Contenido

[JavaScript 1](#_Toc207460032)

[¿Qué hace JavaScript? 1](#_Toc207460033)

[Características clave: 1](#_Toc207460034)

[¿Dónde se usa? 1](#_Toc207460035)

[Importancia de su habilitación: 2](#_Toc207460036)

[Enlazar JavaScript 2](#_Toc207460037)

[Variables 3](#_Toc207460038)

[Prompt 4](#_Toc207460039)

[Operadores 5](#_Toc207460040)

[Operadores Aritméticos 5](#_Toc207460041)

[Operadores de Asignación 5](#_Toc207460042)

[Concatenación 6](#_Toc207460043)

[Condicionales 8](#_Toc207460044)

[Arrays 9](#_Toc207460045)

[Arrays asociativos 10](#_Toc207460046)

[Bucles e iteraciones 12](#_Toc207460047)

# JavaScript

JavaScript (JS) es un lenguaje de programación dinámico y multiparadigma que se usa para crear contenido web interactivo y dinámico, tanto en el navegador (lado del cliente) como en el servidor (lado del servidor, como con Node.js). Permite a los desarrolladores añadir funcionalidades como menús, animaciones, validación de formularios y actualización de contenido sin recargar la página.

## ¿Qué hace JavaScript?

* **Interactividad en la web:**

Es el lenguaje principal para añadir características interactivas a los sitios web, como juegos, efectos visuales, y formularios dinámicos.

* **Manipulación del DOM:**

Permite interactuar con el Document Object Model (DOM), que es la estructura de una página web, para modificar su contenido, estilo y estructura.

* **Actualización de contenido (**[**AJAX**](https://www.google.com/search?sca_esv=f2b4c39e3966b02c&sxsrf=AE3TifMSj6GQRLFOBNX1cAHppbKTKe715w%3A1756382901411&q=AJAX&sa=X&ved=2ahUKEwido7aXvK2PAxUoVTABHdeVGYIQxccNegUIlAEQAQ&mstk=AUtExfBuICkEOTR4NK8Qtxr1iNpoim6FfCQuJli37FAgR_yJyevQkf7cPahohdPlihIuJ630xGdQR9PKJ-a2stNHVFNfXkUafZljz_ixQcYkG24zzNhSouR3MrtxwaSj9ilwBijfc9BWSmMc8f3mR80qwi_NWs4Z_P_CLoQrq94FDAP2nCqdNfqGYbNwbeoI1E30nDIEqzPmQf0CPtjDUMak9IS2MlO-xDPKm11mbKzDWevk5YAGvLy0WYW1J4aDf1fLjUkQ5B5Plxk2vOkqOo7aovep&csui=3)**):**

Facilita el envío y recepción de datos al servidor en segundo plano, lo que permite actualizar partes de una página sin recargarla por completo, como en las redes sociales.

* **Desarrollo del lado del servidor:**

Con Node.js, JavaScript puede ejecutarse fuera del navegador, lo que permite crear aplicaciones web completas y del lado del servidor.

## Características clave:

* **Lenguaje de scripting:**

Se ejecuta para realizar tareas específicas, especialmente en el navegador.

* **Tipado dinámico:**

Los tipos de datos se pueden inferir en tiempo de ejecución, lo que le da flexibilidad.

* **Multiparadigma:**

Admite múltiples estilos de programación, incluyendo orientada a objetos y funcional.

## ¿Dónde se usa?

* **Navegadores web:**

Es el lenguaje de scripting más utilizado en los navegadores, con un motor JavaScript integrado en cada uno.

* **Servidores:**

Entornos como Node.js permiten usar JavaScript para el desarrollo de servidores y aplicaciones web.

* **Aplicaciones móviles:**

También se utiliza para el desarrollo de aplicaciones móviles, mejorando la experiencia del usuario.

## Importancia de su habilitación:

Para que las funcionalidades de una página web con JavaScript operen correctamente, este debe estar habilitado en el navegador. Si está deshabilitado, el contenido o la interactividad de la página pueden ser limitados o no estar disponibles.

# Enlazar JavaScript

Para enlazar JavaScript a HTML, utiliza la etiqueta <script> y dos métodos principales: incrustar el código JavaScript directamente dentro de la etiqueta, o usar el atributo src para referenciar un archivo .js externo, ubicando la etiqueta en la sección <head> o <body> de tu HTML para controlar su momento de carga.

**Método 1: Incrustar JavaScript en línea**

1. Abre tu archivo HTML en un editor de código.
2. Agrega la etiqueta <script> en la sección <head> o <body>.
3. Escribe tu código JavaScript dentro de las etiquetas <script> y </script>.

Código:

<!DOCTYPE html>  
 <html>  
 <head>  
 <title>Ejemplo de JavaScript en línea</title>  
 </head>  
 <body>  
 <h1>Mi Página Web</h1>

<script>

alert("¡Hola desde JavaScript en línea!"); // Esto se ejecutará al cargar la página  
 </script>

</body>  
 </html>

**Método 2: Enlazar un archivo JavaScript externo**

1. Crea un archivo nuevo con la extensión .js (por ejemplo, script.js) en la misma carpeta que tu archivo HTML.
2. Escribe tu código JavaScript en este archivo.

JavaScript:

*// En script.js*  
 console.log("¡Hola desde el archivo externo!");

1. En tu archivo HTML, usa la etiqueta <script> con el atributo src.

Código:

<!DOCTYPE html>  
 <html>  
 <head>  
 <title>Ejemplo de JavaScript externo</title>  
 </head>  
 <body>  
 <h1>Mi Página Web</h1>  
 <script src="script.js"></script>  
 </body>  
 </html>

**Ubicación de la etiqueta <script>:**

* **En el <head>:**

El código JavaScript se cargará y ejecutará antes de que el contenido del <body> sea procesado.

* **Al final del <body>:**

El código se ejecutará después de que el contenido HTML haya sido renderizado, lo cual es útil para scripts que interactúan con el DOM y evita retrasos en la carga de la página.

# Variables

En JavaScript, los tipos de variables más importantes son los tipos de datos primitivos como String, Number, BigInt, Boolean, Undefined, Null y Symbol, los cuales almacenan un solo valor. Aparte de estos, existe el tipo de dato no primitivo Object, que puede almacenar colecciones de datos como objetos y funciones.

**Tipos de datos primitivos**

Estos tipos de datos no son objetos y no tienen métodos asociados.

* String: Representa texto, como "Hola Mundo".
* Number: Almacena números, tanto enteros como decimales, como 10 o 3.14.
* BigInt: Se utiliza para números enteros muy grandes que no caben en el tipo Number.
* Boolean: Contiene valores lógicos, true (verdadero) o false (falso).
* Undefined: Indica que una variable ha sido declarada pero aún no tiene un valor asignado.
* Null: Representa la ausencia intencional de un valor, con un único valor null.
* Symbol: Es un valor único e inmutable, utilizado como clave en objetos para asegurar que no haya colisiones.

**Tipos de datos no primitivos**

* Object: Es el único tipo de dato no primitivo y se utiliza para agrupar información. Puede contener otras variables, como objetos o funciones.

**Para declarar variables en JavaScript, se utilizan las palabras clave var, let y const.**

* var:

Declara una variable con alcance de función, pero su uso es limitado en el código moderno.

* let:

Permite declarar variables con alcance de bloque y reasignarles un valor en el futuro.

* const:

Declara una constante, una variable cuyo valor no se puede reasignar después de su inicialización.

# Prompt

En JavaScript, prompt() es un método del objeto Window que muestra una ventana emergente (o diálogo) con un mensaje y un campo de entrada para que el usuario ingrese texto. Este método es útil para solicitar información sencilla del usuario, como su nombre o edad, y devuelve el texto ingresado como una cadena de texto o null si el usuario cancela la operación.

**Sintaxis:**

La sintaxis general del método prompt() es la siguiente:

let respuesta = prompt(mensaje, valorPorDefecto);

* mensaje: Un string opcional que se muestra al usuario en el diálogo.
* valorPorDefecto: Un string opcional que aparece en el campo de entrada del diálogo.

**Cómo funciona:**

1. **Muestra un diálogo:**

Al ejecutar prompt(), se abre una ventana modal que detiene la ejecución del script.

1. **Solicita información:**

El usuario ve el mensaje y un campo de texto con el valorPorDefecto (si se proporciona).

1. **Recopila datos:**

El usuario puede escribir su respuesta, usar el valor predeterminado o cancelar.

1. **Devuelve el resultado:**

Si el usuario hace clic en "Aceptar", la función devuelve el texto ingresado como una cadena de texto.

Si el usuario hace clic en "Cancelar" o presiona la tecla Esc, la función devuelve null.

**Ejemplo práctico:**

let nombre = prompt("Por favor, introduce tu nombre:", "Anónimo");  
alert("Hola, " + nombre + "!");

* Este código primero mostrará un prompt pidiendo el nombre, con "Anónimo" como valor por defecto.
* Luego, el nombre que el usuario introduzca se guardará en la variable nombre.
* Finalmente, se mostrará una alerta con el mensaje "Hola, [nombre introducido]".

# Operadores

## Operadores Aritméticos

Los operadores aritméticos en JavaScript realizan cálculos con números. Incluyen Adición (+), Resta (-), Multiplicación ()\*, División (/), Módulo (%) y Exponenciación (). También existen operadores unarios como el [incremento (++)](https://www.google.com/search?sca_esv=2fee5486871b890d&cs=0&sxsrf=AE3TifPOKJOu2_hPRZHj9r9BdPOlZhXZ-Q:1756469651190&q=incremento+(%2B%2B)&sa=X&ved=2ahUKEwjvjt2s_6-PAxWfRTABHVelATgQxccNegQIBRAB) y [decremento (--)](https://www.google.com/search?sca_esv=2fee5486871b890d&cs=0&sxsrf=AE3TifPOKJOu2_hPRZHj9r9BdPOlZhXZ-Q:1756469651190&q=decremento+(--)&sa=X&ved=2ahUKEwjvjt2s_6-PAxWfRTABHVelATgQxccNegQIBRAC), y operadores de asignación compuestos que combinan la operación con la asignación, como +=.

## Operadores de Asignación

Los operadores de asignación en JavaScript se usan para asignar valores a variables, siendo el más básico el signo igual (=). Además del operador básico, existen los operadores de asignación compuestos, que combinan una operación aritmética con la asignación, como += (suma y asigna) o -= (resta y asigna). También existen operadores lógicos de asignación como ||= (OR lógico y asigna) y &&= (AND lógico y asigna) para realizar asignaciones condicionales de forma concisa.

**Operadores de asignación básicos**

* =:

Es el operador de asignación simple. Asigna el valor del operando derecho a la variable del operando izquierdo.

**Ejemplo**: const x = 10; // Asigna el valor 10 a la constante x.

**Operadores de asignación compuestos**

Combinan una operación aritmética o bit a bit con una asignación, siendo más concisos que escribir la operación completa por separado.

* +=:

Suma el valor de la derecha a la variable y asigna el resultado a la variable. x += 5 es equivalente a x = x + 5.

* -=:

Resta el valor de la derecha de la variable y asigna el resultado. x -= 3 es equivalente a x = x - 3.

* \*=:

Multiplica el valor de la variable por el valor de la derecha y asigna el resultado. x \*= 2 es equivalente a x = x \* 2.

* /=:

Divide el valor de la variable por el valor de la derecha y asigna el resultado. x /= 4 es equivalente a x = x / 4.

* %=:

Calcula el módulo (resto) de la división del valor de la variable por el valor de la derecha y asigna el resultado. x %= 3 es equivalente a x = x % 3.

* **`** =`\*\*:

Realiza una exponenciación y asigna el resultado. x \*\*= 2 es equivalente a x = x \*\* 2.

**Operadores de asignación lógica (adiciones modernas)**

Permiten asignaciones condicionales de una forma más declarativa y concisa.

* ||=:

Asigna el valor del operando derecho a la variable si el valor actual de la variable es "falsy".

* &&=:

Asigna el valor del operando derecho a la variable solo si el valor actual de la variable es "truthy".

* ??=:

Asigna el valor del operando derecho a la variable si el valor actual de la variable es null o undefined.

# Concatenación

En JavaScript, la concatenación es el proceso de unir dos o más cadenas de texto usando el operador +, el método String.prototype.concat(), o las plantillas literales (backticks `` ${} ``) para combinar texto con variables y otros valores. Las plantillas literales son la forma más moderna y recomendada, especialmente al mezclar cadenas y variables.

1. **Con el operador + (el más común)**

Este operador une cadenas y otros tipos de datos que se pueden convertir a cadenas.

let saludo = "Hola";

let nombre = "Mundo";

let mensaje = saludo + " " + nombre + "!"; *// "Hola Mundo!"*

console.log(mensaje); // Muestra "Hola Mundo!" en la consola

1. **Con el método String.prototype.concat()**

Este método también une cadenas, pero se utiliza llamándolo en una cadena y pasando otras cadenas como argumentos.

let primeraParte = "JavaScript es ";

let segundaParte = "divertido";

let resultado = primeraParte.concat(segundaParte); *// "JavaScript es divertido"*

console.log(resultado); // Muestra "JavaScript es divertido"

1. **Con las plantillas literales (backticks `` ` ``)**

Introducidas en ES6, son la forma más flexible, permitiendo insertar variables y expresiones directamente dentro de la cadena.

let pelicula = "Avatar";

let director = "James Cameron";

let mensajePersonalizado = `Mi película preferida es ${pelicula} de

${director}`;

console.log(mensajePersonalizado); // Muestra "Mi película preferida es Avatar de James Cameron"

Consideraciones adicionales

* **Espacios:**

Al usar el operador +, debes añadir espacios manualmente si los necesitas, como en saludo + " " + nombre.

* **Combinación de tipos:**

Si intentas concatenar un número y una cadena, el número se convertirá en una cadena.

* **Recomendación:**

Para la mayoría de los casos, especialmente cuando se mezclan variables, las plantillas literales con ` `` `` son la opción más clara y elegante.

# Condicionales

En JavaScript, los condicionales como if, else if, else, switch y el operador ternario (? :) te permiten controlar qué bloques de código se ejecutan basándose en la veracidad de una condición. Puedes usar operadores de comparación (===, <, >=) y lógicos (&&, ||, !) para crear condiciones más complejas y decidir el flujo de tu programa.

* **Sentencia if**

Ejecuta un bloque de código solo si una condición es verdadera.

let edad = 18;

if (edad >= 18) {

console.log("Es mayor de edad."); *// Este mensaje se mostrará*

}

* **Sentencia if...else**

Ejecuta un bloque de código si la condición es verdadera y otro bloque si es falsa.

let nota = 4;  
if (nota < 5) {  
 console.log("Suspenso");  
} else {  
 console.log("Aprobado"); *// Este mensaje se mostrará*

}

* **Sentencia if...else if...else**

Permite evaluar múltiples condiciones consecutivamente. Si la primera condición es falsa, se pasa a la siguiente.

let temperatura = 25;  
if (temperatura < 0) {  
 console.log("Hace frío.");  
} else if (temperatura < 15) {  
 console.log("Hace fresco.");  
} else {  
 console.log("Hace calor."); // Este mensaje se mostrará  
}

* **Sentencia switch**

Evalúa un valor contra múltiples casos posibles y ejecuta el bloque de código correspondiente al caso que coincida.

let dia = "lunes";  
switch (dia) {  
 case "lunes":  
 console.log("Iniciando la semana.");  
 break; *// Termina la ejecución del switch*

case "viernes":  
 console.log("¡Casi fin de semana!");  
 break;

default: // *Se ejecuta si ningún caso coincide*

console.log("Es un día normal.");

}

* **Operador condicional (ternario ? :)**

Es una forma concisa de escribir un if...else para asignar valores o ejecutar expresiones.

let saldo = 1000;  
let mensaje = saldo > 500 ? "Tiene saldo suficiente" : "Saldo insuficiente";  
console.log(mensaje); // "Tiene saldo suficiente" se mostrará

# Arrays

En JavaScript, un array es un objeto especial similar a una lista para almacenar colecciones de datos. Puedes declarar un array usando corchetes [], y cada elemento tiene un índice numérico que empieza en 0 para acceder a él. Los arrays en JavaScript son dinámicos, permitiéndote agregar o eliminar elementos, cambiar su longitud y almacenar diferentes tipos de datos en una misma colección, aunque se recomienda mantener la coherencia del tipo de dato.

**¿Para qué sirven los arrays?**

* **Agrupar datos:**

Permiten organizar y almacenar múltiples valores relacionados dentro de una sola variable, simplificando el manejo de conjuntos de datos.

* **Facilitar operaciones:**

Ofrecen métodos incorporados para manipular y transformar los datos, como unirlos, invertirlos, ordenarlos o filtrar elementos.

* **Recorrer colecciones:**

Se utilizan comúnmente con bucles (forEach, for) para procesar cada uno de los elementos de la colección de manera eficiente.

**¿Cómo crear un array?**

Se usan corchetes [] para declarar un array, separando los elementos con comas.

*// Array de números*

let numeros = [1, 2, 3, 4, 5];

/*/ Array de cadenas*

let nombres = ["Ana", "Luis", "Sofía"];

// *Array con diferentes tipos de datos (no es una buena práctica)*

*le*t mixto = [10, "texto", true, { nombre: "Juan" }];

**¿Cómo acceder a los elementos?**

Utiliza el índice del elemento (que empieza en 0) dentro de corchetes.

let frutas = ["manzana", "banana", "naranja"];

console.log(frutas[0]); *// Salida: manzana*

console.log(frutas[1]); // Salida: banana

**Propiedad length**

La propiedad length indica el número de elementos en el array.

let colores = ["rojo", "verde", "azul"];

console.log(colores.length); // Salida: 3

**Métodos comunes de arrays**

JavaScript proporciona varios métodos para trabajar con arrays:

* push(...items)(): Agrega elementos al final del array.
* pop()(): Elimina el último elemento del array.
* shift()(): Elimina el primer elemento del array.
* unshift(...items)(): Agrega elementos al principio del array.
* forEach(callback)(): Ejecuta una función para cada elemento del array.

# Arrays asociativos

En JavaScript, no existen los arrays asociativos como tipo de dato independiente; en su lugar, se utilizan los objetos para almacenar pares de clave-valor. Estos objetos funcionan de manera similar a un mapa o tabla de consulta, donde se asocia una clave (una cadena de texto) con un valor. A diferencia de los arrays tradicionales, los objetos no tienen una propiedad length y se accede a sus elementos mediante nombres de propiedad definidos por el usuario, no por índices numéricos.

**Cómo crear un objeto (array asociativo en JavaScript)**

Puedes crear un objeto de dos maneras principales:

1. **Literal de objeto:** Definir las propiedades y valores directamente dentro de llaves {}.

const persona = {

nombre: "Jose",  
 edad: 30,  
 ciudad: "Bogotá"

};

1. Usando la palabra clave new Object() o creando una variable y añadiendo propiedades después :

const producto = {}; *// Creación vacía*

producto.nombre = "Camiseta";

producto.precio = 25.99;

**Acceso y manipulación de los datos**

Puedes acceder a los valores de un objeto usando la notación de punto (.) o la notación de corchetes ([]).

**Notación de punto:**

console.log(persona.nombre); // Salida: Jose

**Notación de corchetes:**

Útil cuando la clave es una variable o contiene caracteres especiales.

const clave = "ciudad";

console.log(persona[clave]); // Salida: Bogotá

**Diferencias clave con los arrays numéricos**

* **Índices:**

Los arrays tienen índices numéricos (0, 1, 2...), mientras que los objetos usan claves de texto.

* **Propiedad length:**

Los arrays tienen una propiedad length que indica su tamaño; los objetos no.

* **Iteración:**

No se pueden recorrer los objetos con un bucle for tradicional. Para recorrerlos, se usan métodos como Object.keys() o for...in para obtener las claves y luego acceder a sus valores.

# Bucles e iteraciones

En JavaScript, los bucles (loops) son estructuras que ejecutan un bloque de código repetidamente, mientras que una iteración es cada una de esas repeticiones. Los bucles for, while y do...while son los fundamentales para la ejecución repetitiva, mientras que for...in y for...of son específicos para iterar sobre propiedades de objetos y elementos de arrays, respectivamente.

**Tipos de Bucles**

* **for:** Ideal cuando se conoce la cantidad de repeticiones de antemano. Incluye una expresión inicial, una condición de continuación y una expresión de actualización.

for (let i = 0; i < 5; i++) {

console.log(i); *// Imprime 0, 1, 2, 3, 4*

}

* **while:** Repite un bloque de código mientras una condición sea verdadera. La condición se evalúa antes de cada iteración.

let i = 0;  
while (i < 3) {  
 console.log(i); *// Imprime 0, 1, 2*  
 i++;

}

* **do...while:** Similar al while, pero garantiza que el código dentro del bucle se ejecute al menos una vez, ya que la condición se evalúa al final.

let i = 0;

do {  
 console.log(i); *// Imprime 0*  
 i++;

} while (i < 3);

* **for...of:** Se utiliza para iterar sobre los valores de objetos iterables, como arrays.

const frutas = ["manzana", "banana", "cereza"];

for (const fruta of frutas) {

console.log(fruta); *// Imprime manzana, banana, cereza*  
 }

* **for...in:** Itera sobre los nombres (claves) de las propiedades enumerables de un objeto.

const persona = { nombre: "Ana", edad: 30 };

for (const clave in persona) {

console.log(clave + ": " + persona[clave]); *// Imprime "nombre: Ana", "edad: 30"*  
 }

**Conceptos Clave**

* **Condición**:

La expresión que determina si el bucle debe continuar o terminar.

* **Contador (o Variable de Control)**:

Una variable que se inicializa y se modifica en cada iteración para llevar un seguimiento del progreso y, a menudo, para controlar la condición del bucle.

* **Incremento/Decremento**:

La operación (como i++ o i--) que cambia el valor del contador en cada paso, asegurando que el bucle eventualmente pueda terminar.

* **Bucles Infinitos**:

Una situación en la que la condición del bucle nunca se vuelve falsa, lo que provoca que el programa se quede atascado ejecutándolo eternamente.

**Uso de continue y break**

* **continue:** Salta la iteración actual y pasa a la siguiente repetición del bucle.
* **break:** Termina completamente el bucle.

# Funciones

Las funciones en JavaScript son bloques de código reutilizable que realizan una tarea específica cuando son llamadas o invocadas. Se definen con la palabra clave function, un nombre, paréntesis para parámetros (entradas) y llaves que contienen el código a ejecutar. Permiten la organización, modularidad y eficiencia del código al evitar la repetición y encapsular funcionalidades, siendo un concepto fundamental en la programación.

**¿Por qué usar funciones?**

* **Reutilización de código:**

Escribe un bloque de código una vez y llámalo múltiples veces en diferentes partes del script, evitando la duplicación.

* **Organización y legibilidad:**

Separan el código en unidades lógicas, haciendo el script más limpio, fácil de mantener y entender.

* **Modularidad:**

Permiten dividir un programa complejo en partes más pequeñas y manejables, lo que mejora el diseño.

**Sintaxis básica**

function nombreDeLaFuncion(parametro1, parametro2) {  
 *// Código a ejecutar*

return valor; /*/ Opcional: devuelve un valor*

}

* function: La palabra clave que indica la definición de una función.
* nombreDeLaFuncion: El identificador que usarás para llamar la función.
* parametro1, parametro2: Variables que reciben los valores (argumentos) al llamar la función.
* { ... }: El cuerpo de la función, donde se encuentra el código a ejecutar.
* return valor: La instrucción que devuelve un resultado desde la función.

**Ejemplo**

*// Definición de la función*

*f*unction saludar(nombre) {  
 return "Hola, " + nombre + "!";

}  
  
// *Llamada (invocación) de la función*

*let* mensaje = saludar("Mundo");

console.log(mensaje); // Salida: Hola, Mundo!

**Tipos principales de funciones**

1. **Funciones declarativas:** Se definen usando la palabra function seguida del nombre de la función.
2. **Funciones de expresión:** Se asignan a una variable. Pueden ser anónimas o con nombre, y a menudo se usan las funciones de flecha (=>) para una sintaxis más concisa.
3. **Funciones de flecha (Arrow Functions):** Una sintaxis más moderna y corta para definir funciones, especialmente útil en funciones de expresión.

# POO (Programación Orientada a Objetos)

La Programación Orientada a Objetos (POO) en JavaScript consiste en organizar el código en objetos que representan elementos del mundo real, combinando datos y comportamiento (métodos) para crear un software más modular, reutilizable y fácil de mantener. Aunque JavaScript se basa internamente en un modelo de herencia prototípica, las clases introducidas en ECMAScript 6 (ES6) proporcionan una sintaxis más familiar para la POO, similar a otros lenguajes basados en clases. Los principios clave de la POO, como la encapsulación, la herencia y el polimorfismo, se aplican en JavaScript para lograr un diseño de software más robusto y organizado.

Principios Fundamentales de la POO en JavaScript

* [**Clases**](https://www.google.com/search?sca_esv=1f08cea1075aad4d&cs=0&sxsrf=AE3TifNCnc8PI4sKF10OxDIJoT4Z5dXh-A%3A1756667424438&q=Clases&sa=X&ved=2ahUKEwi08ZmO4LWPAxVbQzABHVztEmwQxccNegQIDxAB&mstk=AUtExfCrosXwknB3yOHGSOQSFuFRSj8sdkbi8wZOZ49RgOWlLVOn4rOAxrHxzOwY-EU2ueHWv6jThiMwUh90IAS82g4-8j2A3vEzIZgwArqGlX6dtHW-Ad2r8RjnuXqhOGBGcRVRPjcAMbBMEiJ_UDV5I6mVpOqcoMKvLG8jb7FYPUicQos&csui=3)**y**[**Objetos**](https://www.google.com/search?sca_esv=1f08cea1075aad4d&cs=0&sxsrf=AE3TifNCnc8PI4sKF10OxDIJoT4Z5dXh-A%3A1756667424438&q=Objetos&sa=X&ved=2ahUKEwi08ZmO4LWPAxVbQzABHVztEmwQxccNegQIDxAC&mstk=AUtExfCrosXwknB3yOHGSOQSFuFRSj8sdkbi8wZOZ49RgOWlLVOn4rOAxrHxzOwY-EU2ueHWv6jThiMwUh90IAS82g4-8j2A3vEzIZgwArqGlX6dtHW-Ad2r8RjnuXqhOGBGcRVRPjcAMbBMEiJ_UDV5I6mVpOqcoMKvLG8jb7FYPUicQos&csui=3)**:**

Una clase es una plantilla (un concepto abstracto) para crear objetos, y los objetos son instancias concretas de esa clase, conteniendo sus propias propiedades (datos) y métodos (funciones).

* [**Encapsulamiento**](https://www.google.com/search?sca_esv=1f08cea1075aad4d&cs=0&sxsrf=AE3TifNCnc8PI4sKF10OxDIJoT4Z5dXh-A%3A1756667424438&q=Encapsulamiento&sa=X&ved=2ahUKEwi08ZmO4LWPAxVbQzABHVztEmwQxccNegQIERAB&mstk=AUtExfCrosXwknB3yOHGSOQSFuFRSj8sdkbi8wZOZ49RgOWlLVOn4rOAxrHxzOwY-EU2ueHWv6jThiMwUh90IAS82g4-8j2A3vEzIZgwArqGlX6dtHW-Ad2r8RjnuXqhOGBGcRVRPjcAMbBMEiJ_UDV5I6mVpOqcoMKvLG8jb7FYPUicQos&csui=3)**:**

Agrupar los datos (propiedades) y las funciones (métodos) que operan sobre ellos dentro de un objeto.

* [**Herencia**](https://www.google.com/search?sca_esv=1f08cea1075aad4d&cs=0&sxsrf=AE3TifNCnc8PI4sKF10OxDIJoT4Z5dXh-A%3A1756667424438&q=Herencia&sa=X&ved=2ahUKEwi08ZmO4LWPAxVbQzABHVztEmwQxccNegQIEBAB&mstk=AUtExfCrosXwknB3yOHGSOQSFuFRSj8sdkbi8wZOZ49RgOWlLVOn4rOAxrHxzOwY-EU2ueHWv6jThiMwUh90IAS82g4-8j2A3vEzIZgwArqGlX6dtHW-Ad2r8RjnuXqhOGBGcRVRPjcAMbBMEiJ_UDV5I6mVpOqcoMKvLG8jb7FYPUicQos&csui=3)**:**

Permite que una clase (subclase o clase hija) herede propiedades y métodos de otra clase (superclase o clase padre), promoviendo la reutilización de código.

* [**Polimorfismo**](https://www.google.com/search?sca_esv=1f08cea1075aad4d&cs=0&sxsrf=AE3TifNCnc8PI4sKF10OxDIJoT4Z5dXh-A%3A1756667424438&q=Polimorfismo&sa=X&ved=2ahUKEwi08ZmO4LWPAxVbQzABHVztEmwQxccNegQIEhAB&mstk=AUtExfCrosXwknB3yOHGSOQSFuFRSj8sdkbi8wZOZ49RgOWlLVOn4rOAxrHxzOwY-EU2ueHWv6jThiMwUh90IAS82g4-8j2A3vEzIZgwArqGlX6dtHW-Ad2r8RjnuXqhOGBGcRVRPjcAMbBMEiJ_UDV5I6mVpOqcoMKvLG8jb7FYPUicQos&csui=3)**:**

La capacidad de objetos de diferentes tipos de responder a la misma llamada de método de manera diferente, ofreciendo flexibilidad.

**Sintaxis de Clases en JavaScript (ES6 y posteriores)**

Con la sintaxis de clases, la creación de objetos es más intuitiva:

*// Definición de una clase (plantilla)*

class Persona {  
 constructor(nombre, edad) {  
 this.nombre = nombre; /*/ Propiedad*

this.edad = edad; /*/ Propiedad*

}  
  
 /*/ Método*

saludar() {  
 console.log(`Hola, mi nombre es ${this.nombre}`);  
 }  
}  
  
// *Creación de objetos (instancias) a partir de la clase*

*co*nst persona1 = new Persona('Ana', 30);

const persona2 = new Persona('Juan', 25);

// *Uso de los objetos*

*pe*rsona1.saludar(); // Salida: Hola, mi nombre es Ana

**Herencia con**extends

Se utiliza la palabra clave extends para crear una subclase que herede de otra:

class Estudiante extends Persona {  
 constructor(nombre, edad, carrera) {  
 super(nombre, edad); *// Llama al constructor de la clase padre*  
 this.carrera = carrera; *// Nueva propiedad para Estudiante*  
 }  
  
 estudiar() {  
 console.log(`${this.nombre} está estudiando ${this.carrera}.`);  
 }  
}  
  
const estudiante1 = new Estudiante('Maria', 22, 'Informática');  
estudiante1.saludar(); // *Resultado de la herencia: Hola, mi nombre es Maria*

*e*studiante1.estudiar(); // Resultado de la subclase: Maria está estudiando Informática.