

Introducción al Desarrollo web

Curso: Desarrollo Frontend con React

Fundamentos de JavaScript



¿Qué vamos a aprender?



1 Introducción a JavaScript

Historia y usos principales



2 Variables y tipos de datos

Cómo almacenar y manejar información



3 Operadores

Realizar operaciones con datos



4 Estructuras de control

Condicionales y bucles



5 Ejemplos prácticos

Aplicando lo aprendido

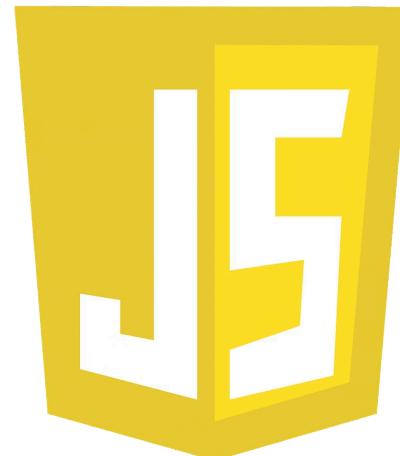
¿Qué es JavaScript?

JavaScript es un lenguaje de programación interpretado que se ejecuta principalmente en el navegador web.

Fue creado en 1995 por Brendan Eich en solo 10 días mientras trabajaba en Netscape.

A pesar de su nombre, [no tiene relación directa con Java](#). Son lenguajes completamente diferentes.

JavaScript



¿Por qué JavaScript?



Ubicuidad

Presente en casi todas las páginas web modernas



Versatilidad

Frontend, backend, móvil, escritorio, IoT...



Comunidad

Gran comunidad de desarrolladores y recursos

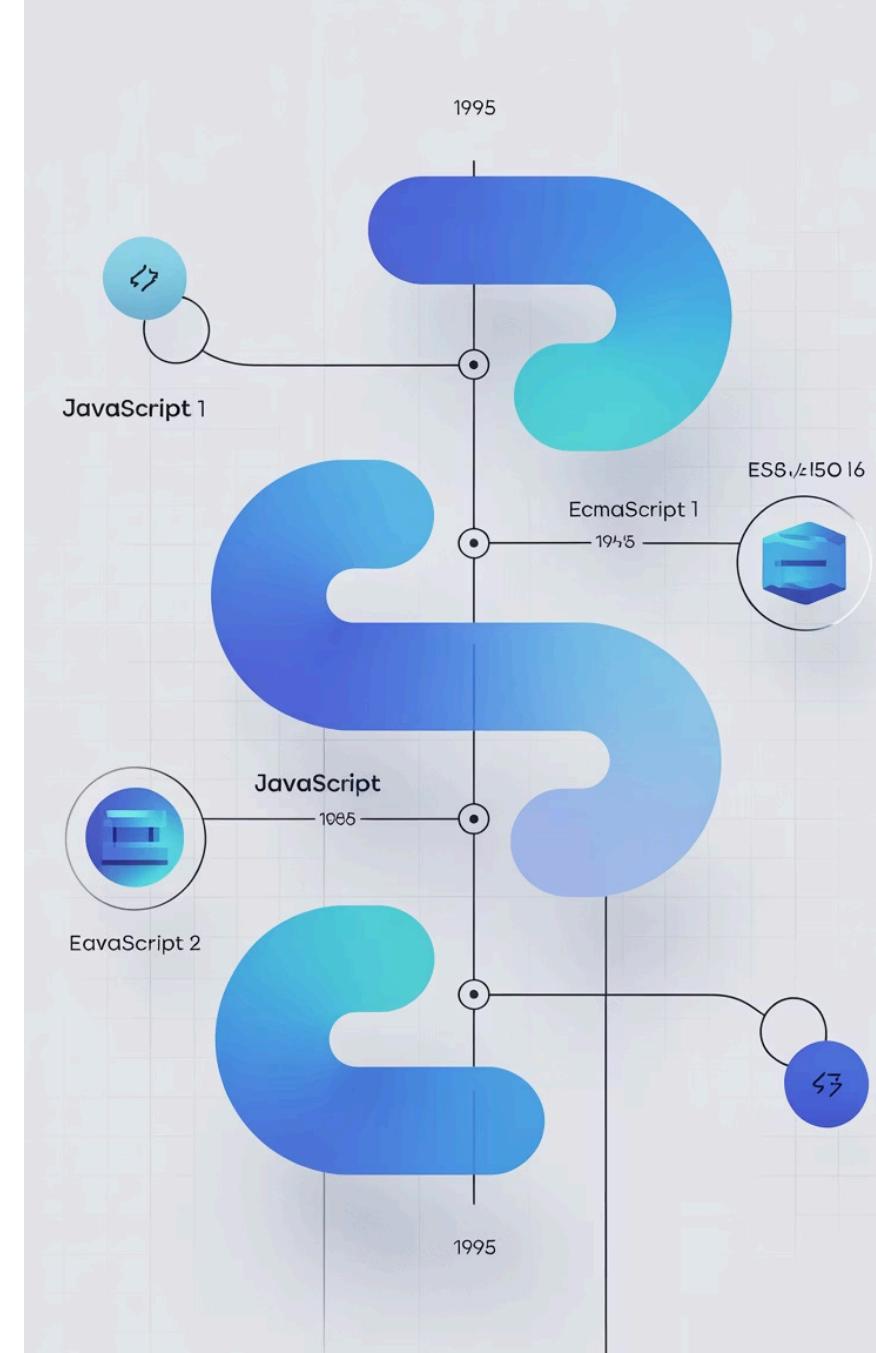
Es el [lenguaje más utilizado](#) según la encuesta anual de Stack Overflow desde 2013.

Evolución de JavaScript

A lo largo de los años, JavaScript ha evolucionado significativamente:

- 1995: Nace como Mocha, luego LiveScript
- 1997: Se establece el estándar ECMAScript
- 2009: ES5 introduce mejoras importantes
- 2015: ES6/ES2015 revoluciona el lenguaje
- Actualización anual desde entonces

Hoy aprenderemos la sintaxis moderna de JavaScript (ES6+)



Integración de JavaScript en la web

Existen tres formas principales de incluir JavaScript en una página web:

Interno (dentro de etiquetas script)

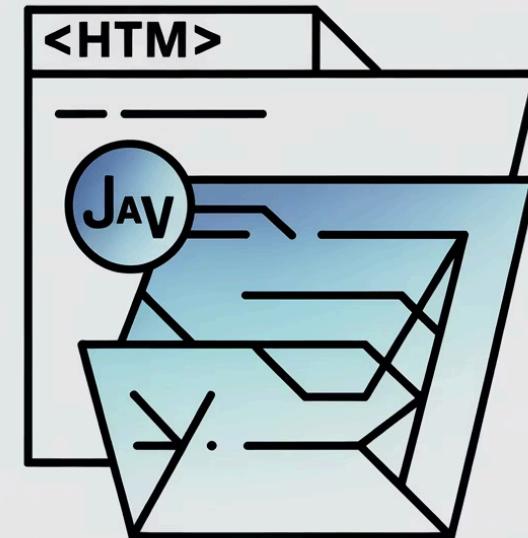
```
<script>  
  alert("¡Hola mundo!");  
</script>
```

Externo (archivo separado)

```
<script src="app.js"></script>
```

Inline (en atributos HTML)

```
<button onclick="alert('¡Hola!')">  
  Clic aquí  
</button>
```

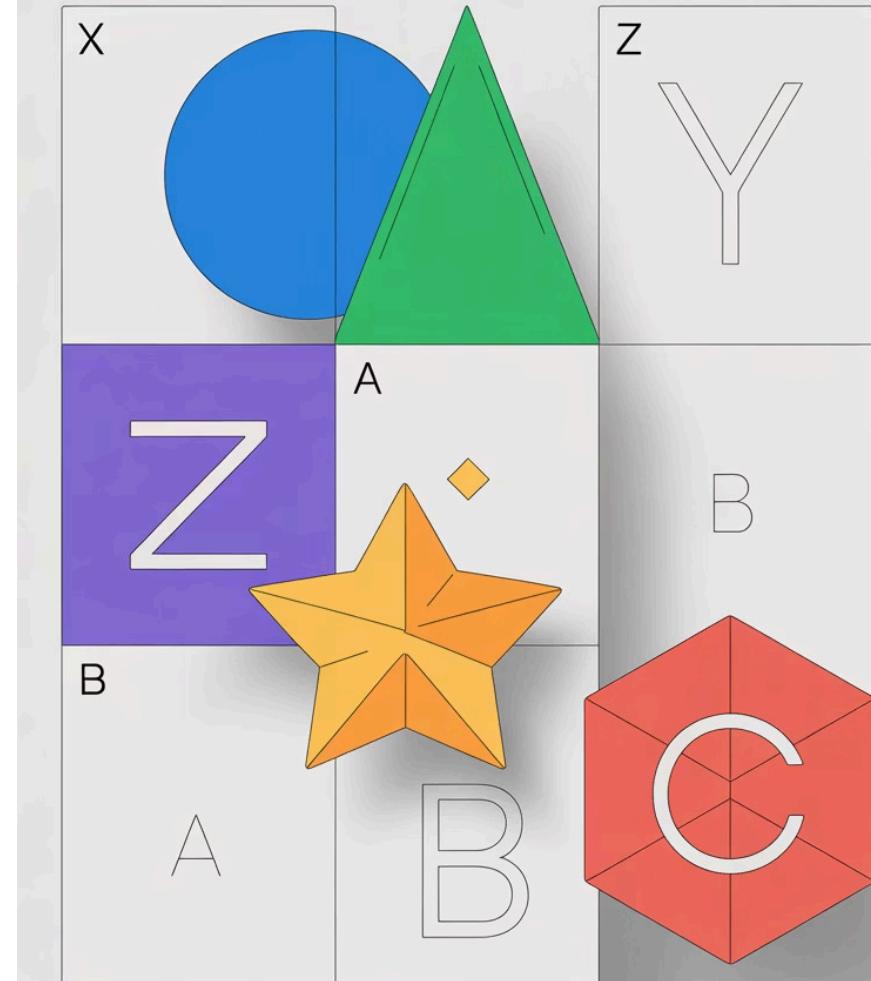


Variables en JavaScript

Las variables son contenedores para almacenar datos que podemos usar y modificar en nuestro programa.

```
// Sintaxis básica  
nombreVariable = valor;
```

En JavaScript moderno, [siempre debemos declarar nuestras variables](#) antes de usarlas usando var, let o const.



Declaración de variables

var

Forma tradicional (pre-ES6)

```
var edad = 18;  
var nombre = "Ana";
```

Tiene ámbito de función o global

let

Forma moderna recomendada

```
let puntuacion = 75;  
let estaActivo = true;
```

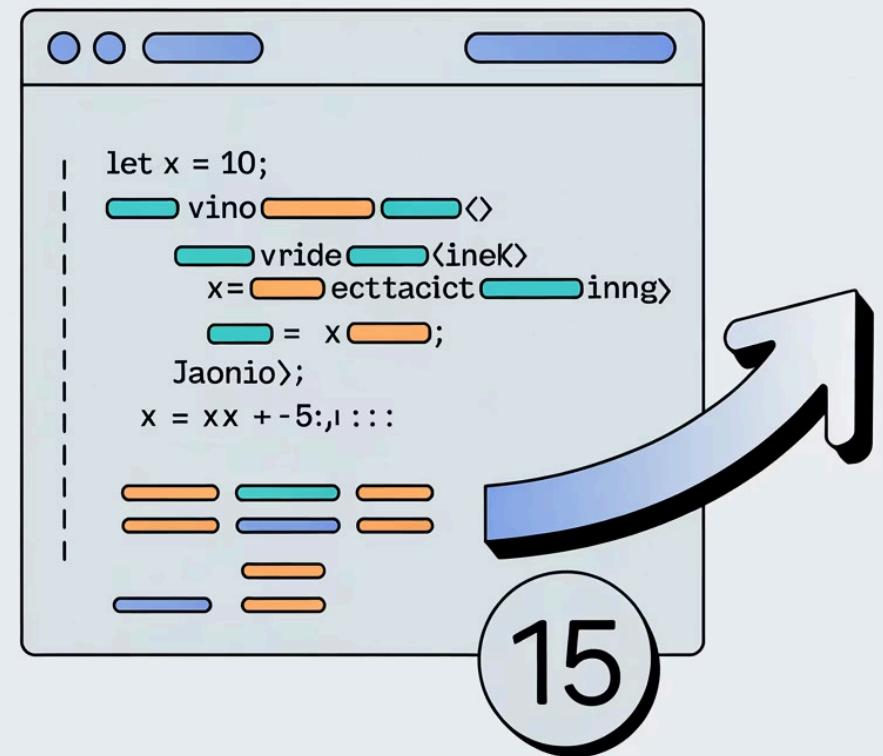
Tiene ámbito de bloque

const

Para valores constantes

```
const PI = 3.14159;  
const URL = "https://...";
```

No se puede reasignar



Ejemplos prácticos de variables

// Usando let (valores que cambian)

```
let contador = 0;  
contador = contador + 1; // Ahora es 1
```

// Usando const (valores fijos)

```
const GRAVEDAD = 9.8;  
// GRAVEDAD = 10; // Error, no se puede cambiar
```

¿Cuándo usar cada tipo de variable?

var

En código antiguo o cuando necesitas ámbito de función.

Se recomienda evitar su uso en código nuevo.

let

Para valores que necesitan cambiar durante la ejecución.

Ejemplos: contadores, acumuladores, banderas.

const

Para valores que **no deben cambiar** (predeterminado).

Ejemplos: configuraciones, URLs, constantes matemáticas.

Regla general: Usa **const** por defecto, cambia a **let** solo si necesitas reasignar.

Tipos de datos en JavaScript

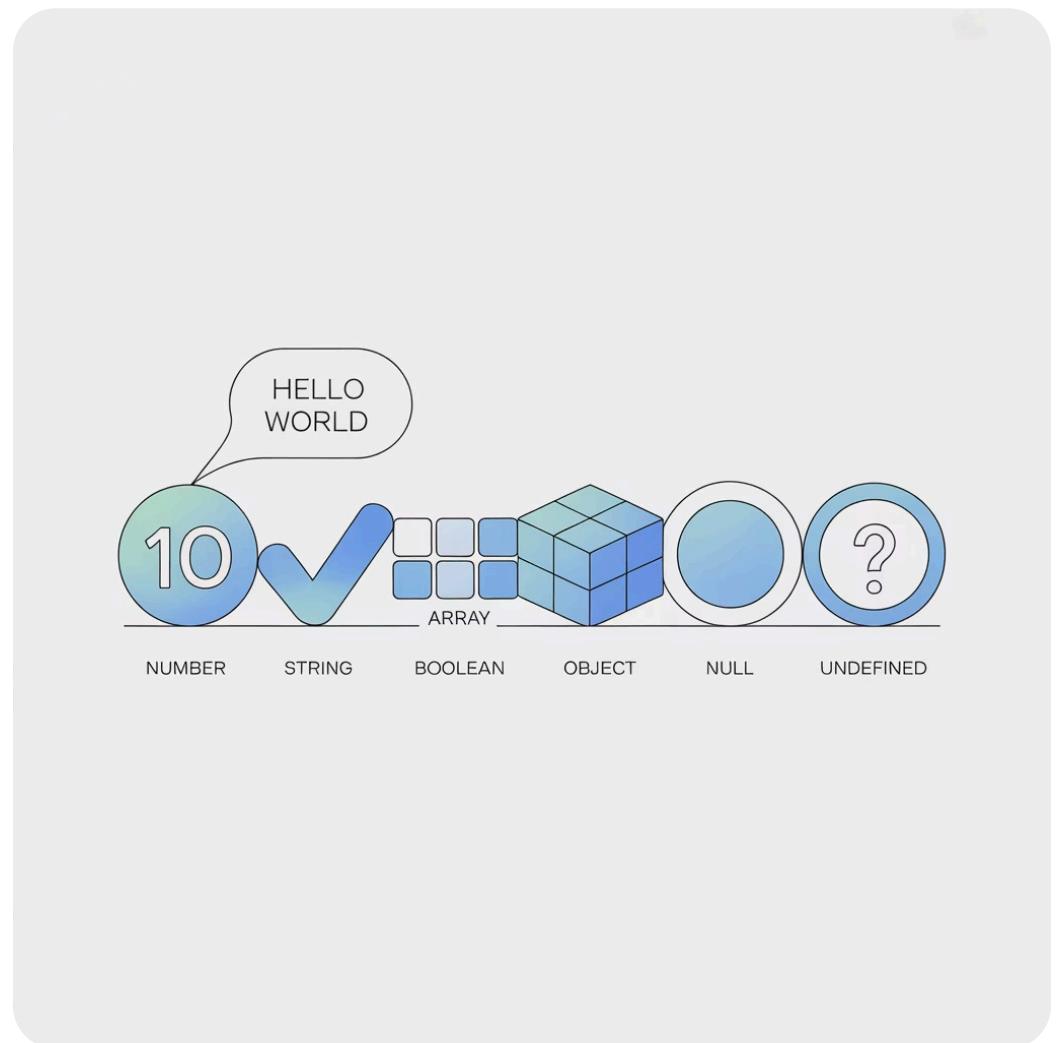
JavaScript tiene 8 tipos de datos básicos divididos en dos categorías:

Primitivos

- **String:** Texto - "Hola"
- **Number:** Números - 42, 3.14
- **Boolean:** true/false
- **undefined:** Valor no asignado
- **null:** Valor vacío intencional
- **Symbol:** Identificador único
- **BigInt:** Números enteros grandes

Referencia

- **Object:** Colección de propiedades
- Arrays (técticamente objetos)
- Funciones (técticamente objetos)
- Fechas (técticamente objetos)
- ... y más



Tipos de datos: Ejemplos

```
// Strings (cadenas de texto)
let nombre = "María";
let mensaje = 'Hola';
let frase = `${nombre} dice ${mensaje}`; // Template literal

// Numbers (números)
let entero = 42;
let decimal = 3.14;
let negativo = -10;

// Booleans (valores lógicos)
let esMayorDeEdad = true;
let tieneDescuento = false;

// undefined y null
let noDefinido; // undefined automáticamente
let valorVacio = null; // hay que asignarlo explícitamente

// Object (objeto)
let persona = {
  nombre: "Carlos",
  edad: 25,
  esEstudiante: true
};
```

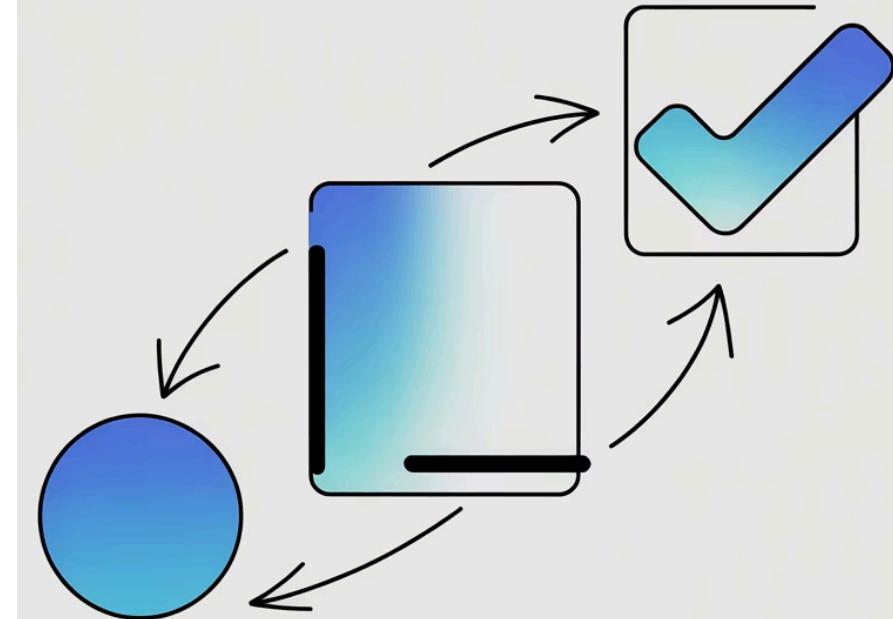
Tipado dinámico

JavaScript es un lenguaje de **tipado dinámico**, lo que significa que:

- No necesitas declarar el tipo de una variable
- Las variables pueden cambiar de tipo durante la ejecución

```
let variable = 42;    // Number  
variable = "Hola";   // Ahora es String  
variable = true;    // Ahora es Boolean  
variable = [1, 2, 3]; // Ahora es Array  
variable = null;    // Ahora es null
```

Esto da flexibilidad pero también puede causar errores inesperados si no tienes cuidado.



Comprobación de tipos

Para saber el tipo de una variable, usamos el operador `typeof`:

```
let texto = "Hola";
console.log(typeof texto); // "string"
```

```
let numero = 42;
console.log(typeof numero); // "number"
```

```
let booleano = false;
console.log(typeof booleano); // "boolean"
```

```
let array = [1, 2, 3];
console.log(typeof array); // "object" (¡cuidado!)
```

```
let nulo = null;
console.log(typeof nulo); // "object" (¡peculiaridad de JS!)
```

```
let noDefinido;
console.log(typeof noDefinido); // "undefined"
```

Atención: Hay algunas peculiaridades como que `typeof null` devuelve "object" y `typeof array` también devuelve "object".

Operadores en JavaScript

Los operadores nos permiten realizar operaciones con variables y valores.

Operadores aritméticos

Para cálculos matemáticos

- + Suma
- Resta
- * Multiplicación
- / División
- % Módulo (resto)
- ** Exponente
- ++ Incremento
- Decremento

Operadores de asignación

Para asignar valores

- = Asignación simple
- += Suma y asignación
- = Resta y asignación
- *= Multiplicación y asignación
- /= División y asignación
- %= Módulo y asignación

Operadores de comparación

Devuelven booleanos

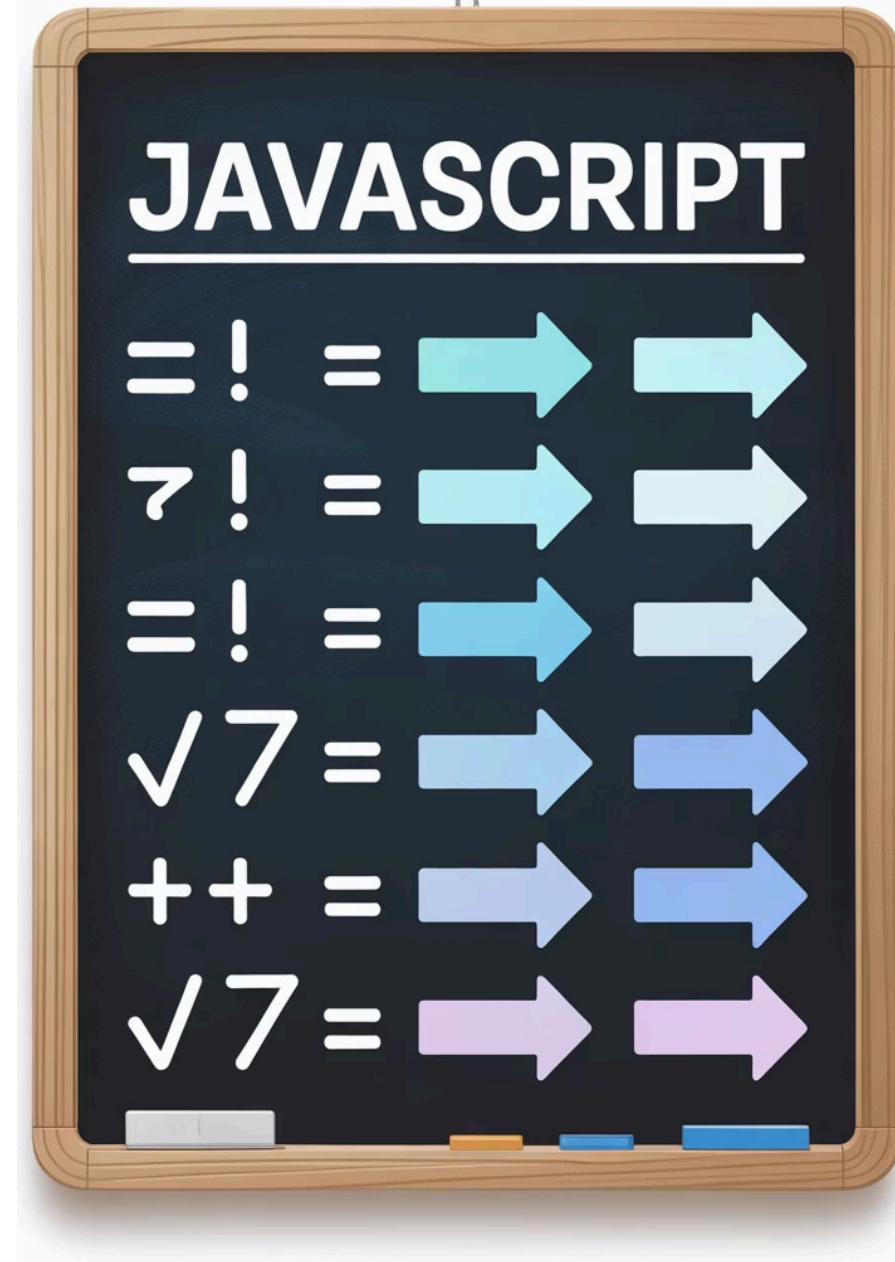
- == Igualdad (valor)
- === Igualdad estricta (valor y tipo)
- != Desigualdad
- !== Desigualdad estricta
- > Mayor que
- < Menor que
- >= Mayor o igual que
- <= Menor o igual que

Operadores: Ejemplos prácticos

```
// Aritméticos
let a = 5 + 3; // 8
let b = 10 - 4; // 6
let c = 3 * 4; // 12
let d = 20 / 5; // 4
let e = 10 % 3; // 1 (resto de 10/3)
let f = 2 ** 3; // 8 (2 elevado a 3)

// Incremento/decremento
let contador = 1;
contador++; // ahora es 2
contador--; // vuelve a 1

// Asignación compuesta
let numero = 5;
numero += 3; // equivale a: numero = numero + 3
// ahora numero es 8
```



Operadores de comparación

Los operadores de comparación devuelven siempre un valor booleano (true o false).

```
// Comparación  
let x = 5;  
let y = "5";  
  
console.log(x == y); // true (compara valor)  
console.log(x === y); // false (compara valor y tipo)
```

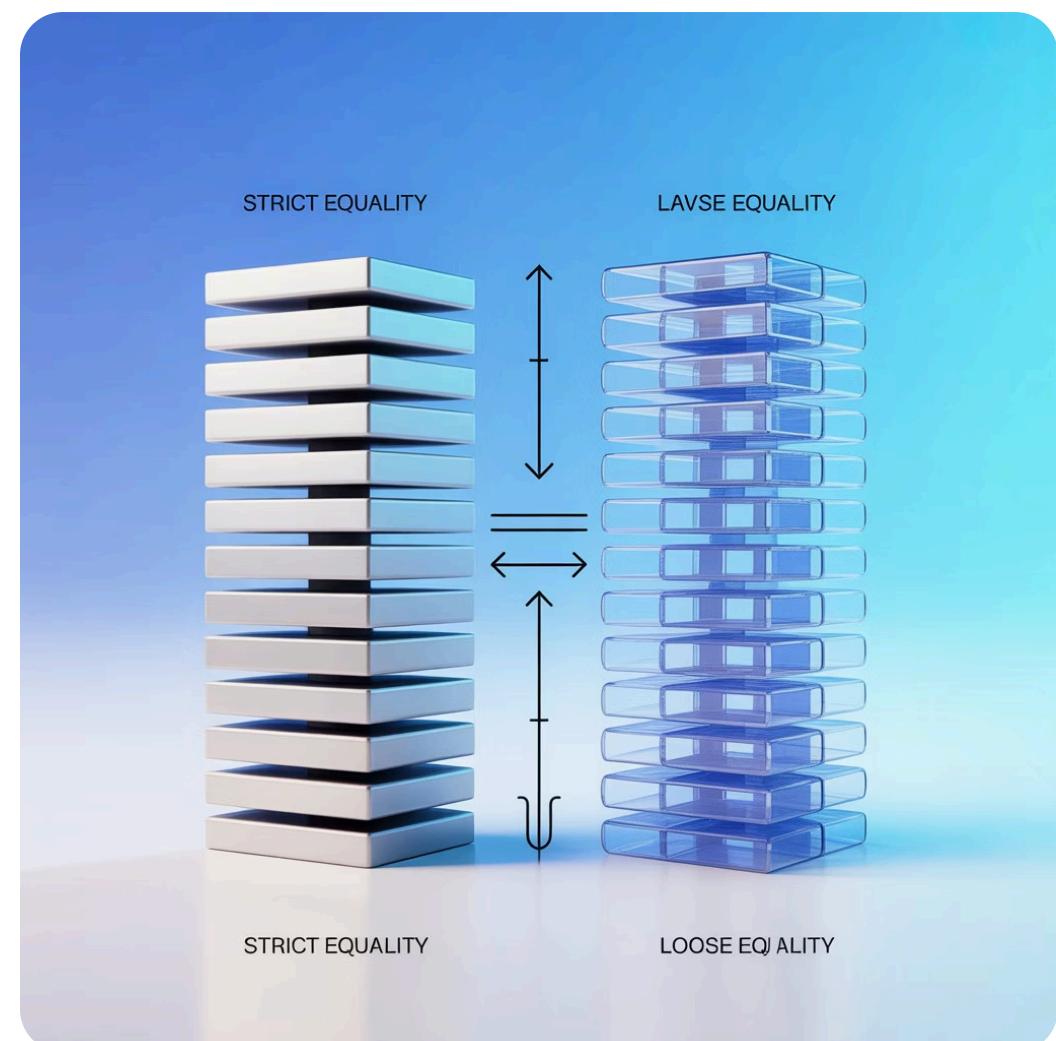
```
console.log(x != y); // false  
console.log(x !== y); // true
```

```
let a = 10;  
let b = 5;  
console.log(a > b); // true  
console.log(a < b); // false  
console.log(a >= b); // true  
console.log(a <= b); // false
```

⚠️ ¡Importante!

Siempre usa el operador de igualdad estricta (==>) en lugar del operador de igualdad simple (==).

La igualdad estricta comprueba tanto el valor como el tipo, evitando comportamientos inesperados.



Operadores lógicos

AND lógico (&&)

Devuelve true solo si **ambos** operandos son true.

```
true && true // true  
true && false // false  
false && true // false  
false && false // false
```

OR lógico (||)

Devuelve true si **al menos uno** de los operandos es true.

```
true || true // true  
true || false // true  
false || true // true  
false || false // false
```

NOT lógico (!)

Invierte el valor de un operando booleano.

```
!true // false  
!false // true  
  
let activo = true;  
let inactivo = !activo; // false
```

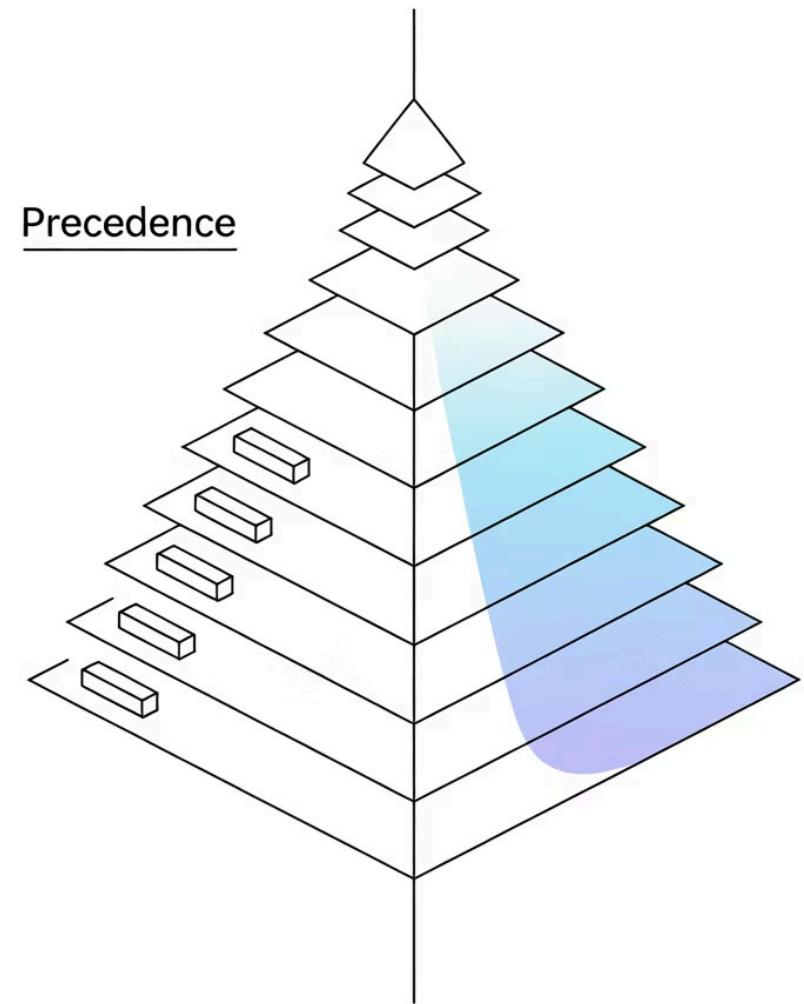
Estos operadores son fundamentales para crear condiciones complejas.

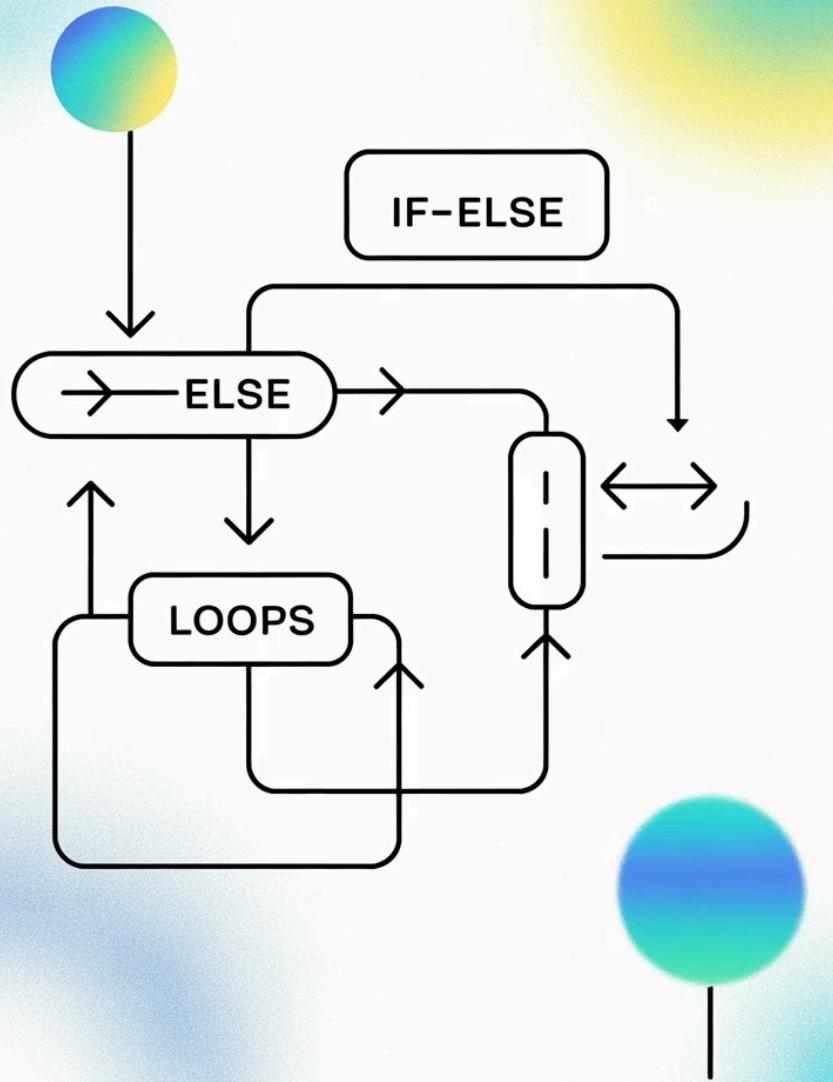
Precedencia de operadores

Al igual que en matemáticas, JavaScript sigue un orden de operaciones:

1. Paréntesis ()
2. Incremento/decremento ++, --
3. Negación !, unario +, -
4. Exponente **
5. Multiplicación *, división /, módulo %
6. Suma +, resta -
7. Comparación <, >, <=, >=
8. Igualdad ==, ===, !=, !==
9. AND lógico &&
10. OR lógico ||
11. Asignación =, +=, etc.

Cuando tengas dudas, usa [paréntesis](#) para forzar el orden de evaluación.





Estructuras de control

Las estructuras de control nos permiten modificar el flujo de ejecución de nuestro programa.

Condicionales

- **if / else if / else:** Ejecuta código según una condición
- **switch:** Evalúa una expresión contra múltiples casos
- **operador ternario:** Versión abreviada de if/else

Bucles

- **for:** Repite código un número específico de veces
- **while:** Repite mientras una condición sea verdadera
- **do...while:** Como while, pero garantiza una ejecución
- **for...of:** Itera sobre elementos de objetos iterables
- **for...in:** Itera sobre propiedades de un objeto

Condicional if / else

La estructura `if / else` nos permite ejecutar código solo si se cumple una condición.

```
// Sintaxis básica  
if (condicion) {  
    // Código a ejecutar si la condición es true  
} else {  
    // Código a ejecutar si la condición es false  
}
```

```
// Con else if para múltiples condiciones  
if (condicion1) {  
    // Si condicion1 es true  
} else if (condicion2) {  
    // Si condicion1 es false y condicion2 es true  
} else {  
    // Si todas las condiciones son false  
}
```

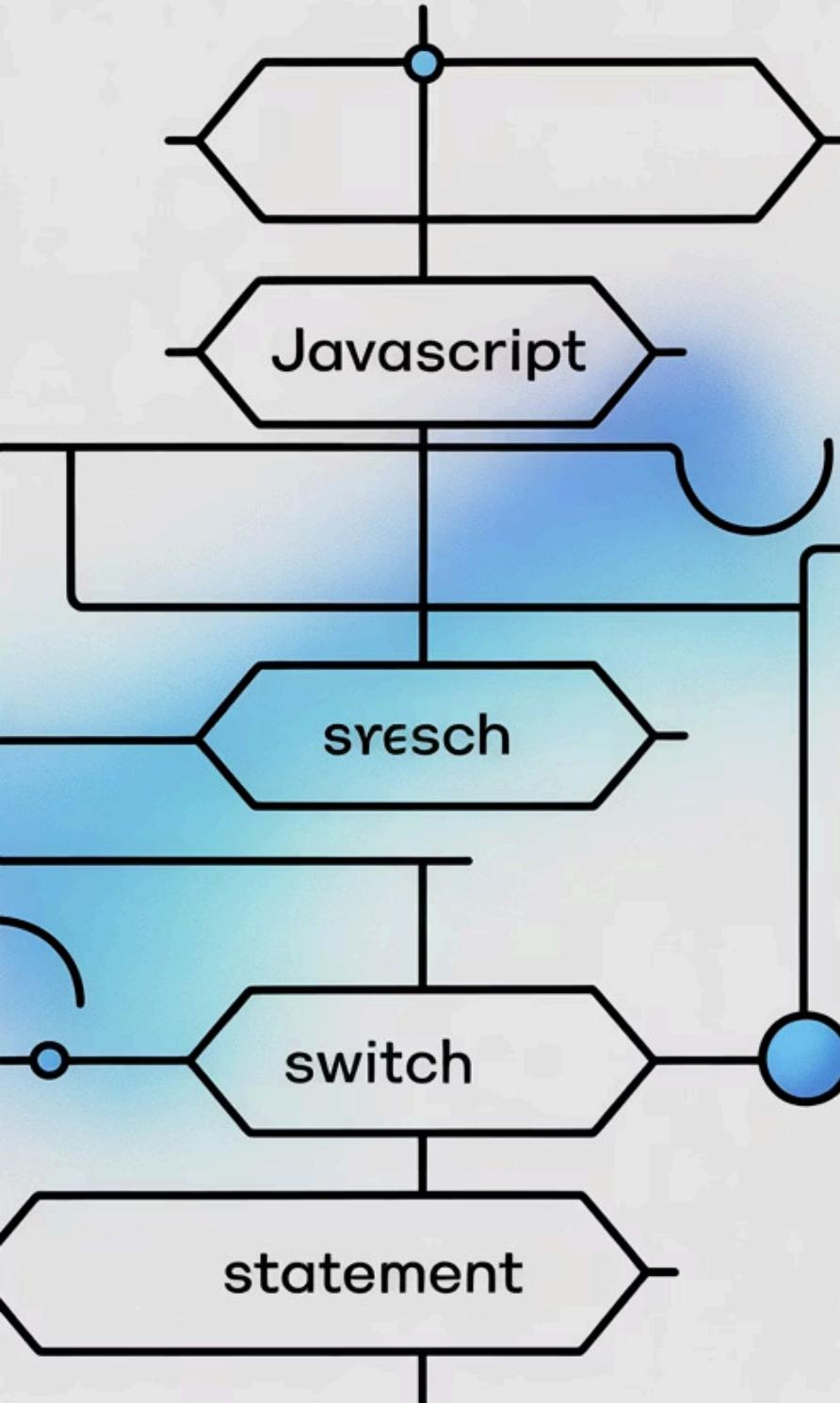
```
// Ejemplo: Sistema de calificaciones  
let nota = 85;  
let mensaje;
```

```
if (nota >= 90) {  
    mensaje = "Sobresaliente";  
} else if (nota >= 70) {  
    mensaje = "Notable";  
} else if (nota >= 60) {  
    mensaje = "Bien";  
} else if (nota >= 50) {  
    mensaje = "Suficiente";  
} else {  
    mensaje = "Insuficiente";  
}
```

```
console.log(mensaje); // "Notable"
```

Switch

La estructura `switch` evalúa una expresión y ejecuta el código correspondiente al caso que coincida.



```
// Sintaxis
switch (expresion) {
  case valor1:
    // Código si expresion === valor1
    break;
  case valor2:
    // Código si expresion === valor2
    break;
  default:
    // Código si no coincide con ningún caso
}
```

```
// Ejemplo: Días de la semana
let dia = 3;
let nombreDia;

switch (dia) {
  case 1: nombreDia = "Lunes"; break;
  case 2: nombreDia = "Martes"; break;
  case 3: nombreDia = "Miércoles"; break;
  case 4: nombreDia = "Jueves"; break;
  case 5: nombreDia = "Viernes"; break;
  case 6: nombreDia = "Sábado"; break;
  case 7: nombreDia = "Domingo"; break;
  default: nombreDia = "Día inválido";
}
```

```
console.log(nombreDia); // "Miércoles"
```

¿Cuándo usar if vs switch?

Usa **if/else if/else** cuando:

- Tienes condiciones lógicas complejas
- Las condiciones implican rangos ($>$, $<$, \geq , etc.)
- Tienes pocas condiciones a evaluar
- Las condiciones son diferentes entre sí

```
if (edad >= 18) {  
    console.log("Eres mayor de edad");  
} else {  
    console.log("Eres menor de edad");  
}
```

Usa **switch** cuando:

- Comparas una misma variable con valores exactos
- Tienes muchos casos posibles
- Necesitas mayor legibilidad con muchas opciones

```
switch (diaSemana) {  
    case 1: case 2: case 3: case 4: case 5:  
        console.log("Día laborable");  
        break;  
    case 6: case 7:  
        console.log("Fin de semana");  
        break;  
    default:  
        console.log("Día inválido");  
}
```

Operador ternario

El operador ternario es una forma concisa de escribir una condición simple if/else en una sola línea.

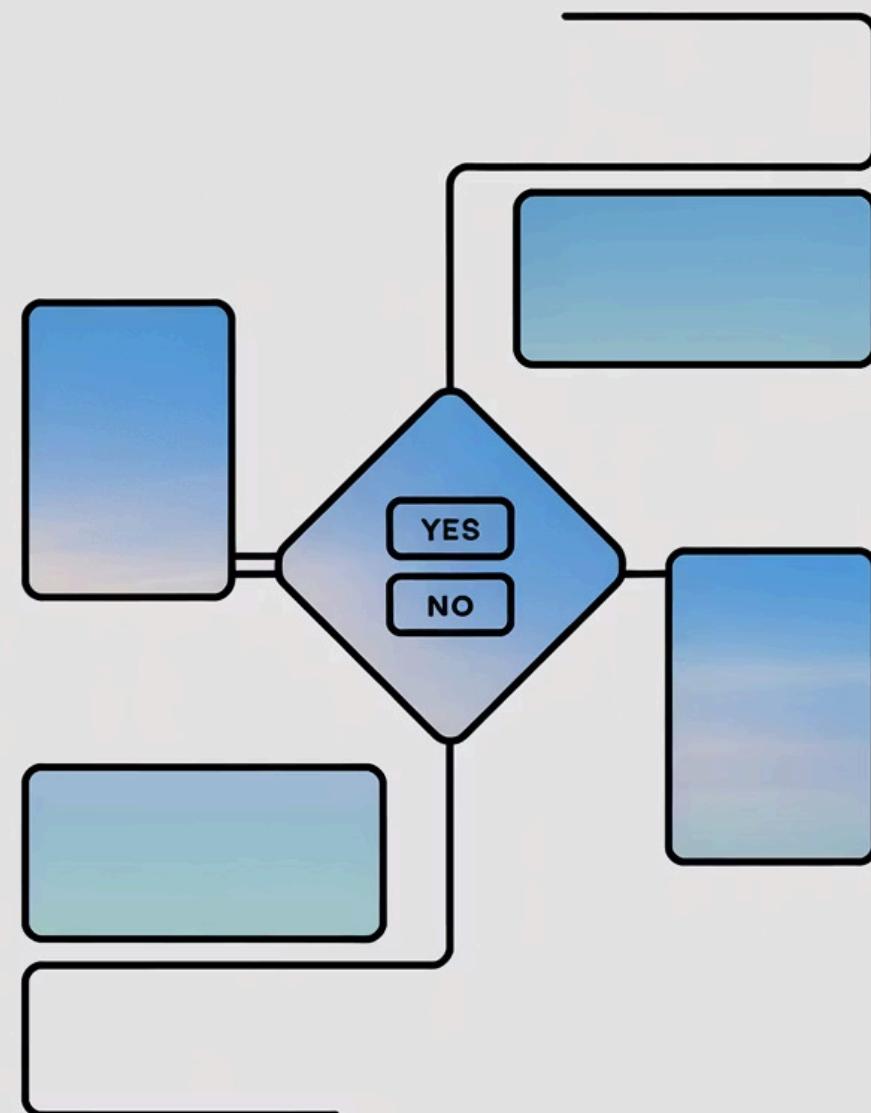
```
// Sintaxis  
condicion ? valorSiTrue : valorSiFalse;
```

```
// Equivalente a:  
if (condicion) {  
    valorSiTrue;  
} else {  
    valorSiFalse;  
}
```

Ejemplos:

```
// Ejemplo 1: Asignar un valor basado en una condición  
let edad = 20;  
let mensaje = edad >= 18 ? "Adulto" : "Menor";  
console.log(mensaje); // "Adulto"
```

```
// Ejemplo 2: Dentro de una plantilla de texto  
let temperatura = 25;  
console.log(`Está ${temperatura} > 30 ? "caliente" : "agradable"}`);  
// "Está agradable"
```



Bucle for

El bucle `for` repite un bloque de código un número específico de veces.

```
// Sintaxis  
for (inicialización; condición; actualización) {  
    // Código a repetir  
}
```

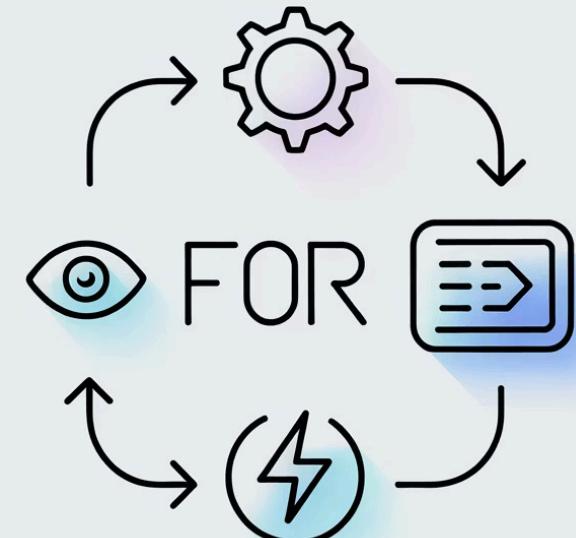
```
// Partes:  
// - inicialización: se ejecuta una vez al inicio  
// - condición: se evalúa antes de cada iteración  
// - actualización: se ejecuta después de cada iteración
```

```
// Ejemplo: Contar del 1 al 5  
for (let i = 1; i <= 5; i++) {  
    console.log(i);  
}  
// Imprime: 1, 2, 3, 4, 5
```

El bucle for es muy versátil y se puede usar para:

- Recorrer arrays
- Repetir una tarea un número exacto de veces
- Generar secuencias numéricas

```
// Recorrer un array  
let frutas = ["Manzana", "Pera", "Uva"];  
for (let i = 0; i < frutas.length; i++) {  
    console.log(frutas[i]);  
}  
// Imprime: "Manzana", "Pera", "Uva"
```



Bucle while

El bucle `while` repite un bloque de código mientras una condición sea verdadera.

```
// Sintaxis
while (condicion) {
  // Código a repetir mientras la condición sea true
}
```

```
// Ejemplo: Contar del 1 al 5
let i = 1;
while (i <= 5) {
  console.log(i);
  i++; // ¡No olvides actualizar la condición!
}
// Imprime: 1, 2, 3, 4, 5
```

¡Importante! Si la condición no cambia dentro del bucle, se creará un **bucle infinito** que puede bloquear tu programa.

Variante: do...while

Similar a while, pero garantiza que el código se ejecute al menos una vez.

```
// Sintaxis
do {
  // Código a repetir
} while (condicion);
```

```
// Ejemplo
let j = 1;
do {
  console.log(j);
  j++;
} while (j <= 5);
// Imprime: 1, 2, 3, 4, 5
```

La diferencia principal es que do...while comprueba la condición **después** de ejecutar el código, no antes.

For...of y For...in

for...of

Itera sobre los **valores** de objetos iterables (arrays, strings, etc.)

```
// Sintaxis
for (let elemento of iterable) {
  // Código a ejecutar con cada elemento
}

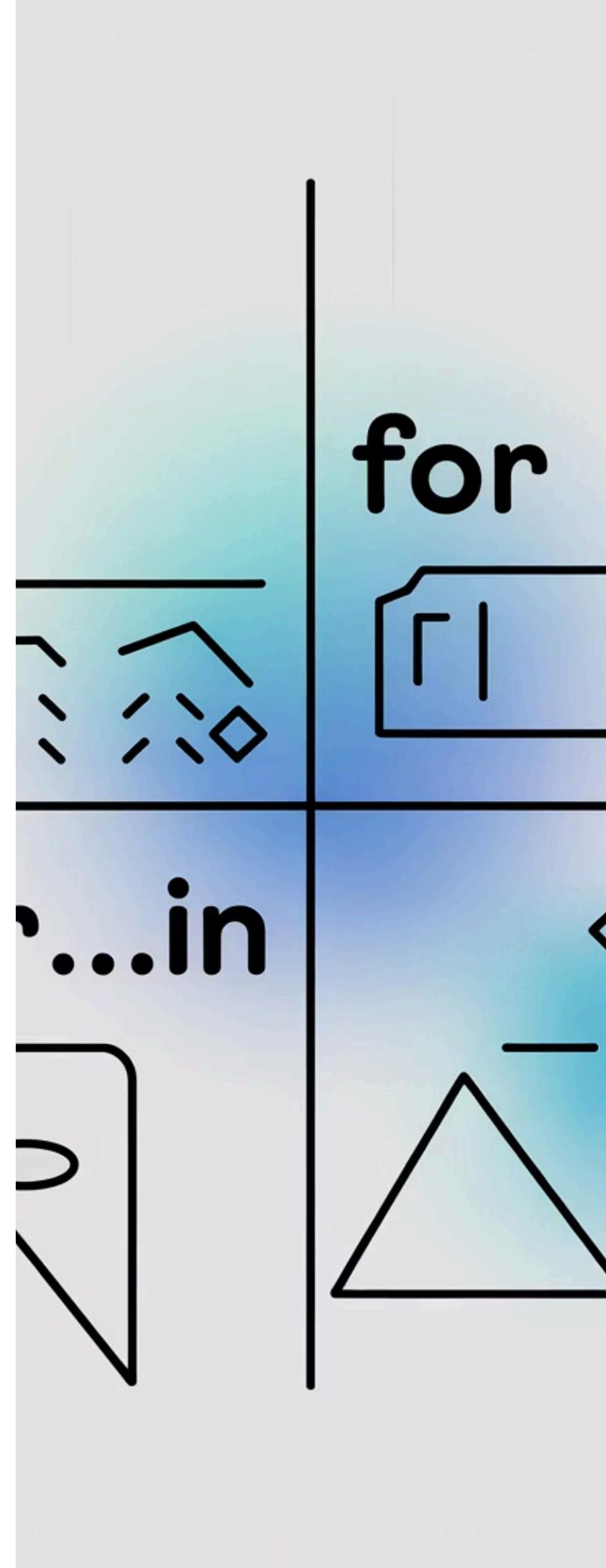
// Ejemplo con array
let colores = ["rojo", "verde", "azul"];
for (let color of colores) {
  console.log(color);
}
// Imprime: "rojo", "verde", "azul"
```

for...in

Itera sobre las **propiedades** de un objeto

```
// Sintaxis
for (let propiedad in objeto) {
  // Código a ejecutar con cada propiedad
}

// Ejemplo con objeto
let persona = {
  nombre: "Ana",
  edad: 28,
  profesion: "Ingeniera"
};
for (let prop in persona) {
  console.log(prop + ": " + persona[prop]);
}
// Imprime: "nombre: Ana", "edad: 28", "profesion: Ingeniera"
```



¿Qué bucle debo usar?

1

for

Cuando conoces el número exacto de iteraciones o necesitas un contador.

```
for (let i = 0; i < 10; i++) { ... }
```

2

while

Cuando no sabes cuántas iteraciones necesitarás y dependes de una condición.

```
while (hayDatos) { ... }
```

3

do...while

Cuando necesitas ejecutar el código al menos una vez antes de verificar la condición.

```
do { ... } while (condicion);
```

4

for...of

Cuando quieres recorrer los valores de un array u otro iterable.

```
for (let valor of array) { ... }
```

5

for...in

Cuando necesitas recorrer las propiedades de un objeto.

```
for (let prop in objeto) { ... }
```

Ejemplos prácticos

Calculadora de promedio

```
// Array de calificaciones
let notas = [85, 90, 78, 92, 88];

// Inicializar la suma
let suma = 0;

// Sumar todas las notas
for (let i = 0; i < notas.length; i++) {
    suma += notas[i];
}

// Calcular el promedio
let promedio = suma / notas.length;

// Determinar la calificación
let calificacion;
if (promedio >= 90) {
    calificacion = "A";
} else if (promedio >= 80) {
    calificacion = "B";
} else if (promedio >= 70) {
    calificacion = "C";
} else if (promedio >= 60) {
    calificacion = "D";
} else {
    calificacion = "F";
}

console.log("Promedio: " + promedio);
console.log("Calificación: " + calificacion);
```

Verificador de número primo

```
// Función que verifica si un número es primo
function esPrimo(numero) {
    // Los números menores que 2 no son primos
    if (numero < 2) {
        return false;
    }

    // Verificar si es divisible por algún número
    // entre 2 y la raíz cuadrada del número
    for (let i = 2; i <= Math.sqrt(numero); i++) {
        if (numero % i === 0) {
            return false; // No es primo
        }
    }

    return true; // Es primo
}

// Probar la función
let numerosAVerificar = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

for (let num of numerosAVerificar) {
    if (esPrimo(num)) {
        console.log(num + " es primo");
    } else {
        console.log(num + " no es primo");
    }
}
```

Preguntas para reflexionar

“**¿Cuál es la diferencia entre let y const?**

let permite reasignar valores a la variable, mientras que const crea una referencia constante que no puede ser reasignada. Sin embargo, los objetos y arrays declarados con const pueden modificar su contenido.

“**¿En qué casos usarías switch en lugar de if/else?**

Usaría switch cuando necesito comparar una misma variable contra múltiples valores exactos, especialmente si hay muchos casos. Proporciona mejor legibilidad con muchas opciones y es más eficiente en estos escenarios.

“**¿Por qué es importante evitar bucles infinitos?**

Los bucles infinitos pueden consumir todos los recursos del sistema, bloquear el navegador o hacer que tu aplicación deje de responder. Siempre asegúrate de que la condición del bucle eventualmente será falsa.

¡Gracias por tu atención! ¿Alguna pregunta?

#EDCOUNIANDES

<https://educacioncontinua.uniandes.edu.co/>

Contacto: educacion.continua@uniandes.edu.co

© - Derechos Reservados: La presente obra, y en general todos sus contenidos, se encuentran protegidos por las normas internacionales y nacionales vigentes sobre propiedad Intelectual, por lo tanto su utilización parcial o total, reproducción, comunicación pública, transformación, distribución, alquiler, préstamo público e importación, total o parcial, en todo o en parte, en formato impreso o digital y en cualquier formato conocido o por conocer, se encuentran prohibidos, y solo serán lícitos en la medida en que se cuente con la autorización previa y expresa por escrito de la Universidad de los Andes.