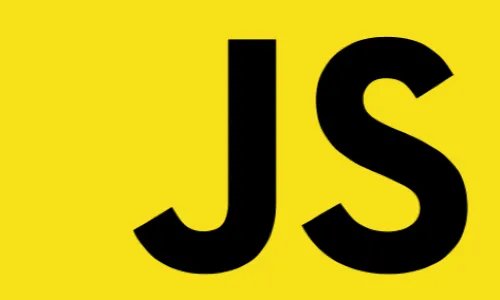
Plan de Temas a desarrollar para aprendizaje de JAVASCRIPT



**ÍNDICE**

[Estructura de Temas](#_si8jv5ionwdv)

[1 - Fundamentos Básicos](#_iqeebgh6oqbr)

[1 - A) Variables y Tipos de Datos:](#_f2vgma7yc77c)

[1 - B) Tipos de Datos Básicos:](#_bsg7cjol6fvz)

[1 - C) Conversión entre Tipos de Datos:](#_7dshgqk2t6uo)

[1 - EJERCICIOS:](#_2earg6jgo4vt)

# Estructura de Temas

* Fundamentos Básicos:
  + Variables y tipos de datos.
  + Operadores.
  + Estructuras de control (if, else, switch).
  + Bucles (for, while).
* Funciones:
  + Declaración y llamada de funciones.
  + Parámetros y argumentos.
  + Retorno de funciones.
  + Funciones anónimas y funciones de flecha (Arrow Functions).
* Objetos y Tipos de Datos Avanzados:
  + Objetos y propiedades.
  + Arreglos y métodos de arreglo.
  + Strings y métodos de string.
  + Objetos Date y Math.
* Programación Orientada a Objetos (OOP):
  + Conceptos básicos de OOP.
  + Creación de clases y objetos.
  + Herencia y polimorfismo.
* DOM (Document Object Model):
  + Acceso y manipulación del DOM.
  + Eventos y manejo de eventos.
  + Traversing del DOM.
* Asincronismo:
  + Callbacks.
  + Promesas.
  + Async/Await.
* ES6 y Funcionalidades Modernas:
  + Let y const.
  + Template literals.
  + Destructuring.
  + Arrow functions.
  + Clases.
  + Módulos.
* Manejo de Errores:
  + Excepciones y manejo de errores.
* AJAX y Fetch API:
  + Realización de solicitudes HTTP asíncronas.
* Herramientas de Desarrollo:
  + Uso de consola.
  + Depuración de código.
* Frameworks y Bibliotecas:
  + Familiarización con al menos uno de los principales frameworks/librerías como React, Angular o Vue.js.
* Testing:
  + Principios básicos de pruebas y uso de frameworks de prueba como Jest.
* Gestión de Paquetes y Módulos:
  + Uso de npm (Node Package Manager) para gestionar dependencias.
* Seguridad:
  + Principios básicos de seguridad en el desarrollo web.
* Optimización y Rendimiento:
  + Técnicas para mejorar el rendimiento de las aplicaciones.

# 1 - Fundamentos Básicos

* Variables y Tipos de Datos:
  + Declaración de variables (var, let, const).
  + Tipos de datos básicos: números, strings, booleanos, null, undefined.
  + Conversión entre tipos de datos.
* Operadores:
  + Operadores aritméticos (+, -, \*, /, %).
  + Operadores de asignación (=, +=, -=, \*=, /=).
  + Operadores de comparación (==, ===, !=, !==, >, <, >=, <=).
  + Operadores lógicos (&&, ||, !).
  + Operadores de incremento (++, --).
  + Operadores ternarios.
* Estructuras de Control:
  + Sentencias if, else if, else.
  + Sentencias switch.
* Bucles:
  + Bucle for.
  + Bucle while.
  + Bucle do-while.

## 1 - A) Variables y Tipos de Datos:

### Declaración de Variables:

En JavaScript, las variables son contenedores que almacenan datos y tienen varios métodos de declaración. Las tres formas comunes de declarar variables son **var, let, y const.**

* **var:** Antes de ECMAScript 6 (ES6), var era la única manera de declarar variables en JavaScript. Sin embargo, tiene alcance de función, lo que significa que puede tener problemas de "hoisting" (levantamiento), y no tiene alcance de bloque.
* **let:** Introducido en ES6, let soluciona algunos problemas de var. Tiene alcance de bloque, lo que significa que solo es válido dentro del bloque en el que se declara.
* **const:** Similar a let, pero constante, es decir, una vez que se le asigna un valor, no se puede reasignar. Sin embargo, el contenido mutable de objetos y arreglos declarados con const aún puede cambiar.

## 

### Tipos de Datos Básicos:

JavaScript tiene varios tipos de datos básicos que se utilizan para almacenar información. Algunos de los más comunes son:

* Número (Number): Representa valores numéricos. Pueden ser enteros o decimales.

let edad = 25;

let altura = 1.75;

* Cadena (String): Secuencia de caracteres, utilizada para texto.

let nombre = "Juan";

* Booleano (Boolean): Representa valores lógicos true o false.

let esMayor = true;

* Nulo (null) y Indefinido (undefined): null se usa para representar la ausencia intencional de cualquier objeto o valor, mientras que undefined generalmente indica que la variable ha sido declarada pero aún no se le ha asignado un valor.

let nulo = null;

let indefinido;

#### CÓDIGO 1:

// Número (Number)

let edad = 25;

let altura = 1.75;

// Cadena (String)

let nombre = "Juan";

// Booleano (Boolean)

let esMayor = true;

// Nulo (null) e Indefinido (undefined)

let nulo = null;

let indefinido;

console.log(typeof edad); // Devolverá "number"

console.log(typeof nombre); // Devolverá "string"

console.log(typeof esMayor); // Devolverá "boolean"

console.log(typeof nulo); // Devolverá "object"

console.log(typeof indefinido); // Devolverá "undefined"

### Conversión entre Tipos de Datos:

### 

JavaScript es un lenguaje de tipado débil, lo que significa que puede realizar conversiones automáticas entre diferentes tipos de datos. Sin embargo, también puedes realizar conversiones explícitas.

* Convertir a Cadena (String):

let numero = 42;

let cadena = String(numero);

* Convertir a Número (Number):

let texto = "123";

let numero = Number(texto);

* Convertir a Booleano (Boolean):

let valor = 0;

let esVerdadero = Boolean(valor); // Devolverá false

Estos conceptos son fundamentales para entender cómo trabajar con variables y tipos de datos en JavaScript, proporcionando la base para manipular y gestionar información en tus programas.

#### CÓDIGO 2:

// Convertir a Cadena (String)

let numero = 42;

let cadena = String(numero);

// Convertir a Número (Number)

let texto = "123";

let numeroConvertido = Number(texto);

// Convertir a Booleano (Boolean)

let valor = 0;

let esVerdadero = Boolean(valor);

console.log(cadena); // Devolverá "42"

console.log(numeroConvertido); // Devolverá 123

console.log(esVerdadero); // Devolverá false

### 1 - A) EJERCICIOS Variables y Tipos de Datos:

1. Declarar una variable numérica y convertirla a cadena. Mostrar ambas variables en la consola.
2. Declarar una cadena que represente un número y convertirla a un valor numérico. Mostrar ambas variables en la consola.
3. Declarar una variable booleana y convertirla a número. Mostrar ambas variables en la consola.
4. Declarar una variable utilizando let e inicializarla como null. Luego, realizar la conversión a una cadena y mostrar el resultado.
5. Declarar una constante con tu edad y otra variable que represente tu altura. Concatenarlas en una cadena y mostrar el resultado en la consola.

## 1 - B) Operadores:

### Operadores aritméticos

* Suma (+):

let suma = 5 + 3; // resultado: 8

* Resta (-):

let resta = 10 - 4; // resultado: 6

* Multiplicación (\*):

let multiplicacion = 3 \* 7; // resultado: 21

* División (/):
* let division = 15 / 3; // resultado: 5

### Operadores de asignación

#### B.1. += (Adición y Asignación):

Este operador incrementa el valor de la variable por el valor proporcionado y luego asigna el resultado a la variable.

let y = 5;

y += 3; // Equivalente a: y = y + 3;

// Ahora, y es igual a 8

#### B.2. -= (Sustracción y Asignación):

Resta el valor proporcionado de la variable y luego asigna el resultado a la variable.

let z = 10;

z -= 4; // Equivalente a: z = z - 4;

// Ahora, z es igual a 6

#### B.3. \*= (Multiplicación y Asignación):

Multiplica el valor de la variable por el valor proporcionado y luego asigna el resultado a la variable.

let a = 2;

a \*= 3; // Equivalente a: a = a \* 3;

// Ahora, a es igual a 6

#### B.4. /= (División y Asignación):

Divide el valor de la variable por el valor proporcionado y luego asigna el resultado a la variable.

let b = 20;

b /= 4; // Equivalente a: b = b / 4;

// Ahora, b es igual a 5

### Propósito de los Operadores de Asignación:

Los operadores de asignación combinada ofrecen una forma concisa de realizar operaciones aritméticas y asignar el resultado a la misma variable. Ayudan a simplificar el código y hacen que sea más legible al evitar repeticiones innecesarias.

Estos operadores son especialmente útiles cuando se realiza una operación sobre una variable y se desea actualizar su valor sin tener que repetir el nombre de la variable. Además, pueden mejorar la eficiencia del código al realizar la operación en un solo paso.

### 

### CÓDIGO 3:

### Operadores de comparación:

#### B.5. Igualdad (==, ===):

== (Igualdad débil):

* + Compara los valores de dos operandos después de realizar conversiones de tipo si es necesario.
  + Si los valores son iguales en términos de valor, devuelve true.

Ejemplo:

let x = 5;

let y = "5";

console.log(x == y); // true (compara valores después de la conversión)

=== (Igualdad estricta):

* + Compara los valores y tipos de dos operandos sin realizar conversiones de tipo.
  + Devuelve true si los valores son idénticos en términos de valor y tipo.

Ejemplo:

let a = 5;

let b = "5";

console.log(a === b); // false (compara valores y tipos sin conversión)

#### B.6. Desigualdad (!=, !==):

* != (Desigualdad débil):
  + Compara los valores de dos operandos después de realizar conversiones de tipo si es necesario.
  + Devuelve true si los valores no son iguales en términos de valor.

Ejemplo:

let p = 10;

let q = "10";

console.log(p != q); // false (compara valores después de la conversión)

!== (Desigualdad estricta):

* + Compara los valores y tipos de dos operandos sin realizar conversiones de tipo.
  + Devuelve true si los valores o tipos no son idénticos.

Ejemplo:

let m = 10;

let n = "10";

console.log(m !== n); // true (compara valores y tipos sin conversión)

#### B.7 Relacionales (> , <, >=, <=):

> (Mayor que):

* + Devuelve true si el operando izquierdo es estrictamente mayor que el operando derecho.

let c = 8;

let d = 5;

console.log(c > d); // true

< (Menor que):

* + Devuelve true si el operando izquierdo es estrictamente menor que el operando derecho.

Ejemplo:

let e = 8;

let f = 12;

console.log(e < f); // true

>= (Mayor o igual que):

* + Devuelve true si el operando izquierdo es mayor o igual al operando derecho.

Ejemplo:

let g = 10;

let h = 10;

console.log(g >= h); // true

<= (Menor o igual que):

* + Devuelve true si el operando izquierdo es menor o igual al operando derecho.

Ejemplo:

let i = 15;

let j = 20;

console.log(i <= j); // true

### Propósito de los Operadores de Comparación:

Estos operadores se utilizan para comparar valores y determinar relaciones entre ellos en términos de igualdad, desigualdad y magnitud. Los operadores de igualdad y desigualdad son útiles para realizar comparaciones y tomar decisiones basadas en valores. Los operadores relacionales son esenciales para evaluar condiciones y controlar el flujo de un programa.

Es importante entender la diferencia entre igualdad débil (==), igualdad estricta (===), desigualdad débil (!=), desigualdad estricta (!==), y los operadores relacionales (>, <, >=, <=) para escribir código preciso y evitar comportamientos inesperados.