

const

=>

[ ]

CUADERNILLO DE ESTUDIO · NIVEL BÁSICO

JavaScript

fundamentos

Variables, operadores, funciones, arreglos, objetos y la página web: lo esencial del lenguaje de la web, con ejemplos para probar en el navegador.

1 Variables y tipos

2 Operadores

3 Condicionales, bucles y funciones

4 Arreglos

5 Objetos, textos y DOM

+ Ejercicios con respuestas

Testing-US · Validá tu aprendizaje
testing-us.com

Edición octubre 2026

Antes de empezar

JavaScript es el lenguaje que hace que las páginas web reaccionen: valida formularios, muestra datos sin recargar y responde a cada clic. También se usa en servidores (Node.js) y en aplicaciones. Saber sus fundamentos es el primer paso para cualquier puesto de desarrollo web.

CÓMO APROVECHARLO

- Probá cada ejemplo en la consola del navegador (F12) y cambiá los valores para ver qué pasa.
- Hacé los ejercicios del final de cada tema antes de mirar las respuestas.
- Cuando termines, rendí la evaluación **JavaScript: fundamentos** en testing-us.com.

TIP Antes de ejecutar un ejemplo, tratá de adivinar qué va a mostrar. Así se entrena justo lo que pide la evaluación.

Contenido

1 **Variables y tipos de datos**
const, let y var, tipos primitivos, typeof y alcance.

2 **Operadores y comparaciones**
Aritmética, conversiones, == y ===, truthy y falsy, ||, && y ??.

3 **Condicionales, bucles y funciones**
if y switch, for y while, parámetros, return, funciones flecha.

4 **Arreglos**
Índices, push y pop, slice y splice, map, filter, reduce y sort.

5 **Objetos, textos y la página (DOM)**
Propiedades, referencias, JSON, métodos de texto, querySelector y eventos.

TEMA 1

1 Variables y tipos de datos

LO QUE VAS A APRENDER

- Declarar variables con const y let.
- Reconocer los tipos de datos primitivos.
- Usar typeof y entender sus sorpresas.
- Saber dónde existe cada variable (alcance).

Declarar variables

- Una **variable** es un nombre que guarda un valor. Se declara con `const`, `let` o `var`.
- `const` declara una variable que **no se puede reasignar**: `const PI = 3.14;`. Si después escribís `PI = 3;`, da un error (`TypeError`). Además, `const` siempre necesita un valor inicial.
- `let` declara una variable que **sí se puede reasignar**: `let edad = 30; edad = 31;`.
- `var` es la forma antigua. Hoy se recomienda usar `const` **por defecto** y `let` solo cuando el valor va a cambiar.
- Ojo: `const` impide reasignar la variable, pero si guarda un objeto o un arreglo, **su contenido sí se puede cambiar**: `const p = { n: 1 }; p.n = 2;` funciona.

Nombres de variables

- Pueden tener letras, números, `_` y `$`, pero **no pueden empezar con un número** ni tener espacios o guiones.
- No se pueden usar palabras reservadas del lenguaje como nombre (`let`, `if`, `function`, etc.).
- JavaScript **distingue mayúsculas de minúsculas**: `edad` y `Edad` son dos variables distintas.
- Se acostumbra escribir en *camelCase*: `nombreCompleto`, `precioFinal`.

Tipos de datos

Los valores **primitivos** más usados son:

- **string** (texto): `"hola"`, `'hola'`. Las comillas simples y las dobles sirven igual.
- **number** (número): `42`, `3.5`, `-7`. Enteros y decimales son del mismo tipo. También son number los valores especiales `NaN` ("no es un número") e `Infinity`.
- **boolean** (lógico): `true` o `false`.
- **undefined**: el valor de una variable declarada a la que todavía no se le asignó nada.
- **null**: "sin valor", puesto a propósito por quien programa.

Además están los **objetos** (incluidos los arreglos y las funciones).



Figura 1.1 Los tipos primitivos y lo que devuelve typeof.

El operador typeof

`typeof` devuelve, como texto, el tipo de un valor:

- `typeof "hola"` da `"string"`; `typeof 42` da `"number"`; `typeof true` da `"boolean"`; `typeof undefined` da `"undefined"`.
- `typeof null` da `"object"`: es un error histórico del lenguaje que se mantiene por compatibilidad.
- `typeof [1, 2]` da `"object"` (para saber si algo es un arreglo se usa `Array.isArray`), y `typeof NaN` da `"number"`.

Alcance (scope)

- `let` y `const` tienen **alcance de bloque**: solo existen dentro de las llaves `{}` donde se declararon. Fuera de ese bloque, usarlas da `ReferenceError`.
- `var` tiene alcance de **función** (ignora los bloques de `if` o `for`) y se "eleva" (*hoisting*): si se usa antes de su línea, vale `undefined`.
- Usar una variable `let` o `const` antes de que se ejecute su declaración da `ReferenceError`.

```
let nombre = "Ana";                          existe en todo el archivo

if (true) {                                  bloque: saludo vive solo acá
    let saludo = "Hola " + nombre; // ✓ ve a nombre
}

console.log(saludo); // ✗ ReferenceError
```

Figura 1.2 Una variable let declarada en un bloque no existe fuera de él.

Textos y comentarios

- La propiedad `length` da la cantidad de caracteres: `"hola".length` es `4`.
- Las **plantillas de texto** (*template literals*) se escriben entre comillas invertidas y permiten insertar valores con `${...}`, por ejemplo para armar "Hola, Ana" a partir de una variable.
- Los comentarios de una línea empiezan con `//`; los de varias líneas van entre `/*` y `*/`. El navegador los ignora.

TIP Abrí la consola del navegador (F12, pestaña Consola) y probá cada ejemplo de este cuadernillo: es la mejor manera de aprender.

Para practicar

1. ¿Qué diferencia hay entre const y let? ¿Cuál conviene usar por defecto?
2. ¿Qué dan typeof null y typeof [1, 2]?
3. ¿Por qué funciona const lista = []; lista.push(1); si lista es const?
4. ¿Qué pasa si usás una variable let fuera del bloque donde se declaró?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 2

2 Operadores y comparaciones

LO QUE VAS A APRENDER

- Usar los operadores aritméticos, incluido el resto (%).
- Entender cuándo + concatena.
- Comparar con === y saber por qué no conviene ==.
- Reconocer los valores falsy y usar ||, && y ??.

Operadores aritméticos

- Suma `+`, resta `-`, multiplicación `*`, división `/`, **resto** `%` y potencia `**`.
- `10 % 3` da `1` (el resto de dividir 10 por 3). `2 ** 3` da `8`.
- Asignación con operación: `x += 2` es lo mismo que `x = x + 2`. `i++` suma 1 a `i`.

El + con textos y las conversiones

- Si uno de los dos lados del `+` es un texto, `+` **concatena**: `5 + "5"` da `"55"`.
- Los demás operadores aritméticos convierten el texto a número: `"10" - 2` da `8` y `"3" * 2` da `6`.
- `Number("12")` da `12`, pero `Number("12a")` da `NaN`. `parseInt("42px")` lee los dígitos del principio y da `42`.
- `NaN` no es igual a nada, ni siquiera a sí mismo: `NaN === NaN` es `false`. Para detectarlo se usa `Number.isNaN(x)`.
- Los decimales tienen pequeñas imprecisiones: `0.1 + 0.2 === 0.3` es `false` (da 0.30000000000000004).

Comparar: == y ===

- `===` (igualdad **estricta**) compara valor y tipo, sin convertir: `5 === "5"` es `false`.
- `==` (igualdad **flexible**) convierte los tipos antes de comparar: `5 == "5"` es `true`. Un caso especial: `null == undefined` es `true`.
- Se recomienda usar **siempre** `===` y `!==`, porque las conversiones de `==` llevan a errores difíciles de ver.
- También existen `<`, `>`, `<=` y `>=`.

== (flexible): convierte tipos

```
5 == "5"              true
0 == false            true
null == undefined     true
NaN == NaN            false
```

=== (estricta): recomendada

```
5 === "5"             false
0 === false           false
null === undefined    false
NaN === NaN           false
```

Figura 2.1 La igualdad flexible convierte tipos; la estricta, no.

Verdadero y falso (truthy y falsy)

- En un `if` o con operadores lógicos, cualquier valor se interpreta como verdadero o falso.
- Los valores **falsy** son `false`, `0` (y `-0`), `""` (texto vacío), `null`, `undefined` y `NaN` (además de `0n`, el cero de BigInt).
- Todo lo demás es **truthy**, incluso `"0"`, `"false"`, `[]` (arreglo vacío) y `{}` (objeto vacío).
- `!` niega: `!true` es `false`. `!!valor` convierte cualquier valor en su booleano: `!!"texto"` es `true`.

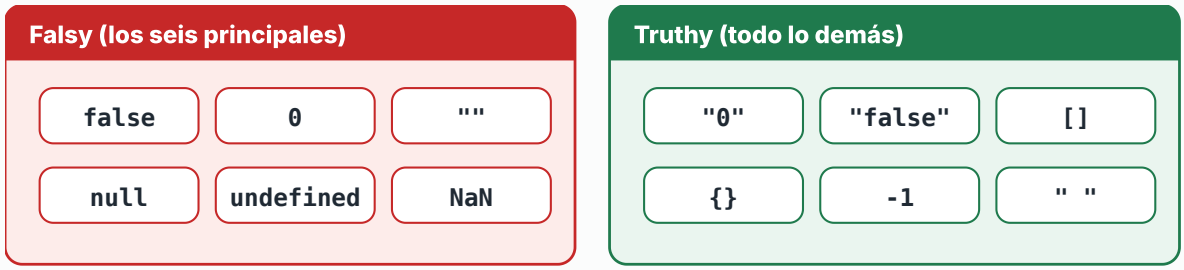


Figura 2.2 Los valores falsy más comunes; todo lo demás es truthy.

Operadores lógicos y de valor por defecto

- `a && b`: si `a` es falsy, devuelve `a`; si no, devuelve `b`. `true && "hola"` da `"hola"`.
- `a || b`: si `a` es truthy, devuelve `a`; si no, devuelve `b`. `0 || 5` da `5`.
- `a ?? b` (*nullish*): devuelve `b` **solo si** `a` es `null` o `undefined`. `0 ?? 5` da `0`, y `null ?? "sin dato"` da `"sin dato"`.
- Operador **ternario**: `condición ? valorSiSí : valorSiNo`. Ejemplo: `nota >= 6 ? "aprobado" : "desaprobado"`.

OJO `||` reemplaza también el 0 y el texto vacío. Si 0 es un valor válido (por ejemplo, una cantidad), usá `??`.

Para practicar

1. ¿Qué dan `5 + "5"`, `"5" * 2` y `7 % 2`?
2. ¿Por qué `NaN === NaN` da false? ¿Cómo se detecta NaN?
3. Nombrá los seis valores falsy más comunes.
4. ¿Qué dan `0 || 10` y `0 ?? 10`?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 3

3 Condicionales, bucles y funciones

LO QUE VAS A APRENDER

- Elegir entre if, else if y switch.
- Escribir bucles for, while y for...of sin quedar en un bucle infinito.
- Definir funciones con parámetros, valores por defecto y return.
- Escribir funciones flecha y pasarlas como callback.

Condicionales

- `if (condición) { ... } else if (otra) { ... } else { ... }`: se ejecuta **el primer bloque** cuya condición sea verdadera.
- `switch` compara un valor con varios `case`. Cada `case` **suele** terminar con `break`; si falta, la ejecución **sigue con el siguiente** `case` (se "cae").
- La condición puede ser cualquier valor: `if ("")` no entra, porque el texto vacío es falsy.

Bucles

- `for (let i = 0; i < 5; i++) { ... }` repite el bloque 5 veces (i vale 0, 1, 2, 3 y 4).
- `while (condición) { ... }` repite mientras la condición sea verdadera; si es falsa desde el principio, no se ejecuta nunca. Si dentro no se cambia nada que haga falsa la condición, el bucle es **infinito**.
- `do { ... } while (condición)` ejecuta el bloque **al menos una vez** y después evalúa.
- `for (const x of arreglo)` recorre los **valores**; `for (const k in objeto)` recorre las **claves** (en un arreglo, los índices como texto: "0", "1"...).
- `break` termina el bucle completo; `continue` saltea lo que queda de esa vuelta y pasa a la siguiente.

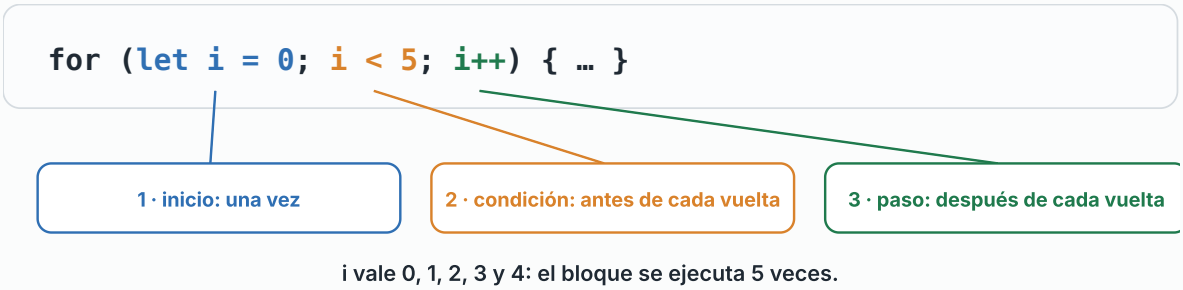


Figura 3.1 Las tres partes de un bucle for.

Funciones

- Una función agrupa código para reutilizarlo: `function doble(n) { return n * 2; }`. Al llamarla, `doble(4)` da `8`.
- En la definición, `n` es el **parámetro**; en la llamada, `4` es el **argumento**.
- `return` devuelve un valor y **termina** la función: lo que está después no se ejecuta. Una función sin `return` devuelve `undefined`.

- **Parámetros por defecto**: `function saludar(nombre = "invitado")` usa "invitado" si no se pasa un argumento.
- Las variables declaradas dentro de una función no existen fuera de ella.
- Las funciones declaradas con `function nombre() {}` se pueden llamar **antes** de la línea donde están escritas (*hoisting*).

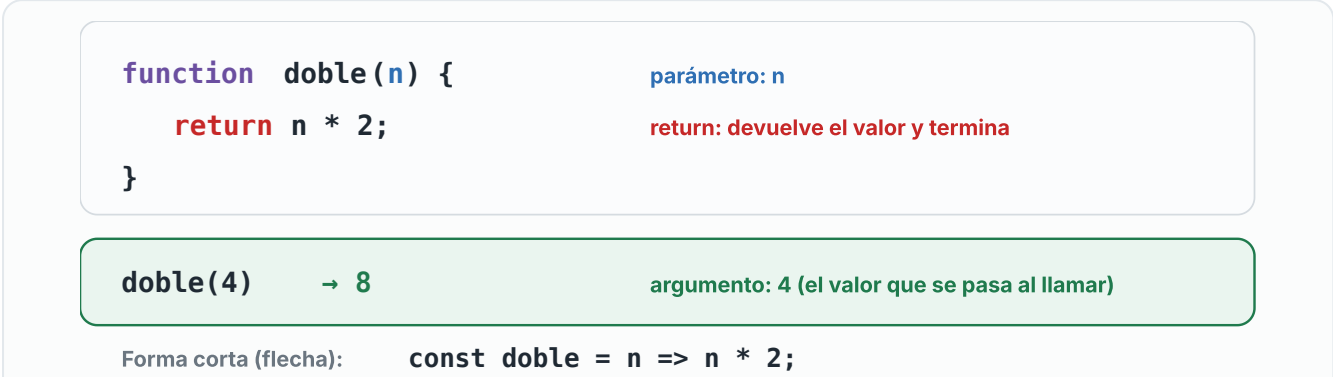


Figura 3.2 Partes de una función y su forma corta.

Funciones flecha y funciones como valores

- Forma corta: `const cuadrado = n => n * n;`. Con un solo parámetro no hacen falta paréntesis; si el cuerpo es una sola expresión, no hacen falta llaves ni `return`.
- Con varios parámetros: `(a, b) => a + b`. Si el cuerpo tiene varias instrucciones, se usan llaves y `return`.
- Las funciones son valores: se pueden guardar en variables y **pasar como argumento** a otra función. A esa función que se pasa se la llama *callback*.
- Ejemplo: `setTimeout(avisar, 2000)` ejecuta la función `avisar` una vez, después de 2 segundos (2000 milisegundos).

Para practicar

1. ¿Qué pasa en un switch si un case no tiene break?
2. ¿Qué diferencia hay entre for...of y for...in en un arreglo?
3. ¿Qué devuelve una función sin return?
4. Escribí como función flecha: function suma(a, b) { return a + b; }

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 4

4 Arreglos

LO QUE VAS A APRENDER

- Leer elementos por su índice.
- Agregar y quitar elementos.
- Buscar con includes, indexOf y find.
- Transformar con map, filter y reduce.

Crear y leer arreglos

- Un **arreglo** (*array*) es una lista ordenada: `const colores = ["rojo", "verde", "azul"];`.
- Las posiciones (**índices**) empiezan en **0**: `colores[0]` es `"rojo"`.
- `colores.length` es `3`; el último elemento es `colores[colores.length - 1]` (también `colores.at(-1)`).
- Leer una posición que no existe da `undefined`, sin error.
- `typeof` de un arreglo da `"object"`; para comprobar que algo es un arreglo se usa `Array.isArray(x)`.

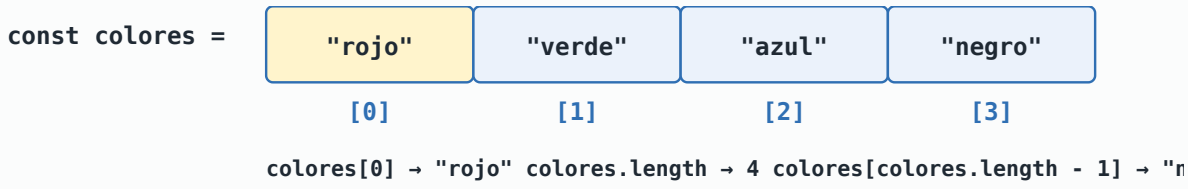


Figura 4.1 Los índices empiezan en 0.

Agregar y quitar

- `push(x)` agrega al **final**; `pop()` quita y devuelve el **último**.
- `unshift(x)` agrega al **principio**; `shift()` quita y devuelve el **primero**.
- `splice(inicio, cantidad)` quita (o reemplaza) elementos **modificando el arreglo original**.
- `slice(inicio, fin)` devuelve una **copia** de una parte, **sin modificar** el original; el elemento en `fin` no se incluye: `[10, 20, 30, 40].slice(1, 3)` da `[20, 30]`.

Buscar

- `includes(x)` dice si está (`true` o `false`).
- `indexOf(x)` da la posición de la primera aparición, o `-1` si no está.
- `find(condición)` devuelve el **primer elemento** que cumple la condición (o `undefined`).

Recorrer y transformar

- `forEach(fn)` ejecuta la función para cada elemento y **devuelve** `undefined`: sirve para hacer algo con cada uno, no para crear un arreglo nuevo.
- `map(fn)` devuelve un **arreglo nuevo** con el resultado de aplicar la función a cada elemento: `[1, 2, 3].map(n => n * 2)` da `[2, 4, 6]`.
- `filter(fn)` devuelve un arreglo nuevo **solo con los elementos que cumplen** la condición: `[5, 12, 8, 20].filter(n => n > 10)` da `[12, 20]`.

- `reduce(fn, inicial)` **acumula** todos los elementos en un solo valor: `[1, 2, 3, 4].reduce((acc, n) => acc + n, 0)` da `10`.
- `map`, `filter` y `slice` no modifican el arreglo original.

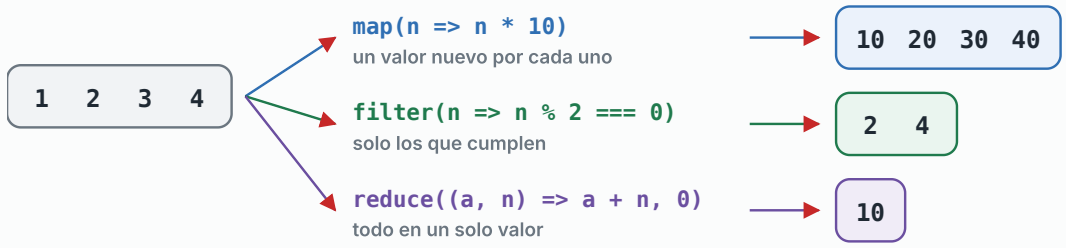


Figura 4.2 El mismo arreglo pasando por map, filter y reduce.

Otras operaciones útiles

- `join(separador)` une los elementos en un texto: `["a", "b", "c"].join("-")` da `"a-b-c"`.
- El operador **spread** `...` "despliega" los elementos: `[...[1, 2], 3]` da `[1, 2, 3]`. Sirve para copiar o combinar arreglos.
- `sort()` sin argumentos ordena **como texto**: `[10, 9, 1].sort()` da `[1, 10, 9]`. Para números se pasa una función: `sort((a, b) => a - b)`. `sort` modifica el arreglo original.

OJO `splice`, `sort`, `push`, `pop`, `shift` y `unshift` modifican el arreglo original; `slice`, `map` y `filter` devuelven uno nuevo.

Para practicar

1. ¿Cómo leés el último elemento de cualquier arreglo?
2. ¿Qué diferencia hay entre slice y splice?
3. Con precios = [100, 250, 80], escribí cómo obtener los precios con 21 % de IVA.
4. ¿Qué da [10, 9, 1].sort() y cómo se ordenan números de menor a mayor?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 5

5 Objetos, textos y la página (DOM)

LO QUE VAS A APRENDER

- Crear objetos y leer sus propiedades.
- Entender que los objetos se copian por referencia.
- Convertir entre objetos y JSON; usar métodos de texto.
- Seleccionar elementos y responder a un clic.

Objetos

- Un **objeto** agrupa datos en pares **clave: valor**: `const persona = { nombre: "Ana", edad: 30 };`.
- Se accede con punto, `persona.nombre`, o con corchetes, `persona["nombre"]`. Cuando la clave está guardada en una variable, se usan **corchetes**: `persona[clave]`.
- Leer una propiedad que no existe da `undefined`.
- Un objeto puede tener **métodos** (funciones como propiedades): `const o = { saludar() { return "hola"; } };` se usa como `o.saludar()`.
- `Object.keys(obj)` da un arreglo con las claves: `Object.keys({ a: 1, b: 2 })` da `["a", "b"]`.
- **Desestructuración**: `const { nombre } = persona;` crea la variable `nombre` con el valor `"Ana"`.

Referencias

- Las variables no guardan el objeto en sí, sino una **referencia** a él. Con `const b = a;` las dos variables apuntan al **mismo** objeto: si se cambia `b.n`, también cambia `a.n`.
- Dos objetos distintos nunca son iguales con `===`, aunque tengan el mismo contenido: `{ a: 1 } === { a: 1 }` es `false`.
- Para copiar un objeto (en un nivel) se usa spread: `const copia = { ...original };`.

TIP Si necesitás modificar un objeto sin tocar el original, primero copialo: `const copia = { ...original };`

JSON

- **JSON** es un formato de texto para intercambiar datos (por ejemplo, con un servidor).
- `JSON.stringify(obj)` convierte un objeto en texto: `JSON.stringify({ a: 1 })` da `'{"a":1}'`.
- `JSON.parse(texto)` hace lo contrario: convierte un texto JSON en objeto.

Métodos de texto

- Los textos (strings) son **inmutables**: sus métodos devuelven un texto nuevo y no se puede cambiar una letra con `s[0] = "H"`: la asignación se ignora (o da `TypeError` en modo estricto).
- `toUpperCase()` y `toLowerCase()` pasan a mayúsculas o minúsculas.
- `trim()` quita los espacios del principio y del final.
- `split(separador)` corta el texto en un arreglo: `"a,b,c".split(",")` da `["a", "b", "c"]`.
- `includes(x)` dice si contiene un texto; distingue mayúsculas y minúsculas.

La página: el DOM

- El navegador convierte el HTML en un árbol de objetos llamado **DOM** (*Document Object Model*). JavaScript lo usa para leer y cambiar la página.
- `document.querySelector("selector")` devuelve el **primer** elemento que coincide con un selector CSS: `"#titulo"` busca por id, `".boton"` por clase. `querySelectorAll` devuelve todos. Si no encuentra ninguno, `querySelector` devuelve `null`.
- `elemento.textContent = "Nuevo texto"` cambia el texto de un elemento.
- `elemento.addEventListener("click", fn)` hace que `fn` se ejecute **cada vez** que se hace clic en el elemento.
- `console.log(valor)` muestra valores en la **consola** del navegador (F12): es la herramienta básica para revisar qué hace el código.

```
<body>
<h1 id="titulo">Hola</h1>
<button class="boton">
Enviar
</button>
</body>
```

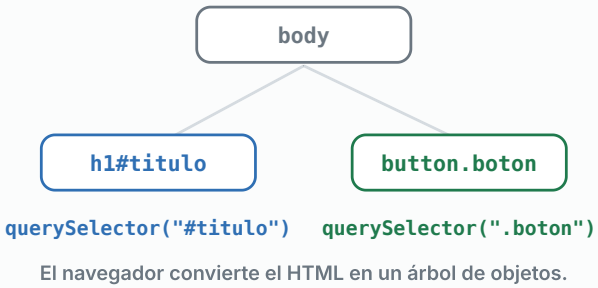


Figura 5.1 Un HTML chico, su árbol DOM y cómo seleccionar elementos.

Para practicar

1. ¿Cuándo hace falta usar corchetes para leer una propiedad?
2. Si b = a y cambiás b.n, ¿qué pasa con a.n? ¿Por qué?
3. ¿Qué hacen JSON.stringify y JSON.parse?
4. Escribí el código para que, al hacer clic en el botón .boton, el título #titulo diga “Enviado”.

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

PARA CONTROLAR

Respuestas de los ejercicios

Tema 1

1. const no se puede reasignar; let sí. Conviene usar const por defecto y let solo si el valor cambia.
2. Las dos dan "object" (lo de null es un error histórico; para arreglos se usa Array.isArray).
3. Porque const impide reasignar la variable, no cambiar el contenido del arreglo.
4. Da ReferenceError: let solo existe dentro del bloque donde se declaró.

Tema 2

1. "55", 10 y 1.
2. NaN no es igual a nada, ni a sí mismo. Se detecta con Number.isNaN(x).
3. false, 0, "" (texto vacío), null, undefined y NaN.
4. 0 || 10 da 10 (0 es falsy); 0 ?? 10 da 0 (0 no es null ni undefined).

Tema 3

1. Sigue ejecutando el case siguiente, hasta encontrar un break o el final.
2. for...of recorre los valores; for...in, los índices (como texto).
3. undefined.
4. const suma = (a, b) ⇒ a + b;

Tema 4

1. arr[arr.length - 1].
2. slice devuelve una copia de una parte sin modificar el original; splice quita o reemplaza en el mismo arreglo.
3. precios.map(p ⇒ p * 1.21) → [121, 302.5, 96.8].
4. Da [1, 10, 9] porque ordena como texto. Para números: sort((a, b) ⇒ a - b).

Tema 5

1. Cuando el nombre de la clave está en una variable o no es un nombre válido: obj[clave].
2. a.n también cambia: las dos variables apuntan al mismo objeto.
3. stringify convierte un objeto en texto JSON; parse convierte un texto JSON en objeto.
4. document.querySelector(".boton").addEventListener("click", () ⇒ { document.querySelector("#titulo").textContent = "Enviado"; });



Validá lo que aprendiste

Rendí la evaluación **JavaScript: fundamentos** en **testing-us.com**. Es gratis: en cada intento te tocan 10 preguntas al azar, dos de cada tema de este cuadernillo. Tenés 15 minutos y aprobás con 70%.

Si aprobás, recibís un certificado con código de verificación que podés sumar a tu currículum y compartir con quien te contrate.

MIS NOTAS

© 2026 Testing-US. Material de estudio propio. Se permite imprimirlo y compartirlo sin modificaciones y sin fines comerciales.

Referencias: especificación ECMAScript (Ecma International, ECMA-262) y documentación de JavaScript de MDN Web Docs (developer.mozilla.org). Los textos y dibujos de este cuadernillo son propios; ninguna de estas fuentes respalda este material. JavaScript es una marca registrada de Oracle Corporation en los Estados Unidos y otros países; se menciona solo para identificar el lenguaje.

El certificado de Testing-US acredita la aprobación de una evaluación de conocimientos. No es un título oficial ni habilita para el ejercicio profesional.