



DESARROLLO *WEB*

FULL STACK

NIVEL BÁSICO

Variables y Tipos de Datos en JavaScript



Variables y Tipos de Datos en JavaScript

Introducción a Variables y Tipos de Datos en JavaScript

JavaScript es un lenguaje de programación fundamental para el desarrollo web. Uno de los conceptos más importantes al aprender JavaScript es **el uso de variables y tipos de datos**, ya que nos permiten almacenar y manipular información dentro de nuestros programas.

Las variables son contenedores en los que podemos guardar datos, como números, textos o incluso listas y objetos. Dependiendo de cómo las declaremos, pueden cambiar o mantenerse constantes. Además, en JavaScript existen diferentes tipos de datos, los cuales determinan cómo se comportan y cómo podemos usarlos en nuestras aplicaciones.

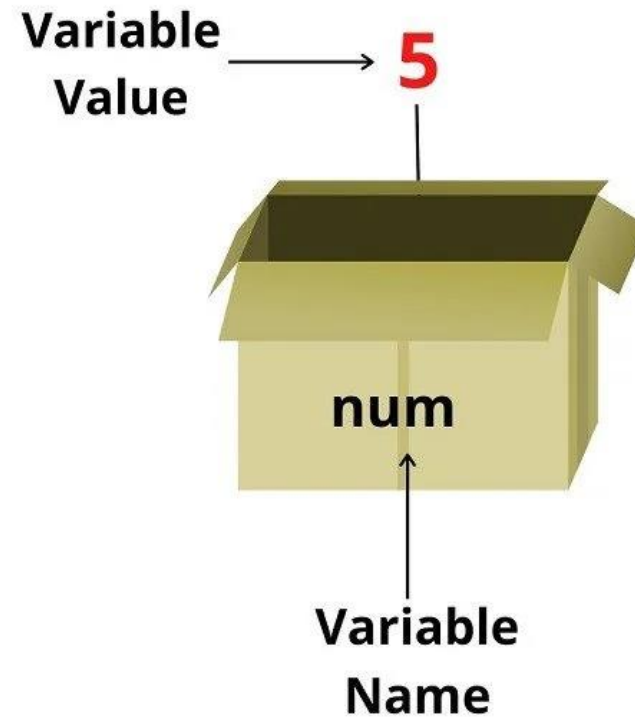
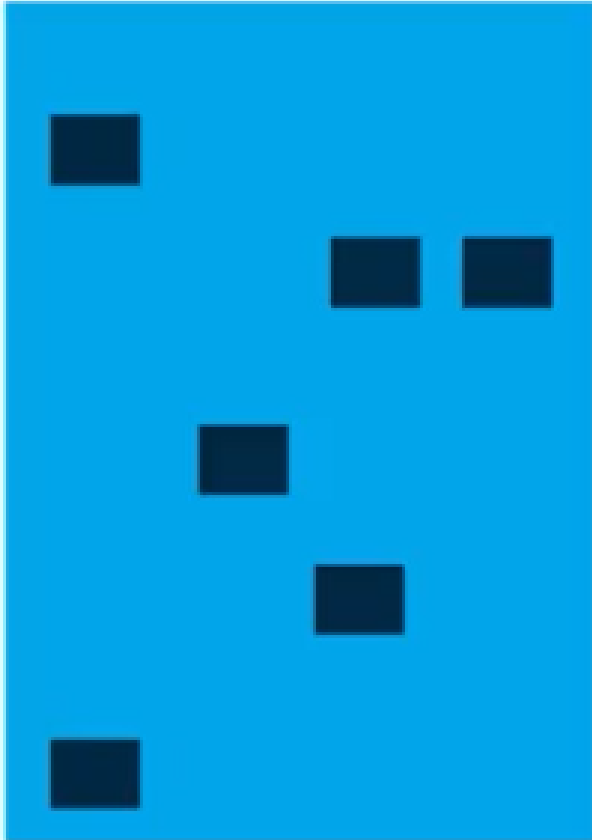
Entender bien estos conceptos es clave para escribir código eficiente y sin errores, ya que nos permitirá controlar mejor la información dentro de nuestros proyectos.

Tipos de Datos en JavaScript

En JavaScript, los tipos de datos definen la naturaleza de los valores que pueden almacenarse en las variables



Variables y Tipos de Datos en JavaScript





Declaración de variable

En JavaScript, existen tres formas de declarar variables:

1.1 `var` (Ya no recomendado)

- Fue la única forma de declarar variables en versiones antiguas de JavaScript.
- Tiene **scope de función**, lo que significa que su alcance se limita a la función en la que fue declarada.
- Puede ser redeclarada y reasignada dentro de su ámbito.

javascript

Copy Edit

```
var nombre = "Juan";  
console.log(nombre); // Juan  
var nombre = "Pedro"; // Redeclaración permitida  
console.log(nombre); // Pedro
```



Declaración de variable

1.2 `let` (Recomendada para variables que cambian de valor)

- Introducida en ES6 (ECMAScript 2015).
- Tiene **scope de bloque**, lo que significa que solo existe dentro del bloque `{}` donde se declaró.
- Puede ser reasignada, pero **no redeclarada** dentro del mismo ámbito.

javascript

Copy Edit

```
let edad = 25;  
console.log(edad); // 25  
edad = 30; // Reasignación permitida  
console.log(edad); // 30
```



Declaración de variable

1.3 `const` (Recomendada para valores constantes)

- También introducida en ES6.
- No puede ser reasignada ni redeclarada.
- Debe inicializarse en el momento de su declaración.
- Si el valor es un objeto o un array, sus propiedades pueden modificarse, pero la referencia no.

javascript

Copy Edit

```
const PI = 3.1416;
console.log(PI); // 3.1416
// PI = 3.14; ❌ Error: No se puede reasignar

const persona = { nombre: "Ana", edad: 28 };
persona.edad = 29; // ✅ Permitido, porque no estamos cambiando la referencia
console.log(persona.edad); // 29
```



Tipos de Datos en JavaScript

En JavaScript, los tipos de datos definen la naturaleza de los valores que pueden almacenarse en las variables. Se dividen en **tipos primitivos** y **tipos de datos complejos (objetos)**.

◆ Tipos Primitivos (Datos Inmutables)

Son los datos más básicos y almacenan valores directamente en la memoria.

1. `string` → Representa cadenas de texto.

javascript

Copy Edit

```
let texto = "Hola, JavaScript";
```



Tipos de Datos en JavaScript

2. `number` → Representa números enteros y decimales.

javascript

Copy Edit

```
let edad = 25;  
let precio = 9.99;
```

3. `bigint` → Se usa para representar números enteros extremadamente grandes.

javascript

Copy Edit

```
let numeroGrande = 12345678901234567890n;
```

4. `boolean` → Representa valores lógicos `true` o `false`.

javascript

Copy Edit

```
let esMayor = true;
```




Tipos de Datos en JavaScript

5. `undefined` → Indica que una variable ha sido declarada pero no tiene un valor asignado.

javascript

Copy Edit

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

6. `null` → Representa la ausencia intencionada de un valor.

javascript

Copy Edit

```
let vacio = null;
```

7. `symbol` → Se usa para crear identificadores únicos.

javascript

Copy Edit

```
let id = Symbol("identificador");
```



Tipos de Datos en JavaScript

◆ Tipos de Datos Complejos (Objetos)

Son estructuras que pueden contener múltiples valores y métodos.

1. `Object` → Estructura que almacena datos en pares clave-valor.

javascript

Copy Edit

```
let persona = { nombre: "Carlos", edad: 30 };
```

```
const persona = { nombre: "Ana", edad: 28 };
```

```
persona.edad = 29; //  Permitido, porque no estamos cambiando la referencia  
console.log(persona.edad); // 29
```



Tipos de Datos en JavaScript

2. **Array** → Lista ordenada de valores.

javascript

Copy Edit

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

3. **Function** → Bloque de código reutilizable.

javascript

Copy Edit

```
function suma(a, b) {  
  return a + b;  
}
```



Conversión de Tipos de Datos

◆ **Operaciones Matemáticas:** Son cálculos numéricos que incluyen suma (+), resta (-), multiplicación (*), división (/), módulo (%) y exponentes (**).

javascript

Copy Edit

```
console.log(10 + 5); // 15
console.log(8 * 3); // 24
console.log(10 % 3); // 1 (resto de la división)
```

◆ **Comparaciones:** Evalúan la relación entre valores y devuelven true o false. Incluyen igualdad (==, ===), desigualdad (!=, !==), mayor (>), menor (<), mayor o igual (>=), menor o igual (<=).

javascript

Copy Edit

```
console.log(5 === "5"); // false (compara tipo y valor)
console.log(10 > 5); // true
console.log(3 !== 2); // true
```

◆ **Concatenaciones:** Unen strings con el operador + o usando *template literals* con `\${}`.

javascript

Copy Edit

```
let nombre = "Juan";
console.log("Hola, " + nombre + "!"); // "Hola, Juan!"
console.log(`Hola, ${nombre}!`); // "Hola, Juan!" (mejor práctica)
```

◆ Conversión Implícita (Coerción de Tipos)

Conversión Implícita (Coerción de Tipos)

La coerción de tipos ocurre cuando JavaScript convierte automáticamente un tipo de dato en otro según el contexto de la operación. Puede ocurrir en operaciones matemáticas, comparaciones y concatenaciones.

◆ Coerción a String

Cuando un número o booleano se usa con un string, JavaScript lo convierte en texto.

javascript

Copy Edit

```
let resultado = "El año es " + 2025;
console.log(resultado); // "El año es 2025"

console.log("10" + true); // "10true"
console.log("5" + 2);      // "52" (convierte 2 en string)
```

◆ Conversión Implícita (Coerción de Tipos)

◆ Coerción a Number

Cuando un string que representa un número se usa en una operación matemática, JavaScript lo convierte automáticamente en un número.

javascript

Copy Edit

```
console.log("5" - 1); // 4 (convierte "5" en número)
console.log("10" * 2); // 20
console.log("8" / "2"); // 4
```

⚠ **Excepción:** La suma (+) prefiere concatenar en lugar de convertir.

javascript

Copy Edit

```
console.log("5" + 1); // "51" (concatenación, no suma)
```

◆ Conversión Implícita (Coerción de Tipos)

◆ Coerción a Booleano

Ciertos valores se consideran **falsos** (`false`) en un contexto booleano:

- **Falsos** (`false`): `0`, `""` (string vacío), `null`, `undefined`, `NaN`.
- **Verdaderos** (`true`): Cualquier otro valor.

javascript

Copy Edit

```
console.log(Boolean(0));           // false
console.log(Boolean(""));          // false
console.log(Boolean(100));          // true
console.log(Boolean("Hola"));       // true
console.log(Boolean([]));           // true (cualquier objeto es true)
```



Conversión Explícita (Casting de Tipos)

Cuando queremos convertir manualmente un tipo de dato a otro, usamos métodos específicos:

1 Convertir a Número

◆ Usando `Number()` → Convierte un valor a número.

javascript

Copy Edit

```
let num1 = Number("42");  
console.log(num1); // 42 (tipo: number)  
  
let num2 = Number(true);  
console.log(num2); // 1 (true se convierte en 1)  
  
let num3 = Number("Hola");  
console.log(num3); // NaN (No es un número)
```

NaN (Not a Number) es un valor especial en JavaScript que indica que una operación matemática o conversión numérica ha fallado y no produce un número válido.



Conversión Explícita (Casting de Tipos)

◆ Usando `parseInt()` → Convierte un string a un número entero.

javascript

Copy Edit

```
let entero = parseInt("100.99");  
console.log(entero); // 100 (ignora decimales)
```

◆ Usando `parseFloat()` → Convierte un string a un número decimal.

javascript

Copy Edit

```
let decimal = parseFloat("100.99");  
console.log(decimal); // 100.99
```



int vs float en JavaScript

◆ **int (Entero):** Número sin decimales, se usa cuando no se necesitan fracciones.



Ejemplo: Contadores, índices, edades.

◆ **float (Flotante):** Número con decimales, se usa para cálculos precisos con fracciones.



Ejemplo: Precios, medidas, promedios.



Conversión Explícita (Casting de Tipos)

2 Convertir a String

◆ Usando `String()` → Convierte cualquier dato en string.

javascript

Copy Edit

```
let str1 = String(123);
console.log(str1); // "123" (tipo: string)

let str2 = String(true);
console.log(str2); // "true"

let str3 = String(null);
console.log(str3); // "null"
```

◆ Usando `.toString()` → Convierte un número o booleano a string.

javascript

Copy Edit

```
let num = 456;
console.log(num.toString()); // "456"

let bool = false;
console.log(bool.toString()); // "false"
```



Conversión Explícita (Casting de Tipos)

3 Convertir a Booleano

◆ Usando `Boolean()` → Convierte valores en `true` o `false`.

javascript

Copy Edit

```
let bool1 = Boolean(1);
console.log(bool1); // true

let bool2 = Boolean(0);
console.log(bool2); // false

let bool3 = Boolean("Hola");
console.log(bool3); // true

let bool4 = Boolean("");
console.log(bool4); // false (string vacío es falso)
```



Formas de Usar JavaScript en HTML

En HTML, JavaScript se puede incluir de varias maneras

1 Código JavaScript en Línea (`inline`)

Se escribe directamente dentro de los atributos `onclick`, `onmouseover`, etc. en una etiqueta HTML.

html

Copy Edit

```
<button onclick="alert('¡Hola, mundo!')">Haz clic aquí</button>
```

✓ Útil para acciones rápidas, pero no recomendable en código grande.



Concepto de Evento en JavaScript

Un evento en JavaScript es una acción o suceso que ocurre en el navegador, como un clic del usuario, el movimiento del mouse, la pulsación de una tecla, o la carga de una página



Formas de Usar JavaScript en HTML

Código JavaScript Interno (<script> en HTML)

Se coloca dentro de la etiqueta <script> en el mismo archivo HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>JS Interno</title>
</head>
<body>
  <button onclick="saludar()">Saludar</button>

  <script>
    function saludar() {
      alert(";Hola desde JavaScript interno!");
    }
  </script>
</body>
</html>
```



Formas de Usar JavaScript en HTML

Código JavaScript Externo (<script src="archivo.js">)

Se guarda el código en un archivo separado (script.js) y se enlaza en el HTML.

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <title>JS Externo</title>
5      <script src="script.js"></script>
6  </head>
7  <body>
8      <button onclick="mostrarMensaje()">Mostrar Mensaje</button>
9  </body>
10 </html>
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejemplo DOM</title>
5  </head>
6  <body>
7      <button id="saludo">Haz clic aquí</button>
8      <p id="mensaje"></p>
9      <script src="script.js"></script>
10 </body>
11 </html>
```

Archivo script.js

javascript

Copy Edit

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



Formas de Usar JavaScript en HTML

4 JavaScript en la Consola del Navegador

Puedes ejecutar JavaScript directamente en la consola de herramientas para desarrolladores (**F12** o **Ctrl + Shift + I**).

javascript

Copy Edit

```
console.log("Hola desde la consola");
```

Útil para pruebas rápidas y depuración.



DOM y su Manipulación en JavaScript

El DOM (Document Object Model) es como un puente entre HTML y JavaScript.

Imagina que la página web es un árbol gigante donde cada elemento (como títulos, párrafos, botones, imágenes) es una rama.

JavaScript usa el DOM para encontrar esas ramas y modificarlas en tiempo real. Así puedes cambiar textos, colores, imágenes o incluso agregar y eliminar elementos sin necesidad de recargar la página.

El DOM (Document Object Model) es una representación en forma de árbol de todos los elementos de una página web.

JavaScript puede acceder y modificar estos elementos para hacer que la página sea interactiva.



DOM y su Manipulación en JavaScript

Ejemplo Practico, si tienes este botón en HTML:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejemplo DOM</title>
5      <script src="script.js"></script>
6  </head>
7  <body>
8      <button id="saludo">Haz clic aquí</button>
9      <p id="mensaje"></p>
10 </body>
11 </html>
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejemplo DOM</title>
5  </head>
6  <body>
7      <button id="saludo">Haz clic aquí</button>
8      <p id="mensaje"></p>
9      <script src="script.js"></script>
10 </body>
11 </html>
12
```

```
document.getElementById("saludo").addEventListener("click", function() {
    document.getElementById("mensaje").textContent = "¡Hola, DOM!";
});
```

◆ 2. Seleccionar Elementos del DOM

◆ 2. Seleccionar Elementos del DOM

Para manipular el DOM, primero debemos seleccionar los elementos con JavaScript. Existen varias formas: **Crea un archivo index, css y js para hacer los ejemplos**

```
1  function pruebas_dom () {  
2  
3  }
```

📌 2.1. `getElementById()` - Selecciona por ID

Devuelve el elemento único con el `id` especificado.

```
javascript Copy Edit  
  
var titulo = document.getElementById("titulo");  
console.log(titulo.textContent); // Muestra "Hola, DOM"
```



DOM y su Manipulación en JavaScript



2.2. `getElementsByClassName()` - Selecciona por Clase

Devuelve una colección (`HTMLCollection`) de elementos con la misma clase.

javascript

Copy Edit

```
var textos = document.getElementsByClassName("texto");  
console.log(textos[0].textContent); // Muestra "Este es un párrafo."
```



2.3. `getElementsByTagName()` - Selecciona por Etiqueta

Devuelve todos los elementos de un tipo de etiqueta (por ejemplo, todos los `<p>`).

javascript

Copy Edit

```
var parrafos = document.getElementsByTagName("p");  
console.log(parrafos[0].textContent);
```



DOM y su Manipulación en JavaScript



2.4. `querySelector()` - Selecciona el Primer Elemento que Coincida

Usa selectores CSS para seleccionar el primer elemento que coincida.

javascript

Copy Edit

```
var primerParrafo = document.querySelector(".texto");  
console.log(primerParrafo.textContent);
```



2.5. `querySelectorAll()` - Selecciona Todos los Elementos que Coincidan

Devuelve una lista de nodos (NodeList) con todos los elementos que coincidan con el selector CSS.

javascript

Copy Edit

```
var todosLosParrafos = document.querySelectorAll("p");  
console.log(todosLosParrafos.length); // Cantidad de párrafos
```



Modificar Elementos del DOM

◆ 3. Modificar Elementos del DOM

Una vez seleccionado un elemento, podemos modificarlo:

📌 3.1. Modificar el Contenido de un Elemento

Usamos `.textContent` o `.innerHTML`:

javascript

Copy Edit

```
var titulo = document.getElementById("titulo");  
titulo.textContent = "Nuevo Título"; // Cambia el texto  
titulo.innerHTML = "<span style='color:red;'>Nuevo Título</span>"; // Permite HTML
```

📌 3.2. Modificar Atributos de un Elemento

Usamos `.setAttribute()` y `.getAttribute()`:

javascript

Copy Edit

```
var imagen = document.getElementById("miImagen");  
imagen.setAttribute("src", "nueva-imagen.jpg"); // Cambia la imagen  
console.log(imagen.getAttribute("src")); // Obtiene el valor del atributo
```





DOM y su Manipulación en JavaScript

3.3. Modificar Estilos CSS

Podemos cambiar los estilos con `.style`:

javascript

 Copy  Edit

```
var titulo = document.getElementById("titulo");
titulo.style.color = "blue";
titulo.style.fontSize = "24px";
```

◆ 4. Agregar y Eliminar Elementos del DOM

Podemos crear, insertar y eliminar elementos en tiempo real.

📌 4.1. Crear un Nuevo Elemento

javascript

📄 Copy 🖋 Edit

```
var nuevoParrafo = document.createElement("p");
nuevoParrafo.textContent = "Este es un nuevo párrafo.";
document.body.appendChild(nuevoParrafo); // Lo agrega al final del <body>
```

📌 4.2. Insertar un Elemento antes de Otro

javascript

📄 Copy 🖋 Edit

```
var contenedor = document.getElementById("contenedor");
var primerElemento = contenedor.firstChild;
contenedor.insertBefore(nuevoParrafo, primerElemento);
```

◆ 4. Agregar y Eliminar Elementos del DOM

📌 4.3. Eliminar un Elemento

javascript

📄 Copy 🖋 Edit

```
var eliminarParrafo = document.getElementById("parrafoAEliminar");  
eliminarParrafo.remove();
```


◆ 5. Eventos en el DOM

Los eventos permiten que los elementos respondan a acciones del usuario (clics, teclas, movimientos del mouse, etc.).

📌 5.1. Evento `onclick`

javascript

Copy Edit

```
var boton = document.getElementById("miBoton");
boton.onclick = function() {
    alert("¡Clic en el botón!");
};
```

📌 5.2. Evento con `addEventListener()`

javascript

Copy Edit

```
document.getElementById("miBoton").addEventListener("click", function() {
    alert("Usando addEventListener");
});
```