



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Sección 1: Variables y Tipos de Datos en JavaScript



### 1.1 Tipos de declaración de variables

- ♦ **var** (deprecado pero aún funcional)

javascript

```
var nombre = "Luis";  
console.log(nombre); // Luis
```

```
var nombre = "Pedro"; // Se puede redeclarar  
console.log(nombre); // Pedro
```

- ♦ **let** (uso recomendado)

javascript

```
let edad = 30;  
edad = 31; // Se puede reasignar  
console.log(edad); // 31
```

- ♦ **const** (constante, no puede reasignarse)

javascript

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

---



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## 1.2 Tipos de datos primitivos

### ♦ String

javascript

```
let mensaje = "Hola mundo";  
let saludo = 'Hola con comillas simples';  
let texto = `Hola con backticks`;
```

### ♦ Number

javascript

```
let entero = 10;  
let decimal = 3.14;  
let negativo = -50;
```

### ♦ Boolean

javascript

```
let esEstudiante = true;  
let estaActivo = false;
```

### ♦ Undefined

javascript

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



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ Null

javascript

```
let vacio = null;  
console.log(vacio); // null
```

## ♦ Symbol

javascript

```
let id = Symbol("id");  
let otroId = Symbol("id");  
console.log(id === otroId); // false
```

## ♦ BigInt

javascript

```
let numeroGrande = 123456789012345678901234567890n;  
console.log(typeof numeroGrande); // bigint
```

---



## 1.3 Tipos de datos estructurados

### ♦ Array

javascript

```
let numeros = [1, 2, 3];  
console.log(numeros[0]); // 1
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ Object

javascript

```
let persona = {  
  nombre: "Laura",  
  edad: 25  
};  
console.log(persona.nombre); // Laura
```

## ♦ Function

javascript

```
function saludar() {  
  return "Hola!";  
}  
console.log(typeof saludar); // function
```

---



## 1.4 Ejercicios variados



### Ejercicio 1: Declaración correcta con **let**, **var**, **const**

javascript

```
var ciudad = "Madrid";  
let pais = "España";  
const continente = "Europa";
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ✓ Ejercicio 2: Detectar tipos con **typeof**

javascript

```
console.log(typeof "Hola");      // string
console.log(typeof 42);          // number
console.log(typeof true);        // boolean
console.log(typeof undefined);   // undefined
console.log(typeof null);        // object (comportamiento
histórico)
console.log(typeof Symbol("id")); // symbol
```

## ✓ Ejercicio 3: Diferencia entre **undefined** y **null**

javascript

```
let a;
let b = null;
console.log(a); // undefined
console.log(b); // null
```

## ✓ Ejercicio 4: Variable no reassignable

javascript

```
const curso = "JavaScript";
// curso = "Python"; ❌ Error
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Sección 2: Operadores en JavaScript

---

### ♦ 2.1 Operadores Aritméticos

Operador	Significado	Ejemplo
+	Suma	5 + 3 // 8
-	Resta	5 - 3 // 2
*	Multiplicación	4 * 2 // 8
/	División	10 / 2 // 5
%	Módulo (resto)	10 % 3 // 1
**	Potencia	2 ** 3 // 8
++	Incremento	a++
--	Decremento	a--



#### Ejemplo:

javascript

```
let x = 5;
console.log(x + 2); // 7
console.log(x % 2); // 1
console.log(x ** 2); // 25
```



#### Ejercicio:

1. Declara dos números y muestra su suma, resta, división, multiplicación y el módulo.
2. Calcula el cuadrado de un número usando **\*\***.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## ♦ 2.2 Operadores de Asignación

Operador	Significado	Ejemplo
=	Asignación simple	<code>x = 10</code>
+=	Suma y asigna	<code>x += 5 // x = x+5</code>
-=	Resta y asigna	<code>x -= 3</code>
*=	Multiplica y asigna	<code>x *= 2</code>
/=	Divide y asigna	<code>x /= 4</code>
%=	Módulo y asigna	<code>x %= 2</code>



### Ejemplo:

javascript

```
let x = 10;  
x += 5; // 15  
x *= 2; // 30
```



### Ejercicio:

1. Usa una sola variable para realizar una operación acumulativa de 4 pasos: suma, resta, multiplicación y módulo.
  2. Escribe una función que reciba un número y le sume 10 usando `+=`.
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 2.3 Operadores de Comparación

Operador	Significado	Ejemplo
<code>==</code>	Igualdad (sin tipo)	<code>"5" == 5 // true</code>
<code>===</code>	Igualdad estricta (con tipo)	<code>"5" === 5 // false</code>
<code>!=</code>	Desigualdad (sin tipo)	<code>"5" != 5 // false</code>
<code>!==</code>	Desigualdad estricta	<code>"5" !== 5 // true</code>
<code>&gt;</code>	Mayor que	<code>7 &gt; 5 // true</code>
<code>&lt;</code>	Menor que	<code>3 &lt; 10 // true</code>
<code>&gt;=</code>	Mayor o igual	<code>5 &gt;= 5 // true</code>
<code>&lt;=</code>	Menor o igual	<code>4 &lt;= 3 // false</code>



### Ejemplo:

javascript

```
console.log(5 == "5"); // true
console.log(5 === "5"); // false
console.log(7 >= 7); // true
```



### Ejercicio:

1. Crea dos variables y compáralas con `==`, `===`, `!=`, y `!==`.
  2. Evalúa si un usuario tiene edad suficiente para entrar (edad mínima 18).
-





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 2.4 Operadores Lógicos

Operador	Significado	Ejemplo
<code>&amp;&amp;</code>	AND (y)	<code>true &amp;&amp; false // false</code>
,		
<code>!</code>	NOT (negación)	<code>!true // false</code>

### Ejemplo:

javascript

```
let tieneEdad = true;
let tieneDNI = false;

console.log(tieneEdad && tieneDNI); // false
console.log(tieneEdad || tieneDNI); // true
console.log(!tieneDNI);           // true
```

### Ejercicio:

1. Evalúa si un usuario puede acceder a una sala si tiene más de 18 años y tiene entrada.
  2. Usa `!` para invertir el valor de una variable booleana.
-

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 2.5 Operador Ternario

Estructura	Descripción
<code>condición ? expr1 : expr2</code>	Evalúa condición, retorna expr1 o expr2

### Ejemplo:

javascript

```
let edad = 20;  
let acceso = (edad >= 18) ? "Permitido" : "Denegado";  
console.log(acceso); // Permitido
```

### Ejercicio:

1. Escribe una función que devuelva "adulto" o "menor" según una edad.
2. Usa un ternario para determinar si un número es par o impar.

---

## ♦ 2.6 Operadores de Concatenación

### ♦ Con + (string + string o string + number)

javascript

```
let nombre = "Juan";  
let saludo = "Hola, " + nombre;  
console.log(saludo); // Hola, Juan
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ Template strings (backticks)

javascript

```
let producto = "camisa";  
let precio = 20;  
console.log(`El precio de la ${producto} es ${precio}`);
```



## Ejercicio:

1. Concatena nombre y apellido usando `+`.
2. Usa template strings para formar una frase con 3 variables.



## EJERCICIOS FINALES DE LA SECCIÓN



### Nivel Básico

1. Declara dos números. Muestra su suma, resta, producto, cociente y módulo.
2. Usa operadores de comparación para evaluar si un número es mayor que otro.



### Nivel Medio

3. Escribe una función que determine si un usuario puede acceder a un sitio según edad, suscripción activa y país.
4. Evalúa varias condiciones usando operadores lógicos y muestra mensajes diferentes.



### Nivel Avanzado

5. Crea una función que reciba tres parámetros: edad, saldo y rol, y devuelva:
  - "Acceso total" si es admin o saldo > 1000
  - "Acceso limitado" si edad > 18 y saldo > 100
  - "Acceso denegado" en cualquier otro caso (usa ternarios anidados)

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## Sección 3: Condicionales en JavaScript

---

### ♦ 3.1 **if, else if, else**

#### Estructura:

javascript

```
if (condición) {  
    // código si la condición es verdadera  
} else if (otraCondición) {  
    // código si la otra condición es verdadera  
} else {  
    // código si ninguna condición se cumple  
}
```

#### Ejemplo básico:

javascript

```
let edad = 17;  
  
if (edad >= 18) {  
    console.log("Mayor de edad");  
} else {  
    console.log("Menor de edad");  
}
```

#### Ejemplo con **else if**:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
let nota = 7;

if (nota >= 9) {
  console.log("Excelente");
} else if (nota >= 6) {
  console.log("Aprobado");
} else {
  console.log("Reprobado");
}
```



## Ejercicios:

1. Escribe un programa que determine si un número es **positivo**, **negativo** o **cero**.
  2. Pide una edad y evalúa si es:
    - Menor de edad
    - Mayor de edad
    - Mayor de 65 (tercera edad)
- 

## ♦ 3.2 Operador ternario (? :)



### Estructura:

javascript

```
condición ? expresiónSiVerdadero : expresiónSiFalso;
```



### Ejemplo:

javascript

```
let usuario = "admin";
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
let mensaje = (usuario === "admin") ? "Bienvenido administrador" :  
"Acceso restringido";  
console.log(mensaje);
```



## Ejercicios:

1. Crea un programa que indique si puedes conducir según tu edad con un ternario.
  2. Usa ternario para mostrar si un número es **par** o **impar**.
- 

## ♦ 3.3 Condicionales anidados



### Estructura:

javascript

```
if (condición1) {  
  if (condición2) {  
    // código si ambas se cumplen  
  }  
}
```



### Ejemplo:

javascript

```
let usuario = "admin";  
let activo = true;  
  
if (usuario === "admin") {  
  if (activo) {  
    console.log("Admin activo");  
  } else {  
    console.log("Admin inactivo");  
  }  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicio:

1. Verifica si un estudiante tiene:
    - Edad mayor a 18
    - Matrícula pagadaSi ambas se cumplen, muestra "Acceso al campus".
- 

## ◆ 3.4 Uso con operadores lógicos (&&, ||, !)

### Ejemplo:

javascript

```
let edad = 20;
let suscrito = true;

if (edad >= 18 && suscrito) {
  console.log("Acceso completo");
}
```

## Ejercicio:

1. Evalúa si un usuario puede acceder al sistema si:
    - Edad mayor de 18
    - Tiene usuario y clave
    - O es "superadmin" (aunque no cumpla lo anterior)
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 3.5 switch



### Estructura:

javascript

```
switch (valor) {  
  case opción1:  
    // código  
    break;  
  case opción2:  
    // código  
    break;  
  default:  
    // código por defecto  
}
```



### Ejemplo:

javascript

```
let dia = "lunes";  
  
switch (dia) {  
  case "lunes":  
    console.log("Inicio de semana");  
    break;  
  case "viernes":  
    console.log("Fin de semana cerca");  
    break;  
  default:  
    console.log("Día común");  
}
```



### Ejercicio:

1. Crea un `switch` que detecte el mes y muestre si es:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Invierno (dic-ene-feb)
  - Primavera (mar-abr-may)
  - Verano (jun-jul-ago)
  - Otoño (sep-oct-nov)
- 

## EJERCICIOS FINALES DE LA SECCIÓN

### Nivel básico

1. Escribe un programa que determine si puedes votar con edad.
2. Evalúa si un número ingresado es positivo, negativo o cero.

### Nivel medio

3. Crea un sistema de calificaciones:
  - 0 a 4: “Insuficiente”
  - 5 a 6: “Suficiente”
  - 7 a 8: “Notable”
  - 9 a 10: “Excelente”

### Nivel avanzado

4. Crea un sistema de acceso con:
  - Usuario
  - Contraseña

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Rol (“admin”, “editor”, “usuario”)
5. **Si es admin y activo**, mostrar “Acceso total”.  
**Si es editor**, “Acceso parcial”.  
**Si es usuario**, “Acceso limitado”.  
**Cualquier otro caso**, “Acceso denegado”.

## Sección 4: Bucles en JavaScript

---

### ◆ 4.1 for

#### Estructura:

javascript

```
for (inicialización; condición; incremento) {  
    // código que se repite  
}
```

#### Ejemplo básico:

javascript

```
for (let i = 1; i <= 5; i++) {  
    console.log("Número:", i);  
}
```

#### Ejercicios:

1. Imprime los números del 1 al 100.
2. Imprime los números pares del 2 al 20.
3. Imprime la tabla de multiplicar del 5 (del 1 al 10).

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## ♦ 4.2 while

### Estructura:

javascript

```
while (condición) {  
    // código que se ejecuta mientras la condición sea true  
}
```

### Ejemplo:

javascript

```
let i = 1;  
while (i <= 5) {  
    console.log("Iteración:", i);  
    i++;  
}
```

### Ejercicios:

1. Cuenta desde 10 hasta 1.
2. Pide al usuario números hasta que introduzca un 0.
3. Suma los números ingresados hasta que se escriba un número negativo.

---

## ♦ 4.3 do...while

### Estructura:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
do {  
    // se ejecuta al menos una vez  
} while (condición);
```



## Ejemplo:

javascript

```
let i = 1;  
do {  
    console.log("Valor:", i);  
    i++;  
} while (i <= 3);
```



## Ejercicios:

1. Solicita contraseñas hasta que escriba la correcta.
  2. Pide al usuario ingresar edades y termina cuando una edad es mayor a 100.
- 

## ♦ 4.4 **for...of** (para arrays y objetos iterables)



## Estructura:

javascript

```
for (let item of array) {  
    // accede al valor directamente  
}
```



## Ejemplo:

javascript

```
const frutas = ["manzana", "banana", "uva"];  
for (let fruta of frutas) {
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
console.log(fruta);  
}
```

## Ejercicios:

1. Recorre un array de nombres e imprime cada uno.
  2. Recorre un array de números y calcula el total.
- 

## ◆ 4.5 **for...in** (para objetos)

### Estructura:

javascript

```
for (let clave in objeto) {  
    // accede a claves  
}
```

### Ejemplo:

javascript

```
const persona = {  
    nombre: "Ana",  
    edad: 28,  
    ciudad: "Madrid"  
};  
  
for (let clave in persona) {  
    console.log(clave + ":", persona[clave]);  
}
```

## Ejercicios:

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

1. Recorre un objeto `alumno` con nombre, edad y curso, y muestra cada clave/valor.
  2. Recorre un objeto `producto` y muestra un string como "El precio es X", etc.
- 

## ♦ 4.6 Control de bucles: `break` y `continue`

### Ejemplo con `break`:

javascript

```
for (let i = 1; i <= 10; i++) {  
  if (i === 5) break;  
  console.log(i); // 1 a 4  
}
```

### Ejemplo con `continue`:

javascript

```
for (let i = 1; i <= 5; i++) {  
  if (i === 3) continue;  
  console.log(i); // 1, 2, 4, 5  
}
```

### Ejercicios:

1. Imprime del 1 al 10, pero salta el número 7 usando `continue`.
  2. Imprime del 1 al 10 y detente cuando encuentres el primer número par mayor que 5 usando `break`.
- 

## Ejercicios Finales de Bucles

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Nivel básico:

1. Imprime todos los múltiplos de 3 del 1 al 30.
2. Calcula el factorial de un número ( $5! = 5 \times 4 \times 3 \times 2 \times 1$ ).

## Nivel intermedio:

3. Dado un array de notas, calcula la media usando un bucle.
4. Cuenta cuántos elementos mayores a 50 hay en un array.

## Nivel avanzado:

5. Recorre un array de objetos con nombre y edad, y muestra solo los mayores de edad.
6. Crea un generador de lista `<ul>` en el DOM a partir de un array con `for...of`.

## Sección 5: Funciones en JavaScript

---

### ◆ 5.1 Función Declarada (Function Declaration)

#### Estructura:

javascript

```
function nombreFuncion(parámetros) {  
  // código  
  return valor;  
}
```

#### Ejemplo:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
function saludar(nombre) {  
  return `Hola, ${nombre}`;  
}  
  
console.log(saludar("Lucía")); // Hola, Lucía
```



## Ejercicios:

1. Crea una función que reciba dos números y retorne su suma.
2. Crea una función que calcule el área de un triángulo.

---

## ♦ 5.2 Función Anónima (Function Expression)



### Estructura:

javascript

```
const miFuncion = function(parámetros) {  
  // código  
};
```



### Ejemplo:

javascript

```
const multiplicar = function(a, b) {  
  return a * b;  
};  
  
console.log(multiplicar(4, 5)); // 20
```



## Ejercicios:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

1. Declara una función anónima que reste dos números.
  2. Asigna una función a una variable y úsala para determinar si un número es par.
- 

## ◆ 5.3 Función Flecha (Arrow Function)

### Estructura:

javascript

```
const suma = (a, b) => a + b;
```

### Ejemplos:

javascript

```
const saludar = nombre => `Hola, ${nombre}`;  
const cuadrado = n => n * n;  
const mostrarMensaje = () => console.log("Hola mundo");
```

### Ejercicios:

1. Crea una función flecha que devuelva el cubo de un número.
  2. Crea una función flecha que determine si un texto tiene más de 10 caracteres.
- 

## ◆ 5.4 Parámetros y Retorno

### Ejemplo con múltiples parámetros:

javascript

```
function calcularPromedio(n1, n2, n3) {  
    return (n1 + n2 + n3) / 3;  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}
```

## Retorno temprano:

javascript

```
function evaluar(nota) {  
  if (nota >= 5) return "Aprobado";  
  return "Reprobado";  
}
```

## Ejercicios:

1. Escribe una función que reciba una edad y retorne si puede votar.
2. Crea una función que reciba un texto y lo devuelva en mayúsculas.

---

## ◆ 5.5 Funciones como argumentos (Callbacks)

### Ejemplo básico:

javascript

```
function ejecutar(fn) {  
  fn();  
}
```

```
ejecutar(() => console.log("¡Callback ejecutado!"));
```

### Ejemplo con parámetros:

javascript

```
function operar(a, b, callback) {  
  return callback(a, b);  
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
console.log(operar(4, 2, (x, y) => x + y)); // 6
```



## Ejercicios:

1. Escribe una función `procesar` que reciba un array y una función, y aplique esa función a cada elemento.
2. Usa `operar()` con una función que reste, multiplique y divida.

---

## ♦ 5.6 Parámetros por defecto



### Ejemplo:

javascript

```
function saludar(nombre = "invitado") {  
  return `Hola, ${nombre}`;  
}
```

```
console.log(saludar()); // Hola, invitado
```



### Ejercicio:

1. Crea una función que reciba un mensaje y un destinatario. Si no se proporciona destinatario, usar "Usuario".

---

## ♦ 5.7 Parámetros Rest (...args) y Spread (...)



### Ejemplo - Parámetros variables:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
function sumar(...numeros) {  
  return numeros.reduce((acc, val) => acc + val, 0);  
}
```

```
console.log(sumar(1, 2, 3, 4)); // 10
```



## Ejercicio:

1. Crea una función que reciba cualquier cantidad de strings y devuelva un solo string concatenado.
- 

## ◆ 5.8 Función dentro de función (Closures básicos)



## Ejemplo:

javascript

```
function crearSaludo(nombre) {  
  return function() {  
    console.log(`Hola, ${nombre}`);  
  };  
}
```

```
const saludoAna = crearSaludo("Ana");  
saludoAna(); // Hola, Ana
```



## Ejercicio:

1. Crea una función `crearContador` que devuelva una función que sume de 1 en 1.
- 



## Ejercicios Finales de Funciones

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Nivel básico:

1. Función que reciba un número y retorne si es múltiplo de 3.
2. Función que reciba una cadena y devuelva la misma cadena invertida.

## Nivel intermedio:

3. Función que reciba un array de edades y devuelva el promedio.
4. Función que reciba un número y devuelva un array con sus divisores.

## Nivel avanzado:

5. Crea una función `crearContadorDesde(x)` que genere una función que al llamarla sume +1 desde `x`.
6. Crea una función `filtrar` que reciba un array y un callback, y devuelva un nuevo array con los elementos que cumplan esa condición.

---

## Sección 6: Arrays en JavaScript

---

### ◆ 6.1 Crear y acceder a arrays

#### Declaración:

javascript

```
const numeros = [1, 2, 3];  
const frutas = new Array("manzana", "banana", "uva");
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Acceso por índice:

javascript

```
console.log(frutas[0]); // manzana
console.log(numeros.length); // 3
```



## Ejercicios:

1. Crea un array con tus 5 comidas favoritas.
2. Muestra la primera y última usando índices.

---

## ♦ 6.2 Métodos mutables (**push**, **pop**, **shift**, **unshift**, **splice**, **sort**, **reverse**)

Método	Descripción
<code>push()</code>	Añade al final
<code>pop()</code>	Elimina el último
<code>shift()</code>	Elimina el primero
<code>unshift()</code>	Añade al principio
<code>splice()</code>	Añade/quita en posición específica
<code>sort()</code>	Ordena alfabéticamente o numéricamente
<code>reverse()</code>	Invierte el orden



## Ejemplo:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
let frutas = ["manzana", "pera"];  
frutas.push("naranja");           // ["manzana", "pera", "naranja"]  
frutas.shift();                   // ["pera", "naranja"]  
frutas.splice(1, 0, "uva");       // ["pera", "uva", "naranja"]
```



## Ejercicios:

1. Crea una lista de tareas y agrega una tarea al final con `push`.
  2. Elimina el primer y último elemento con `shift` y `pop`.
  3. Inserta un nuevo elemento en la segunda posición con `splice`.
- 

## ◆ 6.3 Recorrer arrays

### ◆ `for`

javascript

```
for (let i = 0; i < frutas.length; i++) {  
  console.log(frutas[i]);  
}
```

### ◆ `for...of`

javascript

```
for (let fruta of frutas) {  
  console.log(fruta);  
}
```

### ◆ `forEach()`

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
frutas.forEach((item, i) => {  
  console.log(`${i}: ${item}`);  
});
```



## Ejercicios:

1. Recorre un array de nombres y salúdalos uno por uno.
2. Crea un array de números e imprímelos multiplicados por 2.

---

## ♦ 6.4 Métodos funcionales (**map**, **filter**, **reduce**, **find**, **some**, **every**, **includes**)

---

### ♦ **map()** – transforma elementos

javascript

```
const numeros = [1, 2, 3];  
const dobles = numeros.map(n => n * 2); // [2, 4, 6]
```

### ♦ **filter()** – filtra elementos

javascript

```
const mayores = numeros.filter(n => n > 1); // [2, 3]
```

### ♦ **reduce()** – reduce a un solo valor

javascript

```
const suma = numeros.reduce((acc, n) => acc + n, 0); // 6
```

### ♦ **find()** – encuentra el primero que cumple





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

javascript

```
const encontrado = numeros.find(n => n > 1); // 2
```

## ♦ **some()** – ¿alguno cumple?

javascript

```
const hayMayor = numeros.some(n => n > 2); // true
```

## ♦ **every()** – ¿todos cumplen?

javascript

```
const todosMayores = numeros.every(n => n > 0); // true
```

## ♦ **includes()** – ¿contiene un valor?

javascript

```
console.log(numeros.includes(2)); // true
```



## Ejercicios prácticos:

### Nivel básico

1. Usa **map** para convertir todos los nombres en mayúsculas.
2. Usa **filter** para obtener los mayores de 18 de un array de edades.

### Nivel intermedio

3. Usa **reduce** para sumar todos los números de un array.
4. Usa **find** para encontrar el primer número divisible por 3.

### Nivel avanzado

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

5. Crea una función que reciba un array de objetos con nombre y edad, y devuelva los nombres de los mayores de edad.
  6. Usa `every` para verificar si todos los elementos de un array son pares.
- 

## ◆ 6.5 Combinar, copiar y clonar arrays

### Spread Operator (`...`)

javascript

```
const a = [1, 2];
const b = [3, 4];
const combinado = [...a, ...b]; // [1, 2, 3, 4]
```

### `concat()`

javascript

```
const combinado2 = a.concat(b);
```

### Copiar por valor

javascript

```
const copia = [...a];
```

### Ejercicios:

1. Une dos arrays usando `spread` y `concat`.
  2. Crea una copia de un array original y modifica el nuevo sin afectar el primero.
-

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 6.6 Otras utilidades

### **join()** – convierte array en string

javascript

```
let palabras = ["Hola", "mundo"];  
console.log(palabras.join(" ")); // "Hola mundo"
```

### **split()** – convierte string en array

javascript

```
let texto = "uno,dos,tres";  
let array = texto.split(","); // ["uno", "dos", "tres"]
```

### **Array.isArray()**

javascript

```
console.log(Array.isArray([1, 2])); // true
```

### **Ejercicios:**

1. Convierte un array de palabras en una frase con `join`.
2. Toma una cadena de texto y conviértela en array con `split()`.

---

## Ejercicios Finales de Arrays

### **Nivel básico:**

- Crea un array de números y muestra sus cuadrados con `map()`.
- Filtra los nombres que empiezan con "A".

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Nivel intermedio:

- Usa `reduce()` para calcular el promedio de un array de notas.
- Filtra un array de objetos `{nombre, edad}` por mayores de edad.

## Nivel avanzado:

- Recibe un array de usuarios `{nombre, activo}` y devuelve solo los activos.
- Dado un array de nombres, devuélvelo ordenado alfabéticamente y en mayúsculas.

---

## Sección 7: Objetos en JavaScript

---

### ◆ 7.1 Crear objetos

#### Sintaxis literal:

javascript

```
const persona = {  
  nombre: "Carlos",  
  edad: 30,  
  activo: true  
};
```

#### Constructor **Object**:

javascript

```
const coche = new Object();  
coche.marca = "Toyota";  
coche.modelo = "Corolla";
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Ejercicios:

1. Crea un objeto `alumno` con propiedades: nombre, curso, nota.
  2. Crea un objeto `libro` con propiedades: título, autor, año.
- 

## ♦ 7.2 Acceder y modificar propiedades



### Punto y corchetes:

javascript

```
console.log(persona.nombre);      // Carlos
console.log(persona["edad"]);     // 30
persona.edad = 31;                // modificar
persona.email = "carlos@mail.com" // añadir
```



## Ejercicios:

1. Modifica el nombre de un objeto `alumno`.
  2. Agrega una propiedad `aprobado` al objeto.
- 

## ♦ 7.3 Métodos en objetos (`this`)



### Añadir función como método:

javascript

```
const persona = {
  nombre: "Ana",
  saludar: function () {
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    return `Hola, soy ${this.nombre}`;  
  }  
};
```



## Ejercicios:

1. Crea un método `presentarse()` que devuelva un mensaje con nombre y edad.
2. Crea un objeto `animal` con método `sonido()` que imprima "El gato hace miau".

---

## ♦ 7.4 Recorrer objetos (`for...in`)

javascript

```
const producto = {  
  nombre: "Teclado",  
  precio: 50,  
  stock: true  
};  
  
for (let clave in producto) {  
  console.log(clave + ":", producto[clave]);  
}
```



## Ejercicios:

1. Crea un objeto `usuario` y recórrelo para mostrar clave y valor.
2. Muestra sólo las propiedades cuyo valor sea de tipo string.

---

## ♦ 7.5 Objetos anidados



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

javascript

```
const persona = {  
  nombre: "Luis",  
  direccion: {  
    ciudad: "Madrid",  
    calle: "Gran Vía"  
  }  
};  
  
console.log(persona.direccion.ciudad); // Madrid
```



## Ejercicios:

1. Crea un objeto **empresa** con propiedades anidadas: nombre, dirección, empleados.
  2. Accede a una propiedad anidada y modifícala.
- 

## ♦ 7.6 Arrays de objetos

javascript

```
const productos = [  
  { nombre: "Libro", precio: 10 },  
  { nombre: "Lápiz", precio: 1 }  
];  
  
for (let prod of productos) {  
  console.log(`${prod.nombre}: ${prod.precio}`);  
}
```



## Ejercicios:

1. Crea un array de objetos **tareas** con propiedades: **nombre**, **completado**.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Filtra las tareas completadas con `filter`.
- 

## ♦ 7.7 Desestructuración de objetos

javascript

```
const persona = {  
  nombre: "Eva",  
  edad: 40  
};  
  
const { nombre, edad } = persona;  
console.log(nombre, edad); // Eva 40
```



### Ejercicios:

1. Desestructura un objeto `alumno` para obtener su nombre y nota.
  2. Usa desestructuración para pasar propiedades como parámetros de una función.
- 

## ♦ 7.8 Clonar y combinar objetos



### Spread operator:

javascript

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



### Object.assign():

javascript

```
const copia2 = Object.assign({}, persona);
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicios:

1. Clona un objeto `libro` y agrega una propiedad `disponible`.
  2. Combina dos objetos: `datosPersonales` y `datosContacto`.
- 

## ♦ 7.9 Convertir entre objetos y JSON

### **JSON.stringify()** → objeto a texto

javascript

```
const persona = { nombre: "Laura" };  
const json = JSON.stringify(persona); // '{"nombre":"Laura"}'
```

### **JSON.parse()** → texto a objeto

javascript

```
const obj = JSON.parse('{"nombre":"Laura"}');
```

## Ejercicios:

1. Convierte un objeto en cadena JSON y muéstralo en consola.
  2. Recibe una cadena JSON y accede a una propiedad tras parsearla.
- 

## ♦ 7.10 Introducción a clases (siguiente nivel de objetos)

javascript

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
    return `Hola, me llamo ${this.nombre}`;
  }
}

const p1 = new Persona("Sofía", 22);
console.log(p1.saludar());
```

## Ejercicios:

1. Crea una clase **Coche** con propiedades **marca**, **modelo** y un método **arrancar()**.
2. Crea una clase **Libro** con método **resumen()** que devuelva título y autor.

---

## Ejercicios Finales de Objetos

### Nivel básico:

- Crea un objeto **videojuego** con 3 propiedades y una función **presentarJuego()**.
- Modifica una propiedad y elimina otra con **delete**.

### Nivel intermedio:

- Crea un array de objetos **usuarios** y muestra solo los activos.
- Usa un **for...in** para recorrer un objeto anidado y mostrar sus propiedades.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Nivel avanzado:

- Escribe una función que reciba un objeto y devuelva un array con solo las claves cuyos valores son números.
- Usa clases para modelar una tienda con productos que puedan tener descuento aplicado.

---

## Sección 8: Manipulación del DOM en JavaScript

---

### ♦ 8.1 ¿Qué es el DOM?

El **DOM** es una representación estructurada en forma de árbol del documento HTML. JavaScript permite **acceder y modificar** cualquier parte del DOM en tiempo real.

---

### ♦ 8.2 Seleccionar elementos del DOM

#### Métodos:

Método	Descripción
<code>getElementById()</code>	Por ID
<code>getElementsByClassName()</code>	Por clase (HTMLCollection)
<code>getElementsByTagName()</code>	Por etiqueta
<code>querySelector()</code>	Primer elemento que cumpla el selector

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

`querySelectorAll()` Todos los elementos que cumplan

## Ejemplos:

javascript

```
const titulo = document.getElementById("miTitulo");
const parrafos = document.getElementsByTagName("p");
const items = document.querySelectorAll(".item");
```

## Ejercicios:

1. Selecciona un `h1` por ID y cambia su texto.
2. Selecciona todos los `li` y cambia su color a azul.

---

## ♦ 8.3 Cambiar contenido

Método	Descripción
<code>innerHTML</code>	Cambia el contenido HTML
<code>textContent</code>	Cambia solo texto plano

## Ejemplo:

javascript

```
titulo.innerHTML = "<span>Nuevo Título</span>";
titulo.textContent = "Título sin HTML";
```

## Ejercicios:

1. Cambia el contenido de un `div` por texto nuevo.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Usa `innerHTML` para agregar un enlace dentro de un párrafo.

---

## ♦ 8.4 Cambiar estilos desde JS

### Acceso directo:

javascript

```
element.style.color = "red";  
element.style.fontSize = "20px";
```

### Agregar y quitar clases:

javascript

```
element.classList.add("activo");  
element.classList.remove("oculto");  
element.classList.toggle("resaltado");
```

### Ejercicios:

1. Al hacer clic en un botón, cambia el fondo de un `div`.
2. Alterna entre modo claro y oscuro con una clase CSS.

---

## ♦ 8.5 Crear y eliminar elementos

### Crear:

javascript

```
const nuevoParrafo = document.createElement("p");  
nuevoParrafo.textContent = "Soy nuevo aquí";  
document.body.appendChild(nuevoParrafo);
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Eliminar:

javascript

```
const viejo = document.getElementById("viejo");  
viejo.remove();
```



## Ejercicios:

1. Crea dinámicamente un botón y agrégalo al **body**.
2. Elimina un párrafo al hacer clic en otro botón.

---

## ♦ 8.6 Modificar atributos



## Métodos:

javascript

```
element.setAttribute("src", "imagen.png");  
element.getAttribute("href");  
element.removeAttribute("class");
```



## Ejercicios:

1. Cambia el atributo **src** de una imagen al hacer clic.
2. Modifica el **placeholder** de un input desde un botón.

---

## ♦ 8.7 Manejo de eventos (**addEventListener**)



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Sintaxis:

javascript

```
element.addEventListener("click", function() {  
  console.log("Haz hecho clic");  
});
```

Evento	Descripción
<code>click</code>	Al hacer clic
<code>input</code>	Al escribir en un campo
<code>mouseover</code>	Al pasar el mouse encima
<code>submit</code>	Al enviar un formulario
<code>change</code>	Al cambiar un valor ( <code>input</code> )



## Ejercicios:

1. Muestra un mensaje al hacer clic en un botón.
2. Muestra en pantalla lo que el usuario escribe en un input.

---

## ♦ 8.8 Formularios y eventos



## Capturar datos:

javascript

```
form.addEventListener("submit", function(e) {  
  e.preventDefault(); // evita recarga  
  const valor = document.getElementById("nombre").value;  
  console.log(valor);  
});
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicios:

1. Captura nombre y email de un formulario y muéstralos debajo.
  2. Valida que los campos no estén vacíos antes de enviar.
- 

## ♦ 8.9 Generar HTML dinámico

### Plantilla de tarjetas:

javascript

```
const productos = ["Camisa", "Pantalón"];
productos.forEach(producto => {
  const item = document.createElement("li");
  item.textContent = producto;
  document.getElementById("lista").appendChild(item);
});
```

## Ejercicios:

1. Dado un array de tareas, crea una lista HTML `<ul>` con cada tarea como `<li>`.
  2. Al presionar un botón, agrega un elemento nuevo con datos ingresados.
- 

## Ejercicios Finales de DOM

### Nivel básico:

- Crea un `div` con texto y cambia su color al hacer clic.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Crea un botón que, al hacer clic, oculte un párrafo.

## Nivel intermedio:

- Crea una lista que permita añadir nuevos elementos con un input + botón.
- Al enviar un formulario, muestra los datos capturados en una tarjeta HTML.

## Nivel avanzado:

- Crea una galería que muestre miniaturas. Al hacer clic, muestra una imagen grande.
- Crea un sistema de “modo oscuro” que se aplique a todo el sitio dinámicamente.

## Sección 9: Eventos Avanzados en JavaScript

---

### ♦ 9.1 Propagación de eventos

#### Fases del evento:

- Captura (capture phase): de `document` hacia el elemento.
- Burbuja (bubble phase): del elemento hacia `document`.

#### Sintaxis:

javascript

```
element.addEventListener("click", callback, useCapture);
```

- `useCapture = false` (por defecto): burbuja

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- `true`: fase de captura

## Ejemplo:

html

```
<div id="padre">
  <button id="hijo">Haz clic</button>
</div>

<script>
  document.getElementById("padre").addEventListener("click", () => {
    console.log("Padre clic");
  });

  document.getElementById("hijo").addEventListener("click", (e) => {
    console.log("Hijo clic");
    // e.stopPropagation(); // descomenta para evitar burbuja
  });
</script>
```

## Ejercicio:

1. Crea tres elementos anidados (`div1`, `div2`, `div3`) y muestra qué orden siguen los eventos al hacer clic.

---

## ♦ 9.2 Delegación de eventos

### ¿Qué es?

Asigna un evento a un **contenedor** en lugar de a cada elemento individual. Muy útil para listas dinámicas.

## Ejemplo:

html



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
<ul id="lista">
  <li>Item 1</li>
  <li>Item 2</li>
</ul>

<script>
  document.getElementById("lista").addEventListener("click", (e) =>
  {
    if (e.target.tagName === "LI") {
      console.log("Haz clic en:", e.target.textContent);
    }
  });
</script>
```



## Ejercicio:

1. Crea una lista dinámica de tareas. Al hacer clic en una tarea, márcala como completada usando delegación.

---

## ♦ 9.3 Eventos de teclado

Evento	Descripción
<code>keydown</code>	Al presionar una tecla
<code>n</code>	
<code>keyup</code>	Al soltar una tecla
<code>keypress</code>	(obsoleto) Se usaba para texto



## Ejemplo:

javascript

```
document.addEventListener("keydown", (e) => {
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
console.log("Tecla presionada:", e.key);
});
```

## Ejercicios:

1. Muestra en pantalla qué tecla presionó el usuario.
  2. Si pulsa la tecla "Escape", cierra un modal o cambia de color el fondo.
- 

## ♦ 9.4 Temporizadores (**setTimeout**, **setInterval**)

 **setTimeout**: ejecuta una vez después del tiempo indicado

javascript

```
setTimeout(() => {
  console.log("Pasaron 3 segundos");
}, 3000);
```

 **setInterval**: ejecuta repetidamente

javascript

```
let contador = 0;
const id = setInterval(() => {
  contador++;
  console.log("Contador:", contador);
  if (contador === 5) clearInterval(id);
}, 1000);
```

## Ejercicios:

1. Crea un mensaje que aparezca 2 segundos después de cargar la página.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Crea un reloj que aumente un número en pantalla cada segundo hasta llegar a 10.
- 

## ♦ 9.5 Eventos del navegador

Evento	Descripción
<code>DOMContentLoaded</code>	Cuando el DOM está listo
<code>load</code>	Cuando todo (incluidas imágenes) cargó
<code>beforeunload</code>	Al intentar salir de la página

### Ejemplo:

javascript

```
window.addEventListener("beforeunload", (e) => {
  e.preventDefault();
  e.returnValue = ""; // navegador mostrará aviso
});
```

### Ejercicio:

1. Muestra un aviso si el usuario intenta cerrar la página.
  2. Usa `DOMContentLoaded` para iniciar una función solo cuando el HTML esté cargado.
- 

## ♦ 9.6 Eventos de scroll y resize

### Scroll:

javascript

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
window.addEventListener("scroll", () => {  
  console.log(window.scrollY);  
});
```

## **Resize:**

javascript

```
window.addEventListener("resize", () => {  
  console.log("Ancho:", window.innerWidth);  
});
```

## **Ejercicios:**

1. Muestra un mensaje cuando el usuario llegue al final de la página.
2. Cambia el tamaño de un elemento al redimensionar la ventana.

---

## **Ejercicios Finales de Eventos Avanzados**

### **Nivel básico:**

- Muestra una alerta 3 segundos después de entrar a la página.
- Muestra una cuenta regresiva de 10 a 0 en pantalla.

### **Nivel intermedio:**

- Al presionar **Enter** en un campo, valida el contenido.
- Crea un botón que se desactiva tras hacer clic y se reactiva luego de 5 segundos.

### **Nivel avanzado:**

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Usa delegación para una lista de comentarios que se agregan dinámicamente.
- Crea un sistema de scroll infinito que cargue más elementos simulados.

## Sección 10: Formularios y Validaciones en JavaScript

---

### ◆ 10.1 Capturar envío de un formulario

#### Sintaxis básica:

javascript

```
const form = document.getElementById("miFormulario");

form.addEventListener("submit", function(e) {
  e.preventDefault(); // Evita que se recargue la página
  const nombre = document.getElementById("nombre").value;
  console.log("Nombre enviado:", nombre);
});
```

#### Ejercicios:

1. Crea un formulario con nombre y edad. Captura ambos valores al enviar.
  2. Muestra los datos ingresados debajo del formulario sin recargar la página.
- 

### ◆ 10.2 Acceder a los campos

javascript

```
const nombre = formulario.nombre.value;
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
const email = formulario.elements["email"].value;
```

O directamente por `getElementById()`:

javascript

```
document.getElementById("email").value;
```



## Ejercicios:

1. Accede a varios campos: nombre, email, mensaje.
2. Usa `console.log()` para verificar qué datos fueron ingresados.

---

## ♦ 10.3 Validación básica con JavaScript



### Ejemplo:

javascript

```
if (nombre.trim() === "") {  
    alert("El nombre es obligatorio");  
    return;  
}
```

```
if (!email.includes("@")) {  
    alert("Email inválido");  
    return;  
}
```



## Ejercicios:

1. Valida que el campo nombre no esté vacío.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Valida que la edad sea un número mayor de 0.
3. Valida que la contraseña tenga al menos 6 caracteres.

---

## ◆ 10.4 Validación con expresiones regulares

### Ejemplo (email válido):

javascript

```
const regexEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!regexEmail.test(email)) {
  alert("Email no válido");
}
```

### Ejercicios:

1. Usa una expresión regular para validar un número de teléfono español.
2. Valida un campo contraseña que:
  - Tenga una mayúscula
  - Tenga un número
  - Tenga al menos 8 caracteres

---

## ◆ 10.5 Validación en tiempo real (**input**, **change**)

javascript

```
document.getElementById("nombre").addEventListener("input", (e) => {
  if (e.target.value.length < 3) {
    e.target.style.borderColor = "red";
  }
});
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    } else {  
        e.target.style.borderColor = "green";  
    }  
});
```



## Ejercicios:

1. Cambia el color del borde de un input según si es válido o no.
  2. Muestra un mensaje debajo del campo si está mal completado.
- 

## ♦ 10.6 Mostrar mensajes personalizados

javascript

```
const mensajeError = document.getElementById("errorNombre");  
  
if (nombre === "") {  
    mensajeError.textContent = "El nombre es obligatorio";  
} else {  
    mensajeError.textContent = "";  
}
```



## Ejercicios:

1. Muestra errores debajo de cada campo si es incorrecto.
  2. Al corregir un campo, borra el mensaje de error automáticamente.
- 

## ♦ 10.7 Confirmaciones y éxito

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
form.addEventListener("submit", (e) => {
  e.preventDefault();
  if (todoCorrecto) {
    alert("Formulario enviado correctamente");
    form.reset(); // Limpia todos los campos
  }
});
```



## Ejercicios:

1. Muestra un modal o mensaje con “Enviado con éxito” al validar todo.
2. Usa `form.reset()` solo si los campos son válidos.

---

## ♦ 10.8 Formularios con múltiples inputs dinámicos

javascript

```
// Suponiendo campos clonables de forma dinámica
const inputs = document.querySelectorAll("input[type='text']");

inputs.forEach(input => {
  input.addEventListener("change", () => {
    console.log("Nuevo valor:", input.value);
  });
});
```



## Ejercicio:

1. Crea un formulario con lista de productos agregables dinámicamente.
  2. Valida que todos los campos agregados estén llenos antes de enviar.
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 10.9 Validaciones HTML5 + JS

html

```
<input type="email" required minlength="5">
```

Puedes usar HTML5 para validaciones básicas y JavaScript para validaciones avanzadas.

javascript

```
if (!form.checkValidity()) {  
    alert("Formulario inválido");  
    return;  
}
```



## Ejercicios Finales de Formularios y Validación



### Nivel básico:

- Crea un formulario con nombre, edad y email. Valida que ningún campo esté vacío.
- Muestra los valores debajo del formulario al enviarlo (sin recargar).



### Nivel intermedio:

- Agrega validaciones:
  - Nombre > 2 letras
  - Edad numérica > 17
  - Email válido con regex
- Muestra mensajes personalizados en cada campo si hay error.



### Nivel avanzado:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Crea una lista dinámica de productos con nombre y cantidad.
- Al hacer submit, muestra los datos como tabla.
- Si falta un dato, marca el campo y muestra error específico.
- Muestra “Formulario válido y enviado” si todo está correcto.



## Sección 11: Almacenamiento en el navegador con JavaScript

---

### ◆ 11.1 ¿Qué es LocalStorage?

- Almacena **pares clave–valor** en el navegador de forma **persistente** (incluso tras cerrar el navegador).
- Disponible mediante `localStorage`.



#### Sintaxis:

javascript

```
// Guardar
localStorage.setItem("nombre", "Carlos");

// Leer
let nombre = localStorage.getItem("nombre");

// Eliminar uno
localStorage.removeItem("nombre");

// Eliminar todo
localStorage.clear();
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicios:

1. Guarda el nombre del usuario y recupéralo al recargar la página.
  2. Crea un botón para eliminar un dato específico del localStorage.
- 

## ◆ 11.2 ¿Qué es SessionStorage?

- Similar a `localStorage`, pero los datos se eliminan al cerrar la pestaña o navegador.
- Disponible mediante `sessionStorage`.

## Sintaxis:

javascript

```
sessionStorage.setItem("modo", "oscuro");  
let modo = sessionStorage.getItem("modo");
```

## Ejercicios:

1. Guarda el modo oscuro/claro usando `sessionStorage`.
  2. Al recargar la pestaña, aplica el último modo seleccionado.
- 

## ◆ 11.3 Guardar objetos (con JSON)

Tanto `localStorage` como `sessionStorage` solo almacenan **strings**, por lo que debes convertir objetos con `JSON.stringify()` y `JSON.parse()`.

## Ejemplo:

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
const usuario = { nombre: "Ana", edad: 25 };
localStorage.setItem("usuario", JSON.stringify(usuario));

const datos = JSON.parse(localStorage.getItem("usuario"));
console.log(datos.nombre); // Ana
```



## Ejercicios:

1. Guarda un objeto con nombre, email y edad.
  2. Al cargar la página, muestra los datos en un `div`.
- 

## ♦ 11.4 Guardar listas/arrays



### Ejemplo:

javascript

```
const tareas = ["Leer", "Estudiar", "Practicar"];
localStorage.setItem("tareas", JSON.stringify(tareas));

const tareasGuardadas = JSON.parse(localStorage.getItem("tareas"));
tareasGuardadas.forEach(t => console.log(t));
```



## Ejercicios:

1. Crea una lista de tareas (array) y guárdala.
  2. Al cargar la página, muestra las tareas guardadas en pantalla.
- 

## ♦ 11.5 ¿Qué son las Cookies?



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Las **cookies** también almacenan información en el navegador, pero tienen:
  - Tamaño limitado (~4KB)
  - Fecha de expiración
  - Se envían al servidor en cada petición



## Crear cookie:

javascript

```
document.cookie = "usuario=Carlos; expires=Tue, 01 Jan 2026 00:00:00 UTC; path="/;
```



## Leer cookies:

javascript

```
console.log(document.cookie);
```



## Ejercicios:

1. Crea una cookie llamada **idioma** con valor **"español"**.
2. Al cargar la página, muestra un mensaje personalizado si la cookie existe.




---

## ♦ 11.6 Comparación general

Característica	LocalStorage	SessionStorage	Cookies
Duración	Permanente	Solo sesión	Personalizable
Capacidad aprox.	5–10 MB	5 MB	4 KB



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

Envía al servidor	 No	 No	 Sí
Formato	Clave/valor (string)	Igual	Texto plano

---

## Ejercicios Finales de Almacenamiento

### Nivel básico:

- Guarda un valor simple como “nombre de usuario” en localStorage.
- Muestra el nombre al recargar la página.

### Nivel intermedio:

- Guarda un array de productos con **nombre** y **precio**, y recupéralo al iniciar la app.
- Usa un botón para vaciar el localStorage completamente.

### Nivel avanzado:

- Crea un formulario donde el usuario ingrese sus datos personales (nombre, edad, email).
  - Al hacer submit, guarda esos datos en localStorage.
  - Al volver a la página, si hay datos, rellena automáticamente el formulario.
  - Crea una opción para “Olvidar mis datos” que los borre del localStorage.
- 

## Sección 12: Proyecto práctico — To-Do List (Lista de tareas)



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---



## Paso 1: Estructura HTML mínima

html

```
<h1>Mi Lista de Tareas</h1>

<form id="form-tarea">
  <input type="text" id="input-tarea" placeholder="Escribe una
tarea..." required />
  <button type="submit">Agregar</button>
</form>

<ul id="lista-tareas"></ul>
```

---



## Paso 2: Lógica JavaScript básica

javascript

```
const formulario = document.getElementById("form-tarea");
const input = document.getElementById("input-tarea");
const lista = document.getElementById("lista-tareas");
let tareas = [];

// Cargar tareas del localStorage al iniciar
document.addEventListener("DOMContentLoaded", () => {
  const guardadas = localStorage.getItem("tareas");
  if (guardadas) {
    tareas = JSON.parse(guardadas);
    tareas.forEach(crearTareaEnDOM);
  }
});
```

---

## + Paso 3: Agregar tarea



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

javascript

```
formulario.addEventListener("submit", (e) => {
  e.preventDefault();
  const texto = input.value.trim();
  if (texto === "") return;

  const nuevaTarea = {
    id: Date.now(),
    texto,
    completada: false
  };

  tareas.push(nuevaTarea);
  guardarTareas();
  crearTareaEnDOM(nuevaTarea);
  input.value = "";
});
```

---



## Paso 4: Crear tarea en el DOM

javascript

```
function crearTareaEnDOM(tarea) {
  const li = document.createElement("li");
  li.textContent = tarea.texto;
  li.dataset.id = tarea.id;
  if (tarea.completada) li.classList.add("completada");

  // Botón eliminar
  const btnEliminar = document.createElement("button");
  btnEliminar.textContent = "✖";
  btnEliminar.className = "borrar";
  li.appendChild(btnEliminar);

  lista.appendChild(li);
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}
```

---



## Paso 5: Eliminar tarea (delegación de eventos)

javascript

```
lista.addEventListener("click", (e) => {  
  const li = e.target.closest("li");  
  const id = li.dataset.id;  
  
  if (e.target.matches("button.borrar")) {  
    tareas = tareas.filter(t => t.id !== Number(id));  
    guardarTareas();  
    li.remove();  
  }  
});
```

---



## Paso 6: Marcar como completada

javascript

```
lista.addEventListener("click", (e) => {  
  const li = e.target.closest("li");  
  const id = Number(li.dataset.id);  
  
  if (!e.target.matches("button.borrar")) {  
    tareas = tareas.map(t => {  
      if (t.id === id) t.completada = !t.completada;  
      return t;  
    });  
    guardarTareas();  
    li.classList.toggle("completada");  
  }  
});
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## Paso 7: Guardar y recuperar desde **localStorage**

javascript

```
function guardarTareas() {  
  localStorage.setItem("tareas", JSON.stringify(tareas));  
}
```

---





## Paso 8: Estilos CSS básicos (opcional)

CSS

```
.completada {  
  text-decoration: line-through;  
  color: gray;  
}  
  
button.borrar {  
  margin-left: 10px;  
  color: red;  
  background: none;  
  border: none;  
  cursor: pointer;  
}
```

---

## Retos Extra (para práctica avanzada)

1.  Añadir botón “Eliminar todas las tareas”.
2.  Añadir campo de búsqueda para filtrar tareas por texto.
3.  Mostrar total de tareas, completadas y pendientes.
4.  Añadir animaciones (usando `classList` y CSS).

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

5. ☒ Exportar lista como JSON descargable.
  6. ☒ Cambiar el tema (modo oscuro) y guardar en `localStorage`.
- 

## ☒ Resultado esperado

Una app funcional que:

- Permite agregar tareas
- Permite marcarlas como completadas
- Permite eliminarlas
- Guarda las tareas en el navegador (persistencia)
- Funciona aunque se recargue la página

## Sección 13: Proyecto práctico — Lista de usuarios desde una API

---

### Objetivo del proyecto

- ☒ Conectarse a una API pública (<https://jsonplaceholder.typicode.com/users>)
  - ☒ Obtener y mostrar datos de usuarios (nombre, email, ciudad...)
  - ☒ Mostrar un loading mientras carga
  - ☒ Añadir botón para recargar
  - ☒ Opcional: ver detalles del usuario al hacer clic
- 

### Paso 1: HTML base

html

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- `<h1>Lista de Usuarios</h1>`
  - `<button id="btn-recargar">Recargar usuarios</button>`
  - `<p id="estado">Cargando...</p>`
  - `<ul id="lista-usuarios"></ul>`
- 

## Paso 2: Código JavaScript base

javascript

- `const lista = document.getElementById("lista-usuarios");`
- `const estado = document.getElementById("estado");`
- `const boton = document.getElementById("btn-recargar");`
- 
- `// Función para obtener usuarios`
- `async function obtenerUsuarios() {`
- `estado.textContent = "Cargando...";`
- `lista.innerHTML = "";`
- 
- `try {`
- `const res = await`
- `fetch("https://jsonplaceholder.typicode.com/users");`
- `const datos = await res.json();`
- `estado.textContent = "Usuarios cargados";`
- `datos.forEach(mostrarUsuario);`
- `} catch (error) {`
- `estado.textContent = "Error al cargar usuarios";`
- `}`
- `}`
- 
- `// Mostrar usuario en la lista`
- `function mostrarUsuario(usuario) {`



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- `const li = document.createElement("li");`
- `li.innerHTML = ``
- `<strong>${usuario.name}</strong><br>`
- `Email: ${usuario.email}<br>`
- `Ciudad: ${usuario.address.city}`
- ``;`
- `lista.appendChild(li);`
- `}`
- 
- `// Al cargar la página`
- `document.addEventListener("DOMContentLoaded",`  
 `obtenerUsuarios);`
- 
- `// Botón de recarga`
- `boton.addEventListener("click", obtenerUsuarios);`



## Resultado hasta aquí

- Se muestra una lista de usuarios con `name`, `email` y `city`.
- Hay feedback visual con `"Cargando..."` y `"Usuarios cargados"`.
- Se puede recargar la lista con un botón.



## Paso 3 (Opcional): Mostrar detalles del usuario al hacer clic

javascript

- `function mostrarUsuario(usuario) {`





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- `const li = document.createElement("li");`
- `li.innerHTML = `<strong>${usuario.name}</strong>`;`
- `li.style.cursor = "pointer";`
- 
- `li.addEventListener("click", () => {`
- `alert(`Nombre: ${usuario.name}`
- `Email: ${usuario.email}`
- `Teléfono: ${usuario.phone}`
- `Sitio: ${usuario.website}`);`
- `});`
- 
- `lista.appendChild(li);`
- `}`



## Retos extra



### Nivel básico:

1. Añade una animación de “cargando” antes de que lleguen los datos.
2. Muestra solo los usuarios cuyo nombre empiece con “C”.



### Nivel intermedio:

3. Crea un campo de búsqueda en vivo (filtra por nombre mientras escribes).
4. Muestra una tarjeta más detallada al hacer clic en un usuario.



### Nivel avanzado:

5. Usa otra API (<https://jsonplaceholder.typicode.com/posts>) y muestra los **posts** del usuario clicado.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

6. Implementa paginación (ejemplo: mostrar 3 usuarios por página y avanzar con botones).

## Sección 14: Proyecto práctico — Galería de imágenes con búsqueda

---

### ¿Qué necesitas?

- Una cuenta gratuita en <https://unsplash.com/developers>
- Crear una app y obtener una **API Key**
- La URL base de búsqueda:

```
https://api.unsplash.com/search/photos?query=palabra&client_id=TU_API_KEY
```

---

### Paso 1: Estructura HTML base

html

```
<h1>Galería de Imágenes</h1>

<form id="form-busqueda">

  <input type="text" id="input-busqueda" placeholder="Buscar imágenes..." required />

  <button type="submit">Buscar</button>

</form>
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
<p id="estado">Escribe una palabra y pulsa buscar</p>
```

```
<div id="galeria" style="display: grid; gap: 10px;  
grid-template-columns: repeat(auto-fill, minmax(200px,  
1fr));"></div>
```



## Paso 2: Código JavaScript base

javascript

```
const form = document.getElementById("form-busqueda");  
const input = document.getElementById("input-busqueda");  
const galeria = document.getElementById("galeria");  
const estado = document.getElementById("estado");  
  
const API_KEY = "TU_API_KEY"; // reemplázala por tu clave real  
  
form.addEventListener("submit", async (e) => {  
  e.preventDefault();  
  const consulta = input.value.trim();  
  
  if (consulta === "") return;
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
galeria.innerHTML = "";

estado.textContent = "Buscando imágenes...";


try {

    const res = await
fetch(`https://api.unsplash.com/search/photos?query=${consulta}&client_id=${API_KEY}`);

    const data = await res.json();

    if (data.results.length === 0) {

        estado.textContent = "No se encontraron resultados.";

        return;

    }

    estado.textContent = `Mostrando resultados para: "${consulta}"`;

    data.results.forEach(imagen => {

        const img = document.createElement("img");

        img.src = imagen.urls.small;

        img.alt = imagen.alt_description;

        img.title = imagen.description || imagen.alt_description ||
"Imagen";

        img.style.width = "100%";

        galeria.appendChild(img);
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    });  
  
    } catch (error) {  
        estado.textContent = "Error al buscar imágenes";  
    }  
});
```

---

## Retos adicionales

### Nivel básico:

1. Mostrar solo las 9 primeras imágenes (`data.results.slice(0, 9)`).
2. Si no se encuentra nada, mostrar mensaje “No hay imágenes”.

### Nivel intermedio:

3. Añade un botón "Ver más" para cargar más imágenes (paginación).
4. Muestra el autor y el número de likes debajo de cada imagen.

### Nivel avanzado:

5. Muestra cada imagen en un modal grande al hacer clic.
  6. Permite guardar búsquedas recientes en `localStorage`.
- 

## Estilos recomendados (CSS)



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

CSS

```
img {  
  
  border-radius: 10px;  
  
  transition: transform 0.2s ease;  
  
}  
  
img:hover {  
  
  transform: scale(1.05);  
  
  cursor: pointer;  
  
}
```

---

## Resultado esperado

- Una galería dinámica que permite buscar imágenes.
- Carga imágenes desde la API de Unsplash y las muestra en una cuadrícula.
- Muestra mensajes si hay errores, no hay resultados o está cargando.



## Sección 15: Proyecto Completo — Portal Personal Interactivo

---



### Objetivo del proyecto

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

Construir un **sitio web interactivo tipo "dashboard personal"**, que incluya:

1. ☒ Nombre del usuario guardado en `localStorage`
2. ☒ Un mensaje de bienvenida personalizado
3. ☒ Un buscador de imágenes (API de Unsplash)
4. ☒ Una lista de tareas (To-Do List)
5. ☒ Modo oscuro activable
6. ☒ Animaciones suaves
7. ☒ Todo **sin recargar la página**

---

## Estructura general del HTML

html

```
<body>
```

```
  <header>
```

```
    <h1 id="saludo">¡Bienvenido!</h1>
```

```
    <button id="modo">🌙 Modo Oscuro</button>
```

```
  </header>
```

```
  <section id="usuario">
```

```
    <input type="text" id="nombre" placeholder="Escribe tu nombre" />
```

```
    <button id="guardarNombre">Guardar nombre</button>
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
</section>
```

```
<section id="tareas">
```

```
<h2> Mis Tareas</h2>
```

```
<form id="form-tarea">
```

```
<input type="text" id="input-tarea" placeholder="Agregar  
tarea" required />
```

```
<button>Agregar</button>
```

```
</form>
```

```
<ul id="lista-tareas"></ul>
```

```
</section>
```

```
<section id="galeria">
```

```
<h2> Buscar Imágenes</h2>
```

```
<form id="form-busqueda">
```

```
<input type="text" id="input-busqueda" placeholder="Buscar  
imágenes..." required />
```

```
<button>Buscar</button>
```

```
</form>
```

```
<div id="imagenes" class="grid"></div>
```

```
</section>
```

```
</body>
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## JavaScript modular (resumen por función)

### 1. Nombre de usuario personalizado

javascript

```
const saludo = document.getElementById("saludo");

const nombreInput = document.getElementById("nombre");

const btnGuardarNombre = document.getElementById("guardarNombre");

function cargarNombre() {

    const nombre = localStorage.getItem("usuario");

    if (nombre) saludo.textContent = `¡Bienvenido, ${nombre}!`;

}

btnGuardarNombre.addEventListener("click", () => {

    const nombre = nombreInput.value.trim();

    if (nombre) {

        localStorage.setItem("usuario", nombre);

        cargarNombre();

        nombreInput.value = "";

    }

});
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
document.addEventListener("DOMContentLoaded", cargarNombre);
```

---

## 2. Lista de tareas (To-Do List)

javascript

```
const formTarea = document.getElementById("form-tarea");
const inputTarea = document.getElementById("input-tarea");
const listaTareas = document.getElementById("lista-tareas");
let tareas = JSON.parse(localStorage.getItem("tareas")) || [];

function renderTareas() {
  listaTareas.innerHTML = "";
  tareas.forEach(t => {
    const li = document.createElement("li");
    li.textContent = t.texto;
    li.className = t.completada ? "hecha" : "";
    li.addEventListener("click", () => {
      t.completada = !t.completada;
      guardarTareas();
      renderTareas();
    });
  });
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    });  
  
    listaTareas.appendChild(li);  
  
  });  
}  
  
function guardarTareas() {  
  localStorage.setItem("tareas", JSON.stringify(tareas));  
}  
  
formTarea.addEventListener("submit", e => {  
  e.preventDefault();  
  
  const texto = inputTarea.value.trim();  
  
  if (texto) {  
    tareas.push({ texto, completada: false });  
  
    guardarTareas();  
  
    renderTareas();  
  
    inputTarea.value = "";  
  }  
});  
  
renderTareas();
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## 3. Buscar imágenes (API Unsplash)

javascript

```
const formBusqueda = document.getElementById("form-busqueda");
const inputBusqueda = document.getElementById("input-busqueda");
const contenedorImagenes = document.getElementById("imagenes");
const API_KEY = "TU_API_KEY";

formBusqueda.addEventListener("submit", async (e) => {
  e.preventDefault();

  const query = inputBusqueda.value.trim();

  if (!query) return;

  contenedorImagenes.innerHTML = "Cargando...";

  const res = await
  fetch(`https://api.unsplash.com/search/photos?query=${query}&client_
  id=${API_KEY}`);

  const data = await res.json();

  contenedorImagenes.innerHTML = "";

  data.results.forEach(foto => {

    const img = document.createElement("img");
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
img.src = foto.urls.small;

img.alt = foto.alt_description;

img.title = foto.description || foto.alt_description ||
"Imagen";

contenedorImagenes.appendChild(img);

});

});
```

---

## 4. Activar modo oscuro

javascript

```
const btnModo = document.getElementById("modo");

btnModo.addEventListener("click", () => {

    document.body.classList.toggle("oscuro");

    localStorage.setItem("modo",
document.body.classList.contains("oscuro") ? "oscuro" : "claro");

});

document.addEventListener("DOMContentLoaded", () => {

    if (localStorage.getItem("modo") === "oscuro") {

        document.body.classList.add("oscuro");
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}  
));
```

---

## CSS básico sugerido

CSS

```
body {  
  font-family: sans-serif;  
  padding: 20px;  
  transition: background-color 0.3s ease;  
}  
.oscuro {  
  background-color: #121212;  
  color: white;  
}  
.grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));  
  gap: 10px;  
}  
img {
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
width: 100%;

border-radius: 8px;

transition: transform 0.2s;
}

img:hover {

  transform: scale(1.05);
}

.hecha {

  text-decoration: line-through;

  color: gray;
}
```

---


## Resultado esperado

- Sitio interactivo donde el usuario:
    - Guarda su nombre
    - Añade tareas marcables
    - Busca imágenes
    - Cambia a modo oscuro
    - Y todo se guarda sin perderse al cerrar la página
-

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Retos adicionales

### Nivel intermedio:

- Agregar contador de tareas pendientes.
- Permitir eliminar tareas con botón .
- Mostrar fecha y hora actual en pantalla.

### Nivel avanzado:

- Mostrar historial de búsquedas de imágenes.
- Crear una galería favorita usando `localStorage`.
- Implementar un botón “Resetear todo” (borra todos los datos guardados).

## Sección 16: Programación Orientada a Objetos en JavaScript (POO + ES6)

---

### ◆ 16.1 ¿Qué es la POO?

La **Programación Orientada a Objetos (POO)** es un paradigma basado en objetos que tienen:

- **Propiedades** (atributos o estado)
- **Métodos** (funciones que actúan sobre el objeto)
- **Encapsulamiento**: esconder los detalles internos
- **Herencia**: compartir propiedades entre clases





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- **Polimorfismo:** redefinir comportamientos
- 

## ♦ 16.2 Crear una clase con **class**

javascript

```
class Persona {  
  
  constructor(nombre, edad) {  
  
    this.nombre = nombre;  
  
    this.edad = edad;  
  
  }  
  
  saludar() {  
  
    return `Hola, soy ${this.nombre}`;  
  
  }  
}  
  
const p1 = new Persona("Ana", 28);  
  
console.log(p1.saludar()); // Hola, soy Ana
```



### Ejercicio:

1. Crea una clase **Animal** con propiedades **especie** y **sonido**.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Añade un método `emitirSonido()` que devuelva el sonido del animal.

---

## ♦ 16.3 Métodos y propiedades

javascript

```
class Producto {  
  
  constructor(nombre, precio) {  
  
    this.nombre = nombre;  
  
    this.precio = precio;  
  
  }  
  
  
  mostrarPrecio() {  
  
    return `${this.nombre} cuesta ${this.precio}`;  
  
  }  
  
}
```



### Ejercicio:

1. Crea un objeto `Producto` y usa su método para mostrar el precio.
  2. Añade un método `conIVA()` que devuelva el precio con 21% adicional.
-

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 16.4 Encapsulamiento con # (privado)

javascript

```
class Cuenta {  
  
    #saldo = 0;  
  
    constructor(titular) {  
        this.titular = titular;  
    }  
  
    ingresar(cantidad) {  
        if (cantidad > 0) this.#saldo += cantidad;  
    }  
  
    mostrarSaldo() {  
        return `Saldo de ${this.titular}: $$${this.#saldo}`;  
    }  
}
```

### Ejercicio:

1. Crea una clase **Caja** con propiedad privada **#contenido**.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Añade métodos `abrir()` y `guardar(cosa)` que interactúen con el contenido.
- 

## ♦ 16.5 Herencia (**extends**, **super**)

javascript

```
class Usuario {  
  
  constructor(nombre) {  
  
    this.nombre = nombre;  
  
  }  
  
  saludar() {  
  
    return `Hola, soy ${this.nombre}`;  
  
  }  
}
```

```
class Admin extends Usuario {  
  
  constructor(nombre, rol) {  
  
    super(nombre);  
  
    this.rol = rol;  
  
  }  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
saludar() {  
    return `Administrador ${this.nombre} (${this.rol})`;  
}  
}
```

## Ejercicio:

1. Crea una clase `Empleado` que herede de `Persona`.
2. Añade propiedades extra como `puesto` y `salario`, y sobrescribe `saludar()`.

---

## ◆ 16.6 Getters y Setters

javascript

```
class Usuario {  
    constructor(nombre) {  
        this._nombre = nombre;  
    }  
  
    get nombre() {  
        return this._nombre.toUpperCase();  
    }  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
set nombre(valor) {  
    if (valor.length >= 3) this._nombre = valor;  
}  
}
```

## Ejercicio:

1. Usa `get` y `set` para validar el formato de un email.
2. Crea un `getter` que devuelva el nombre formateado como "Sr. Nombre".

---

## ◆ 16.7 Métodos estáticos (`static`)

javascript

```
class Calculadora {  
    static sumar(a, b) {  
        return a + b;  
    }  
}
```

```
console.log(Calculadora.sumar(3, 4)); // 7
```

## Ejercicio:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

1. Crea una clase `Convertor` con métodos estáticos `kmAMillas`, `kgALibras`.
- 

## ♦ 16.8 Composición (objetos dentro de objetos)

javascript

```
class Motor {  
    constructor(cilindrada) {  
        this.cilindrada = cilindrada;  
    }  
}  
  
class Coche {  
    constructor(marca, motor) {  
        this.marca = marca;  
        this.motor = motor;  
    }  
  
    describir() {  
        return `${this.marca} con motor de ${this.motor.cilindrada} cc`;  
    }  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
const motor = new Motor(1600);  
  
const coche = new Coche("Toyota", motor);
```

## Ejercicio:

1. Crea un objeto `Usuario` con un objeto `Direccion` anidado.

---

## ♦ 16.9 Clases como modelo de datos (CRUD simple)

javascript

```
class ListaUsuarios {  
  
  constructor() {  
  
    this.usuarios = [];  
  
  }  
  
  agregar(usuario) {  
  
    this.usuarios.push(usuario);  
  
  }  
  
  eliminar(nombre) {  
  
    this.usuarios = this.usuarios.filter(u => u.nombre !== nombre);  
  
  }  
  
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}

mostrar() {
    return this.usuarios;
}
}
```



## Ejercicio:

1. Implementa una clase `ListaTareas` con métodos `agregar()`, `eliminar()`, `listar()` y `buscar()`.



## Proyecto final (Mini CRM)

**Clase:** Cliente (nombre, email, historial de compras)

**Clase hija:** ClientePremium (añade descuento)

- Métodos: `saludar()`, `agregarCompra()`, `calcularTotal()`
- Encapsula el total gastado
- Usa `get` para mostrar resumen personalizado

# Sección 17: JavaScript Avanzado — Funcional y Patrones de Diseño



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ◆ 17.1 ¿Qué es la programación funcional?

Es un paradigma donde las **funciones son tratadas como valores**, evitando mutaciones y efectos secundarios. Se basa en:

- ✓ Funciones puras
  - ✓ Inmutabilidad
  - ✓ Funciones de orden superior
  - ✓ Transparencia referencial
- 

## ◆ 17.2 Funciones puras

**Definición:** siempre devuelven el mismo resultado con los mismos argumentos, y **no modifican el exterior**.

javascript

```
// Pura

const suma = (a, b) => a + b;


// Impura (usa variable externa)

let total = 0;

function agregar(n) {

    total += n; // modifica algo externo

}
```



### Ejercicio:

1. Escribe una función pura que duplique un array sin modificar el original.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Escribe una versión **impura** y luego **pura** de una función que capitalice nombres.

---

## ♦ 17.3 Inmutabilidad

Evita **modificar directamente** objetos, arrays o valores.

javascript

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

```
// ❌ mutación
```

```
persona.edad = 31;
```

```
// ✅ inmutable
```

```
const personaActualizada = { ...persona, edad: 31 };
```



### Ejercicio:

1. A partir de un objeto usuario, crea una copia modificando solo el email.
2. A partir de un array, añade un nuevo elemento sin usar **push**.

---

## ♦ 17.4 Funciones de orden superior

Son funciones que **reciben otras funciones como argumentos** o **devuelven funciones**.

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
function operar(fn, a, b) {  
  return fn(a, b);  
}
```

```
const suma = (x, y) => x + y;  
console.log(operar(suma, 4, 5)); // 9
```



## Ejercicio:

1. Crea una función `filtrarNumeros(array, fn)` que use un callback como filtro.
2. Crea una función que devuelva otra función multiplicadora:  
`crearMultiplicador(2)(5) → 10`

---

## ♦ 17.5 Composición de funciones

Combina funciones pequeñas para formar lógicas más complejas.

javascript

```
const mayusculas = str => str.toUpperCase();  
const agregarSigno = str => str + "!";  
const exclamacion = str => agregarSigno(mayusculas(str));  
  
console.log(exclamacion("hola")); // HOLA!
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicio:

1. Compón funciones para formatear un nombre como: "luis" → "LUIS."
  2. Escribe una función `pipe(f1, f2, f3)` que las combine en orden.
- 

## ◆ 17.6 Currying

Transforma una función que recibe varios argumentos en una cadena de funciones de un solo argumento.

javascript

```
const sumar = a => b => c => a + b + c;
```

```
console.log(sumar(1)(2)(3)); // 6
```

## Ejercicio:

1. Crea una función currificada para crear saludos personalizados:  
`saludar("Hola")("Carlos")`
- 

## ◆ 17.7 Patrones de diseño (intro)

Patrones comunes que solucionan problemas de estructura en JavaScript.

---



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ✓ Module Pattern

Encapsula variables en una función autoejecutable.

javascript

```
const contador = (function () {  
  let valor = 0;  
  return {  
    incrementar: () => ++valor,  
    obtener: () => valor  
  };  
})();
```



## Ejercicio:

1. Crea un módulo `calculadora` con funciones `sumar`, `restar`, y una propiedad interna `memoria`.

---

## ✓ Factory Pattern

Crea objetos sin usar clases.

javascript

```
function crearUsuario(nombre) {  
  return {
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    nombre,  
    saludar: () => `Hola, soy ${nombre}`  
  };  
}
```

## Ejercicio:

1. Usa una factory para crear objetos `producto(nombre, precio)` con método `descuento()`.

---

## Observer Pattern (básico)

Permite que varios objetos “escuchen” cambios en otro.

javascript

```
class Emisor {  
  constructor() {  
    this.subs = [];  
  }  
  
  suscribir(fn) {  
    this.subs.push(fn);  
  }  
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
emitir(mensaje) {  
  this.subs.forEach(fn => fn(mensaje));  
}  
}
```

## Ejercicio:

1. Crea un sistema de notificaciones donde múltiples funciones reaccionan a una alerta emitida.
- 

## Proyecto Integrador Avanzado

Mini app de carrito funcional (sin clases):

- Productos como objetos puros
- `addToCart()` como función pura
- `cartReducer()` estilo Redux (estado puro)
- LocalStorage funcional
- `pipe()` para aplicar transformaciones a cada producto (descuento + formato)

## Sección 18: JavaScript Asíncrono — Callbacks, Promesas y Async/Await

---

### ◆ 18.1 ¿Qué es asincronía?





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

En JavaScript, **el código se ejecuta línea a línea** (de forma síncrona), pero algunas tareas **esperan o tardan** (como peticiones HTTP, timers, lectura de archivos...).

En lugar de **bloquear el flujo**, JavaScript usa **eventos y promesas** para seguir trabajando mientras espera.

---

## ♦ 18.2 Callbacks

Un **callback** es una función que se pasa como argumento para que se ejecute **después** de algo.

javascript

```
function esperar(mensaje, callback) {  
    setTimeout(() => {  
        console.log(mensaje);  
        callback();  
    }, 2000);  
}
```

```
esperar("Cargando...", () => console.log("Terminado"));
```



### Ejercicio:

1. Simula un login con `setTimeout` y luego una función `mostrarBienvenida(nombre)`.
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ◆ 18.3 Problema del Callback Hell

Encadenar muchos callbacks anidados puede generar código difícil de leer:

javascript

```
login(usuario, () => {  
  getDatos(usuario, () => {  
    mostrarInfo(() => {  
      actualizarVista();  
    });  
  });  
});
```

➡ Solución: Promesas

---

## ◆ 18.4 Promesas

Una **promesa** representa el resultado eventual de una operación:

javascript

```
const promesa = new Promise((resolve, reject) => {  
  const exito = true;  
  setTimeout(() => {  
    exito ? resolve("Éxito") : reject("Error");  
  });  
});
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    }, 1000);  
  });
```

promesa

```
.then(res => console.log("Respuesta:", res))  
.catch(err => console.error("Error:", err))  
.finally(() => console.log("Fin"));
```

## Ejercicio:

1. Crea una función `simularPetición(url)` que resuelva o rechace aleatoriamente.
2. Muestra el estado y resultado con `then` y `catch`.

---

## ◆ 18.5 Encadenamiento `.then()`

javascript

```
getUsuario()  
  
  .then(usuario => getPedidos(usuario.id))  
  
  .then(pedidos => mostrar(pedidos))  
  
  .catch(err => console.error("Algo falló:", err));
```

---



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 18.6 `async` y `await`

Una forma **más limpia** de escribir código asíncrono moderno:

javascript

```
async function cargarDatos() {  
  try {  
    const res = await  
fetch("https://jsonplaceholder.typicode.com/users");  
    const data = await res.json();  
    console.log("Usuarios:", data);  
  } catch (err) {  
    console.error("Error al cargar:", err);  
  }  
}
```



### Ejercicio:

1. Crea una función `buscarPosts()` que use `await` y muestre los títulos.
2. Captura errores de red con `try/catch`.

---

## ♦ 18.7 `Promise.all` y `Promise.race`

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
// Espera a que TODAS se completen

Promise.all([

  fetch("/api/productos"),

  fetch("/api/clientes")

])

.then(respuestas => console.log("Todo listo"))

.catch(err => console.error("Error en alguna"));


// Solo espera la primera que termine

Promise.race([

  fetch("/api/a"),

  fetch("/api/b")

]).then(r => console.log("Primera resuelta"));
```



## Ejercicio:

1. Simula 3 tareas con `setTimeout` y úsalas con `Promise.all` y `race`.

---

## ◆ 18.8 AbortController (cancelar peticiones)

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
const controller = new AbortController();

fetch("https://jsonplaceholder.typicode.com/users", {
  signal: controller.signal
})

  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error("Cancelado:", err.name));

// Cancela en 1 segundo
setTimeout(() => controller.abort(), 1000);
```

---



## Proyecto final: Buscador de usuarios con loading y errores

1. Input para buscar usuario por ID (1-10)
  2. Muestra info desde <https://jsonplaceholder.typicode.com/users/ID>
  3. Usa `fetch`, `await`, `try/catch`
  4. Muestra mensaje de error si el ID no existe
  5. Muestra “Cargando...” mientras espera la respuesta
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Retos adicionales

### Básico:

- Simula una función `esperar(ms)` que devuelva una promesa que resuelve tras X milisegundos.
- Encadena 3 mensajes con `setTimeout` usando `async/await`.

### Intermedio:

- Crea una `caja de carga` con mensaje mientras se esperan datos de 2 endpoints con `Promise.all`.

### Avanzado:

- Implementa reintento automático si `fetch` falla (hasta 3 veces).
- Implementa timeout con `Promise.race([petición, temporizador])`.



## Sección 19: Web APIs Avanzadas en JavaScript

Estas APIs son nativas del navegador y te permiten hacer cosas como:

- ✓ Acceder a ubicación
- ✓ Dibujar con Canvas
- ✓ Arrastrar elementos
- ✓ Leer portapapeles
- ✓ Notificar al usuario
- ✓ Entrar en pantalla completa



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- ✓ Vibrar (en móviles)
  - ✓ Acceder a audio/video
- 

## ♦ 19.1 Geolocalización (Geolocation API)

javascript

```
navigator.geolocation.getCurrentPosition(
  (pos) => {
    const { latitude, longitude } = pos.coords;
    console.log(`Ubicación: ${latitude}, ${longitude}`);
  },
  (err) => {
    console.error("Error de ubicación:", err.message);
  }
);
```



### Ejercicio:

1. Muestra tu ubicación en un mapa de Google Maps con un `iframe`.
  2. Muestra la ubicación solo si el usuario lo permite.
- 

## ♦ 19.2 Canvas (gráficos dinámicos)

html

```
<canvas id="miCanvas" width="300" height="150"></canvas>
```

javascript

```
const canvas = document.getElementById("miCanvas");
const ctx = canvas.getContext("2d");
```

```
// Dibujar un rectángulo rojo
```





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
ctx.fillStyle = "red";
ctx.fillRect(10, 10, 100, 50);

// Círculo azul
ctx.beginPath();
ctx.arc(150, 75, 30, 0, Math.PI * 2);
ctx.fillStyle = "blue";
ctx.fill();
```



## Ejercicio:

1. Dibuja tu nombre con `ctx.fillText()`.
2. Crea un pequeño juego de dibujo con mouse (tipo Paint).

---

## ♦ 19.3 Drag & Drop API

html

```

<div id="soltar" style="width:200px; height:200px; border:1px dashed
#000;"></div>
```

javascript

```
const drag = document.getElementById("arrastrar");
const drop = document.getElementById("soltar");

drag.addEventListener("dragstart", e =>
e.dataTransfer.setData("text", drag.id));

drop.addEventListener("dragover", e => e.preventDefault());
drop.addEventListener("drop", e => {
  e.preventDefault();
  const data = e.dataTransfer.getData("text");
  drop.appendChild(document.getElementById(data));
});
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
});
```



## Ejercicio:

1. Arrastra imágenes a una galería.
2. Arrastra tareas para reordenar una lista.

---

## ♦ 19.4 Clipboard API (copiar y pegar)

javascript

```
navigator.clipboard.writeText("¡Copiado al portapapeles!")
  .then(() => alert("Texto copiado"))
  .catch(err => console.error("Error:", err));
```

javascript

```
navigator.clipboard.readText()
  .then(text => console.log("Pegado:", text));
```



## Ejercicio:

1. Añade botón “Copiar link”.
2. Crea un campo para pegar lo que hay en el portapapeles.

---

## ♦ 19.5 Fullscreen API

javascript

```
document.documentElement.requestFullscreen(); // activar
```

```
document.exitFullscreen(); // salir
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Ejercicio:

1. Añade un botón "Pantalla completa".
  2. Detecta si ya estás en modo fullscreen.
- 

## ◆ 19.6 Notification API

javascript

```
Notification.requestPermission().then(permission => {  
  if (permission === "granted") {  
    new Notification("¡Hola!", {  
      body: "Esta es una notificación",  
      icon: "icon.png"  
    });  
  }  
});
```

## Ejercicio:

1. Envía una notificación al completar una tarea o al iniciar sesión.
- 

## ◆ 19.7 Vibration API (solo móviles)

javascript

```
navigator.vibrate([200, 100, 200]); // vibra -> pausa -> vibra
```

## Ejercicio:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

1. Vibra el teléfono al hacer clic en un botón "alarma".
- 

## ♦ 19.8 MediaDevices API (audio y cámara)

javascript

```
navigator.mediaDevices.getUserMedia({ video: true, audio: false })
  .then(stream => {
    document.querySelector("video").srcObject = stream;
  })
  .catch(error => console.error("No se pudo acceder a la cámara:",
error));
```



### Ejercicio:

1. Muestra tu cámara en pantalla.
  2. Haz una app tipo “espejo selfie” con `video` y `canvas`.
- 



## Proyecto Integrador: Dashboard con Web APIs

1. Muestra tu ubicación en mapa.
2. Botón de "Copiar link de ubicación".
3. Arrastra tareas a distintas columnas (tipo Trello).
4. Modo fullscreen para enfoque.
5. Usa notificaciones para avisos (“Tarea completada”).
6. Usa vibración y sonido si estás en móvil.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Sección 20: Testeo en JavaScript — Unit tests, Jest, depuración y mocks

---

### ¿Por qué hacer tests?

Los tests te permiten:

- ✓ Validar que tu código hace lo que debe
  - ✓ Detectar errores antes de que lleguen al usuario
  - ✓ Refactorizar sin miedo
  - ✓ Automatizar la verificación de funcionalidades
  - ✓ Documentar comportamientos esperados
- 

### ◆ 20.1 Tipos de tests

- **Unitarios:** prueban funciones pequeñas (unidad mínima)
- **Integración:** prueban cómo interactúan varias funciones/módulos
- **End-to-End (E2E):** prueban el comportamiento real (como un usuario)

En esta sección nos enfocamos en los **tests unitarios** con **Jest**.

---

### ◆ 20.2 ¿Qué es Jest?

Jest es un framework de testing para JavaScript (desarrollado por Meta/Facebook), ideal para probar funciones, objetos, clases y módulos.

bash

```
npm install --save-dev jest
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

Agrega en tu `package.json`:

json

```
"scripts": {  
  "test": "jest"  
}
```

---

## ◆ 20.3 Primer test con Jest

Archivo: `math.js`

javascript

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

Archivo: `math.test.js`

javascript

```
import { sumar } from './math';  
  
test("suma dos números", () => {  
  expect(sumar(2, 3)).toBe(5);  
});
```

bash

```
npm test
```

---

## ◆ 20.4 Métodos más usados en Jest

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
expect(valor).toBe(esperado);           // ===
expect(valor).toEqual(obj);             // deep equality
expect(fn).toHaveBeenCalled();           // mocks
expect(() => fn()).toThrow();            // errores
expect(array).toContain(valor);
expect(string).toMatch(/regex/);
```

---

## ♦ 20.5 Testear funciones puras

javascript

```
function capitalizar(nombre) {
  return nombre[0].toUpperCase() + nombre.slice(1).toLowerCase();
}
```

javascript

```
test("capitaliza un nombre", () => {
  expect(capitalizar("ana")).toBe("Ana");
});
```

---

## ♦ 20.6 Testear arrays, objetos y errores

javascript

```
function obtenerUsuario() {
  return { nombre: "Luis", edad: 30 };
}
```

```
test("usuario tiene nombre y edad", () => {
  expect(obtenerUsuario()).toEqual({ nombre: "Luis", edad: 30 });
});
```

javascript

```
function lanzar() {
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
    throw new Error("¡Falló!");
  }

test("lanza error", () => {
  expect(() => lanzar()).toThrow("¡Falló!");
});
```

---

## ♦ 20.7 Mocks y spies

Simulan funciones, APIs o módulos sin tener que ejecutarlos realmente.

javascript

```
const enviarCorreo = jest.fn();

enviarCorreo("hola");
expect(enviarCorreo).toHaveBeenCalledWith("hola");
```



### Ejercicio:

1. Mockea una función `enviarMensaje()` y verifica que fue llamada.
  2. Simula un delay con `setTimeout` usando `jest.useFakeTimers()`.
- 

## ♦ 20.8 Testear funciones asíncronas (promesas y `async/await`)

javascript

```
async function obtenerDatos() {
  return "OK";
}
```





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
test("devuelve OK", async () => {  
  const res = await obtenerDatos();  
  expect(res).toBe("OK");  
});
```

javascript

```
function fallar() {  
  return Promise.reject("Error");  
}  
  
test("promesa rechazada", () => {  
  return expect(fallar()).rejects.toBe("Error");  
});
```

---

## ♦ 20.9 Cobertura de código

Para ver qué partes del código se han probado:

bash

```
npx jest --coverage
```

Te indica:

- % de líneas cubiertas
- Funciones sin testear
- Archivos más débiles



---

## Proyecto final: Testea una mini librería

Supón una librería llamada `calculadora.js` con funciones:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- `sumar(a, b)`
- `restar(a, b)`
- `multiplicar(a, b)`
- `dividir(a, b)` → lanza error si `b === 0`

Tu tarea:

- ✓ Crear `calculadora.test.js`
  - ✓ Probar cada operación con varios casos
  - ✓ Testear el error al dividir por cero
  - ✓ Generar cobertura de código completa
- 



## Retos adicionales

### Básico:

- Escribe 3 tests para funciones puras de tu app.
- Testea un componente DOM básico (con Jest + JSDOM).

### Intermedio:

- Usa mocks para simular una API de usuarios.
- Testea un `fetch` usando `jest.fn()` o `jest.mock()`.

### Avanzado:




- Configura un flujo de testing automático con GitHub Actions o npm scripts.
- Usa `jest.spyOn` para espiar llamadas reales a funciones.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## ¿Qué sigue?

¿Deseas continuar con la **Sección 21: Herramientas modernas de desarrollo JS (Parcel, Vite, Webpack, Babel, ESLint, Prettier)**, o deseas que te prepare el **curso completo (Secciones 1 a 20)** como:





-  PDF o EPUB
-  App HTML navegable
-  Repositorio de GitHub con todos los ejemplos y proyectos

## Sección 21: Herramientas modernas de desarrollo JavaScript

---

### ♦ 21.1 ¿Por qué usar herramientas de build?

JavaScript moderno necesita:

-  Transpilar (Babel) para navegadores antiguos
  -  Agrupar módulos (**import/export**)
  -  Optimizar (minificar, lazy loading, cache busting)
  -  Analizar y formatear el código automáticamente
- 

### ♦ 21.2 Parcel — el bundler más fácil

bash

```
npm install -g parcel
```

Estructura mínima:



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

bash

```
project/  
  index.html  
  src/index.js
```

**index.html**

html

```
<script type="module" src="src/index.js"></script>
```

**index.js**

javascript

```
console.log("Hola desde Parcel");
```

**Comando para arrancar:**

bash

```
parcel index.html
```

Parcel detecta automáticamente [HTML](#), [JS](#), [SCSS](#), [images](#), etc.

---

## ♦ 21.3 Vite — ideal para proyectos modernos

bash

```
npm create vite@latest
```

Sigue el asistente (elige Vanilla, React, Vue...)

bash



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
cd nombre-proyecto
npm install
npm run dev
```

Características:

- ✓ Soporte nativo a **ES Modules**
  - ✓ Arranque ultra rápido
  - ✓ Hot Module Replacement
  - ✓ Preconfigurado con Babel, PostCSS, etc.
- 

## ♦ 21.4 Webpack (el más configurable)

Instalación básica:

bash

```
npm init -y
npm install webpack webpack-cli --save-dev
```

**webpack.config.js**

javascript

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist'),
  },
  mode: 'development',
};
```

bash



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

`npx webpack`

---

## ♦ 21.5 Babel — Transpilador ES6+

Convierte código moderno a versiones compatibles con navegadores antiguos.

bash

```
npm install @babel/core @babel/cli @babel/preset-env --save-dev
```

**.babelrc**

json

```
{
  "presets": ["@babel/preset-env"]
}
```

**Comando:**

bash

```
npx babel src --out-dir dist
```

---

## ♦ 21.6 ESLint — Linter para JavaScript

Verifica el estilo y posibles errores del código.

bash

```
npm install eslint --save-dev
npx eslint --init
```

Permite configurar reglas, integrar con editores, y evitar errores comunes.



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

Ejemplo de regla personalizada:

json

```
{
  "rules": {
    "semi": ["error", "always"],
    "no-unused-vars": "warn"
  }
}
```

---

## ♦ 21.7 Prettier — Formateador automático

bash

```
npm install --save-dev prettier
```

Ejecutar:

bash

```
npx prettier . --write
```

Ideal para formatear código automáticamente (se puede integrar con ESLint).

---

## ♦ 21.8 Integraciones útiles



### VSCode

- Extensiones: ESLint, Prettier, Vite, Babel, Live Server
- Configurar `formatOnSave`:

json

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
"editor.formatOnSave": true
```

## **.editorconfig** (opcional)

Configura reglas de indentación y codificación comunes para todos los editores.

---

## **Proyecto de práctica: setup profesional moderno**

1. Crea una app con Vite
  2. Usa Babel para asegurar compatibilidad
  3. Añade ESLint y Prettier
  4. Usa `npm run build` para empaquetar
  5. Sube el resultado a GitHub Pages o Netlify
- 

## **Retos por nivel**

### **Básico:**

- Inicia un proyecto con Parcel y carga un archivo JS moderno.
- Configura ESLint y corrige errores comunes.

### **Intermedio:**

- Integra Babel y ESLint en un proyecto Webpack o Vite.
- Añade un alias para rutas con Webpack (`@components`).

### **Avanzado:**



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Configura múltiples entry points con Webpack.
  - Añade lazy loading y split de código.
  - Usa `dotenv` para variables de entorno seguras.
- 

## Sección 22: Despliegue de Proyectos JavaScript — Git, GitHub, Hosting y Docker

---

### ♦ 22.1 Git y GitHub

#### ♦ ¿Qué es Git?

Sistema de control de versiones que te permite:

- Guardar el historial de tu código
- Colaborar con otras personas
- Deshacer errores
- Compartir proyectos

#### ♦ Comandos básicos

bash

```
git init
git add .
git commit -m "Primer commit"
git remote add origin https://github.com/tu-usuario/proyecto.git
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
git push -u origin main
```



## Ejercicio:

1. Crea un repositorio desde GitHub.
2. Sube tu proyecto desde VSCode o terminal.

---

## ♦ 22.2 GitHub Pages (proyectos estáticos)

Ideal para sitios **HTML + CSS + JS** sin backend.

### ♦ Pasos:

1. Sube tu proyecto a GitHub
2. Ve a **Settings** > **Pages**
3. Elige la rama **main** y la carpeta **/** o **/docs**
4. Obtendrás una URL pública:  
`https://tuusuario.github.io/proyecto/`

Compatible con Vite, Parcel, Webpack: solo asegúrate de subir los archivos del **dist/** o **build/**.

---

## ♦ 22.3 Netlify (más potente)

Hosting gratuito y fácil para proyectos modernos (Vite, React, Vue, etc.)

### ♦ Pasos:

1. Crea cuenta en [netlify.com](https://netlify.com)

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

2. Vincula tu repositorio de GitHub
3. Selecciona rama y carpeta (`dist/`, `build/`)
4. Netlify genera una URL:  
`https://nombre-proyecto.netlify.app`

💡 Puedes usar **formulario de contacto**, **funciones serverless**, y **deploy automático**.

---

## ♦ 22.4 Vercel (ideal para Next.js o JS moderno)

Alternativa a Netlify, especialmente útil para:

- SSR (Next.js)
- APIs serverless
- Autenticación y rutas dinámicas

### ♦ Pasos:

1. Crea cuenta en [vercel.com](https://vercel.com)
  2. Importa tu repositorio desde GitHub
  3. Detecta automáticamente Vite, React, etc.
  4. Despliegue instantáneo y auto-renovado
- 

## ♦ 22.5 Despliegue con Docker (intro)

Ideal si quieres:

- Aislar entornos



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Subir tu app como contenedor a un servidor o cloud

## ♦ Dockerfile básico (proyecto HTML + JS)

Dockerfile

```
FROM nginx:alpine
COPY dist/ /usr/share/nginx/html
EXPOSE 80
```

## ♦ Comandos:

bash

```
docker build -t mi-proyecto .
docker run -p 8080:80 mi-proyecto
```

➡ Accede en <http://localhost:8080>

---

## ♦ 22.6 Opciones de dominio personalizado

**En Netlify o Vercel:**

- Puedes usar tu propio dominio (ej: midominio.com)
- Solo necesitas cambiar los DNS en tu proveedor

**En GitHub Pages:**

- Crea archivo **CNAME** con el dominio
- 



## Proyecto práctico: Despliega tu app personal

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

1. Sube tu dashboard (de la sección 15) a GitHub
  2. Lanza versión pública en:
    - GitHub Pages (si es estático)
    - Netlify (si usas Vite o React)
    - Vercel (si usas rutas dinámicas)
  3. Comparte el link con amigos/clientes
- 

## Retos por nivel

### **Básico:**

- Crea una cuenta en GitHub y sube un proyecto.
- Publica un portfolio básico con GitHub Pages.

### **Intermedio:**

- Despliega una app de Vite o React en Netlify con dominio propio.
- Automatiza el deploy con GitHub Actions (CI/CD básico).

### **Avanzado:**

- Crea una imagen Docker de tu proyecto.
  - Usa Netlify/Vercel functions para añadir backend (formulario, login, etc.).
-



# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Sección 23: Web Components y Custom Elements en JavaScript

Los **Web Components** son una tecnología nativa del navegador que te permite:

- ✓ Crear componentes reutilizables y encapsulados
  - ✓ Escribir HTML, CSS y JS dentro del componente
  - ✓ Usarlos como etiquetas personalizadas (`<mi-boton>`)
- 

### ♦ 23.1 ¿Qué son los Web Components?

Son un conjunto de especificaciones que incluyen:

1. **Custom Elements** – tus propias etiquetas HTML
  2. **Shadow DOM** – encapsula estilo y estructura
  3. **HTML Templates** – define contenido reutilizable
- 

### ♦ 23.2 Crear tu primer Custom Element

javascript

```
class MiSaludo extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = `<p>¡Hola desde un componente!</p>`;  
  }  
}
```

```
customElements.define("mi-saludo", MiSaludo);
```

html

```
<mi-saludo></mi-saludo>
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## ♦ 23.3 Usar Shadow DOM

javascript

```
class MiTarjeta extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: "open" });
    shadow.innerHTML = `
      <style>
        div { border: 1px solid #ccc; padding: 10px; border-radius:
8px; }
      </style>
      <div>
        <h3>Tarjeta</h3>
        <slot></slot>
      </div>
    `;
  }
}
```

```
customElements.define("mi-tarjeta", MiTarjeta);
```

html

```
<mi-tarjeta>
  <p>Contenido interno</p>
</mi-tarjeta>
```

`slot` actúa como un “agujero” donde insertar contenido desde el exterior.

---

## ♦ 23.4 Atributos y propiedades

javascript



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
class MiBoton extends HTMLElement {
  static get observedAttributes() {
    return ["texto"];
  }

  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  connectedCallback() {
    this.render();
  }

  attributeChangedCallback(name, oldVal, newVal) {
    if (name === "texto") this.render();
  }

  render() {
    this.shadowRoot.innerHTML =
`<button>${this.getAttribute("texto")} || "Click"</button>`;
  }
}

customElements.define("mi-boton", MiBoton);
```

html

```
<mi-boton texto="Guardar"></mi-boton>
```

---

## ♦ 23.5 Eventos personalizados (CustomEvent)

javascript

```
class ContadorClick extends HTMLElement {
  constructor() {
```





# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
super();
this.attachShadow({ mode: "open" });
this.contador = 0;
this.shadowRoot.innerHTML = `<button>Clicks: 0</button>`;

this.shadowRoot.querySelector("button").addEventListener("click", ()
=> {
    this.contador++;
    this.shadowRoot.querySelector("button").textContent = `Clicks:
${this.contador}`;
    this.dispatchEvent(new CustomEvent("cambio", { detail:
this.contador }));
});
}
}

customElements.define("contador-click", ContadorClick);

html

<contador-click></contador-click>

<script>

document.querySelector("contador-click").addEventListener("cambio",
e => {
    console.log("Contador cambió a:", e.detail);
});
</script>
```

---



## Proyecto: Componente **<mi-tarea>** interactiva

1. Propiedad: `texto`
2. Botón para marcar como hecha

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

3. Evento `completada` cuando se hace clic
  4. Estilos dentro del Shadow DOM
- 

## Retos por nivel

### Básico:

- Crea un componente `<mi-mensaje>` que muestre texto y permita cambiarlo desde un atributo.
- Añade estilos personalizados dentro del Shadow DOM.

### Intermedio:

- Usa `slot` para permitir contenido HTML dinámico.
- Añade evento personalizado para notificar clics.

### Avanzado:

- Crea una galería de imágenes como componente.
- Usa `localStorage` dentro del componente para guardar datos.

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Sección 24: Integración de Web Components con APIs, almacenamiento, frameworks y rutas

---

### ◆ 24.1 Web Components + JavaScript moderno

Web Components funcionan como cualquier etiqueta del DOM, por lo tanto puedes:

- ✓ Manipularlos con `document.querySelector()`
  - ✓ Escuchar eventos personalizados
  - ✓ Pasarles datos desde JS o atributos
  - ✓ Enlazarlos a lógica de aplicación
- 

### ◆ 24.2 Integrar con API externa (fetch + componente)

- ◆ Componente `<mi-usuario>` que consulta una API:

javascript

```
class MiUsuario extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  connectedCallback() {
    const id = this.getAttribute("id");
    fetch(`https://jsonplaceholder.typicode.com/users/${id}`)
      .then(res => res.json())
      .then(usuario => {
        this.shadowRoot.innerHTML = `
          <div>
            <h3>${usuario.name}</h3>
          </div>
        `;
      });
  }
}
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
        <p>Email: ${usuario.email}</p>
    </div>
    `;
    });
}
}

customElements.define("mi-usuario", MiUsuario);

html

<mi-usuario id="3"></mi-usuario>
```

---

## ♦ 24.3 Con **localStorage** / **sessionStorage**

javascript

```
class MiContador extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
    this.valor = parseInt(localStorage.getItem("contador") || 0);
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `<button>Contador:
    ${this.valor}</button>`;

    this.shadowRoot.querySelector("button").addEventListener("click", ()
    => {
      this.valor++;
      localStorage.setItem("contador", this.valor);
      this.shadowRoot.querySelector("button").textContent =
      `Contador: ${this.valor}`;
    });
  }
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}
```

```
customElements.define("mi-contador", MiContador);
```

---

## ♦ 24.4 Comunicación entre componentes

html

```
<mi-emisor></mi-emisor>
<mi-receptor></mi-receptor>
```

javascript

```
class MiEmisor extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<button>Enviar mensaje</button>`;
    this.querySelector("button").addEventListener("click", () => {
      this.dispatchEvent(new CustomEvent("mensaje", {
        detail: "¡Hola receptor!",
        bubbles: true,
        composed: true
      }));
    });
  }
}

customElements.define("mi-emisor", MiEmisor);

class MiReceptor extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<p>Esperando mensaje...</p>`;
    this.addEventListener("mensaje", e => {
      this.innerHTML = `<p>Mensaje recibido: ${e.detail}</p>`;
    });
  }
}

customElements.define("mi-receptor", MiReceptor);
```

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

---

## ♦ 24.5 Con React, Vue o Angular

⚠ Web Components pueden integrarse, aunque no todos los frameworks los tratan igual.

### React + Web Component:

jsx

```
// Componente React
import { useRef, useEffect } from 'react';

export default function App() {
  const ref = useRef();

  useEffect(() => {
    ref.current.addEventListener("cambio", e =>
console.log(e.detail));
  }, []);

  return <contador-click ref={ref} />;
}
```

➡ React no reconoce atributos como propiedades (usa `ref` o eventos manuales).

---

## ♦ 24.6 Routing (navegación) con Web Components

Puedes combinar Web Components con rutas tipo SPA (Single Page Application).

javascript

```
class VistaInicio extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<h2>Inicio</h2><p>Bienvenido a la app</p>`;
  }
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}  
class VistaContacto extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = `<h2>Contacto</h2><p>contacto@correo.com</p>`;  
  }  
}  
  
customElements.define("vista-inicio", VistaInicio);  
customElements.define("vista-contacto", VistaContacto);  
  
function cambiarRuta(ruta) {  
  const root = document.getElementById("root");  
  root.innerHTML = "";  
  if (ruta === "/")  
    root.appendChild(document.createElement("vista-inicio"));  
  if (ruta === "/contacto")  
    root.appendChild(document.createElement("vista-contacto"));  
}  
  
window.addEventListener("popstate", () =>  
  cambiarRuta(location.pathname));  
  
document.querySelectorAll("a").forEach(enlace => {  
  enlace.addEventListener("click", e => {  
    e.preventDefault();  
    history.pushState({}, "", enlace.getAttribute("href"));  
    cambiarRuta(location.pathname);  
  });  
});  
  
html  
  
<a href="/">Inicio</a>  
<a href="/contacto">Contacto</a>  
<div id="root"></div>
```

---

# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## Proyecto: Mini SPA con Web Components + API + Storage

1. `<vista-login>`: pide nombre y lo guarda
  2. `<vista-dashboard>`: muestra saludo y datos de API
  3. Rutas SPA (sin recarga)
  4. Usa `localStorage` para persistencia
  5. Notificación con `<mi-toast>` si algo falla
- 

## Retos por nivel

### Básico:

- Crea un `<mi-buscador>` que use una API y muestre resultados.
- Guarda el último término buscado en `localStorage`.

### Intermedio:

- Crea componentes que se comuniquen entre sí con eventos.
- Haz una SPA con 2 vistas y rutas con `history.pushState`.

### Avanzado:

- Integra Web Components en un proyecto de React o Vue.
- Usa Web Components dentro de una app desplegada en Netlify.





# Curso Completo de JavaScript — Desde Cero hasta Avanzado



## Sección 25: DOM Virtual, JSX y Mini Framework tipo React

---

### ♦ 25.1 ¿Qué es el DOM virtual?

El **DOM virtual (Virtual DOM)** es una **representación en memoria del DOM real**. Cuando cambia algo en la interfaz:

1. Se actualiza el DOM virtual
2. Se compara con el anterior (diffing)
3. Solo se modifica en pantalla lo que cambió

Esto lo hace **más rápido** y eficiente que modificar el DOM real directamente.

---

### ♦ 25.2 ¿Qué es JSX?

JSX es una extensión de JavaScript (no obligatoria) que parece HTML:

jsx

```
const elemento = <h1>Hola Mundo</h1>;
```

JSX **no es obligatorio** en React ni en tu framework. Se compila a:

js

```
const elemento = React.createElement("h1", null, "Hola Mundo");
```

---

### ♦ 25.3 Crea tu propio **createElement**



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

javascript

```
function createElement(tipo, props, ...hijos) {
  return {
    tipo,
    props: props || {},
    hijos: hijos.flat().map(h =>
      typeof h === "object" ? h : crearTexto(h)
    )
  };
}

function crearTexto(texto) {
  return {
    tipo: "TEXT_ELEMENT",
    props: { nodeValue: texto },
    hijos: []
  };
}
```

---

## ♦ 25.4 Renderizar un árbol virtual al DOM real

javascript

```
function render(elemento, contenedor) {
  const dom =
    elemento.tipo === "TEXT_ELEMENT"
      ? document.createTextNode("")
      : document.createElement(elemento.tipo);

  Object.entries(elemento.props || {}).forEach(([k, v]) => {
    if (k !== "children") dom[k] = v;
  });

  elemento.hijos.forEach(hijo => render(hijo, dom));
  contenedor.appendChild(dom);
}
```



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

```
}
```

---

## ♦ 25.5 Mini ejemplo: renderizar manualmente

javascript

```
const app = createElement("div", null,
  createElement("h1", null, "Hola"),
  createElement("button", { onclick: () => alert("Click") },
    "Púlsame")
);

render(app, document.getElementById("root"));
```

---

## ♦ 25.6 Crear una API tipo JSX sin compilador

Podemos imitar JSX usando una función `h` y `babel` si queremos.

javascript

```
/** @jsx h */
function h(tipo, props, ...hijos) {
  return createElement(tipo, props, ...hijos);
}
```

javascript

```
const app = (
  <div>
    <h2>Hola JSX</h2>
    <p>Soy un componente sin React</p>
  </div>
);
```

---



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 25.7 Añadir estado (tipo `useState`)

javascript

```
let estado = [];  
let indice = 0;  
  
function useState(valorInicial) {  
  const i = indice;  
  estado[i] = estado[i] ?? valorInicial;  
  
  function setValor(nuevo) {  
    estado[i] = nuevo;  
    indice = 0;  
    render(app(), document.getElementById("root"));  
  }  
  
  return [estado[i++], setValor];  
}
```

---

## ♦ 25.8 Hooks personalizados

Puedes crear tu propio `useContador()`, `useStorage()`, etc.

javascript

```
function useContador(inicio) {  
  const [valor, setValor] = useState(inicio);  
  
  function incrementar() {  
    setValor(valor + 1);  
  }  
  
  return [valor, incrementar];  
}
```

---



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

## ♦ 25.9 Mini router (SPA sin recarga)

javascript

```
function Router(rutas) {  
  const path = location.pathname;  
  return rutas[path] ? rutas[path]() : rutas["/404"]();  
}  
  
const app = () => Router({  
  "/": () => createElement("h1", null, "Inicio"),  
  "/contacto": () => createElement("h1", null, "Contacto"),  
  "/404": () => createElement("h1", null, "Página no encontrada")  
});
```

---



## Proyecto final: miniReact (framework funcional)

Tu objetivo:

- `createElement()` → JSX-like
  - `render()` → renderizar al DOM
  - `useState()` → mantener estado
  - `useEffect()` (básico opcional)
  - `Router()` → navegación SPA
  - `Componentes()` → funciones que devuelven nodos
- 



## Retos por nivel

**Básico:**



# Curso Completo de JavaScript — Desde Cero hasta Avanzado

- Implementa `createElement`, `render` y crea una app simple.
- Añade soporte para eventos (onclick, etc).

## Intermedio:

- Implementa `useState()` y render condicional.
- Crea un contador con botones + estado.

## Avanzado:

- Implementa routing manual.
- Añade `useEffect` básico que ejecute lógica tras render.