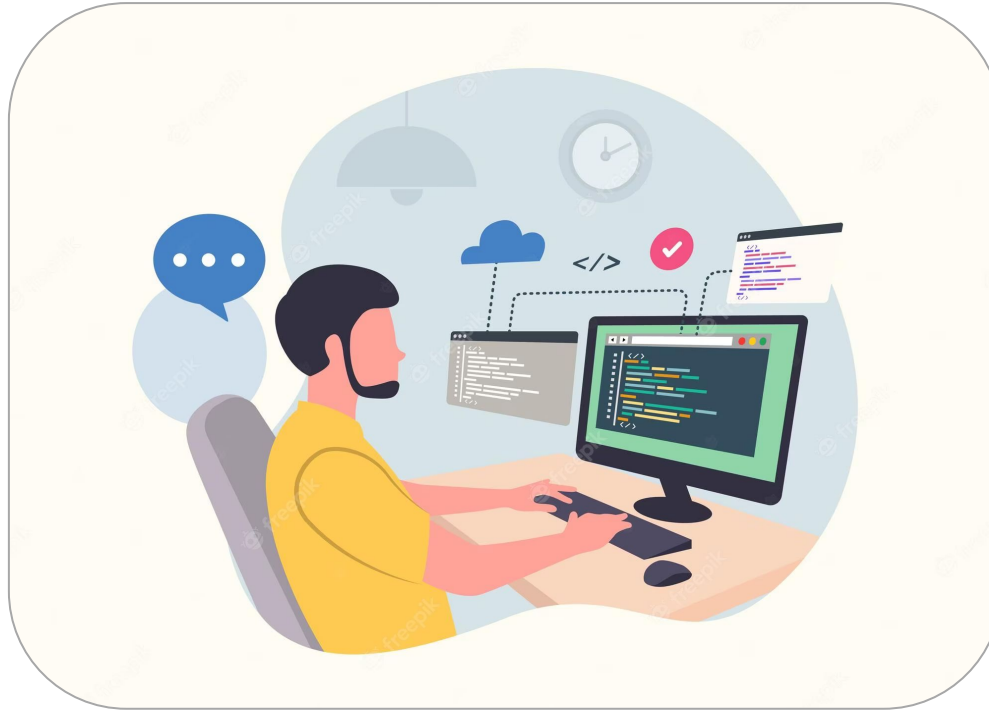
```
function sumar() {
  const num1 = parseFloat(document.getElementById("num1").value);
  const num2 = parseFloat(document.getElementById("num2").value);
  const resultado = num1 + num2;
  document.getElementById("resultado").textContent = "Resultado: " + resultado;
}

function restar() {
  const num1 = parseFloat(document.getElementById("num1").value);
  const num2 = parseFloat(document.getElementById("num2").value);
  const resultado = num1 - num2;
  document.getElementById("resultado").textContent = "Resultado: " + resultado;
}

function multiplicar() {
  const num1 = parseFloat(document.getElementById("num1").value);
  const num2 = parseFloat(document.getElementById("num2").value);
  const resultado = num1 * num2;
  document.getElementById("resultado").textContent = "Resultado: " + resultado;
}

function dividir() {
  const num1 = parseFloat(document.getElementById("num1").value);
  const num2 = parseFloat(document.getElementById("num2").value);
  if (num2 !== 0) {
    const resultado = num1 / num2;
    document.getElementById("resultado").textContent = "Resultado: " + resultado;
  } else {
    document.getElementById("resultado").textContent = "Error: División por cero";
  }
}
```

Actividad: Paso a Paso Javascript Frontend



MUCHAS GRACIAS

