

```

        marca: 'Toyota',
        modelo: 'Corolla',
        color: 'rojo'
    };

// Modificación
coche.color = 'azul';

// Adición
coche.anio = 2020;

// Acceso con corchetes
const clave = 'modelo';
console.log(`El valor de ${clave} es: ${coche[clave]}`); // Salida: El valor de modelo es: Corolla

// Eliminación
delete coche.marca;

console.log(coche); // Salida: { modelo: 'Corolla', color: 'azul', anio: 2020 }

```

### 3. Métodos Object.keys(), Object.values(), Object.entries()

Estos métodos estáticos proporcionan formas sencillas de obtener las claves, valores o pares clave-valor de un objeto.

Método	Descripción	Resultado
--------	-------------	-----------

Object.keys(obj)	Devuelve un array de las <b>claves</b> (nombres de propiedad).	['a', 'b', 'c']
Object.values(obj)	Devuelve un array de los <b>valores</b> de las propiedades.	[1, 2, 3]
Object.entries(obj)	Devuelve un array de arrays [clave, valor].	[['a', 1], ['b', 2], ['c', 3]]

## Ejemplo con los métodos

JavaScript

```
const config = {
  tema: 'oscuro',
  notificaciones: true,
  idioma: 'es'
};

// Obtener todas las claves
const claves = Object.keys(config);
```

```
console.log(claves); // Salida: ['tema', 'notificaciones', 'idioma']
```

```
// Obtener todos los valores
```

```
const valores = Object.values(config);
```

```
console.log(valores); // Salida: ['oscuro', true, 'es']
```

```
// Obtener pares clave-valor
```

```
const pares = Object.entries(config);
```

```
console.log(pares);
```

```
/*
```

```
Salida: [
```

```
  ['tema', 'oscuro'],
```

```
  ['notificaciones', true],
```

```
  ['idioma', 'es']
```

```
]
```

```
*/
```

---

## Métodos y Propiedades Estáticas

Los **métodos estáticos** son funciones que pertenecen a la **clase o constructor** en sí, no a una instancia específica de esa clase. Se invocan directamente en el nombre de la clase, no en un objeto creado a partir de ella.

**Concepto Clave:** Un método estático no puede acceder directamente a las propiedades no estáticas de una instancia (porque no tiene acceso a un this de instancia).

### Ejemplo de Método Estático

## JavaScript

```
class Calculadora {  
    // Propiedad estática: pertenece a la clase  
    static PI = 3.14159;  
  
    // Método estático: se llama en la clase Calculadora.  
    static sumar(a, b) {  
        return a + b;  
    }  
  
    // Método de instancia: se llama en una instancia de Calculadora.  
    restar(a, b) {  
        return a - b;  
    }  
}  
  
// Uso de la propiedad estática  
console.log(Calculadora.PI); // Salida: 3.14159  
  
// Uso del método estático: se llama directamente en la clase  
const resultadoSuma = Calculadora.sumar(10, 5);  
console.log(resultadoSuma); // Salida: 15  
  
// Intentar llamar un método estático en una instancia (no funcionará)  
// const calc = new Calculadora();
```

```
// calc.sumar(1, 1); // Error: calc.sumar is not a function

// Uso del método de instancia: requiere crear un objeto primero
const calc = new Calculadora();
const resultadoResta = calc.restar(10, 5);
console.log(resultadoResta); // Salida: 5
```

---

## Local Storage (Almacenamiento Local)

**localStorage** es un mecanismo de almacenamiento web que permite a las aplicaciones web almacenar datos de forma persistente en el navegador del usuario. Los datos no tienen fecha de caducidad (a diferencia de las `sessionStorage` que se borran al cerrar la pestaña/navegador).

**Importante:** Solo puede almacenar **cadenas de texto**. Por eso, es fundamental usar `JSON.stringify()` y `JSON.parse()` (serialización/deserialización) para guardar y recuperar objetos o arrays.

Método / Propiedad	Descripción
<code>localStorage.setItem(clave, valor)</code>	Almacena un par clave/valor. El valor <b>debe</b> ser una cadena.
<code>localStorage.getItem(clave)</code>	Recupera el valor asociado a la clave. Devuelve null si la clave no existe.
<code>localStorage.removeItem(clave)</code>	Elimina el par clave/valor del

	almacenamiento.
localStorage.clear()	Elimina <b>todo</b> lo almacenado para el dominio actual.
localStorage.length	Propiedad que devuelve el número de entradas almacenadas.

## Ejemplo con localStorage

JavaScript

```
const configuracion = {
  darkMode: true,
  ultimaVisita: new Date().toLocaleDateString()
};

// 1. Guardar un objeto (REQUIERE SERIALIZACIÓN)
// Convierte el objeto a una cadena JSON
const configJSON = JSON.stringify(configuracion);
localStorage.setItem('userConfig', configJSON);
console.log('Configuración guardada.');
```

```
// 2. Recuperar el objeto (REQUIERE DESERIALIZACIÓN)
const configGuardadaJSON = localStorage.getItem('userConfig');

if (configGuardadaJSON) {
    // Convierte la cadena JSON de vuelta a un objeto JavaScript
    const configRecuperada = JSON.parse(configGuardadaJSON);

    console.log('Configuración recuperada:');
    console.log(`Modo oscuro: ${configRecuperada.darkMode}`);
    console.log(`Última visita: ${configRecuperada.ultimaVisita}`);

    // 3. Eliminar la clave después de usarla (opcional)
    // localStorage.removeItem('userConfig');
} else {
    console.log('No se encontró configuración.');
}
```

---

## 1. Manipulación Avanzada de Propiedades: Object.defineProperty()

### Descripción

El método estático **Object.defineProperty()** permite agregar una nueva propiedad directamente a un objeto, o modificar una propiedad existente, y obtener un **control preciso** sobre los atributos internos de esa propiedad.

A diferencia de la asignación simple (objeto.propiedad = valor), este método permite definir los siguientes atributos (llamados *descriptores de propiedades*):

Descriptor	Tipo	Descripción
<b>value</b>	<i>Data</i>	El valor asociado con la propiedad.
<b>writable</b>	<i>Data</i>	Si es false, el valor de la propiedad no puede ser cambiado (read-only).
<b>enumerable</b>	<i>Config</i>	Si es false, la propiedad no aparecerá en bucles como for...in ni en Object.keys().

<b>configurable</b>	<i>Config</i>	Si es false, el tipo de descriptor no puede ser cambiado (de data a accessor o viceversa), y no puede ser eliminada (delete).
<b>get / set</b>	Accessor	Permite definir una función para obtener (get) o establecer (set) el valor de la propiedad, creando una propiedad <b>calculada</b> .

## Ejemplo

Vamos a crear un objeto config y definir una propiedad API\_KEY que sea **de solo lectura** (writable: false) y que **no sea visible** en iteraciones (enumerable: false).

JavaScript

```
const app = {};
// Definir la propiedad 'nombre' de forma normal
```

```
app.nombre = 'MiApp';

// Definir una propiedad 'API_KEY' con Object.defineProperty()
Object.defineProperty(app, 'API_KEY', {
  value: 'abcd-1234-efgh',
  writable: false,    // No se puede reasignar el valor
  enumerable: false,  // No aparece al iterar el objeto
  configurable: false // No se puede eliminar ni cambiar sus descriptores
});

// --- Intentos de modificación y acceso ---

console.log(app.nombre);    // Salida: MiApp
console.log(app.API_KEY);   // Salida: abcd-1234-efgh (Se puede leer)

// 1. Intento de modificar (falla silenciosamente o lanza error en modo estricto)
app.API_KEY = 'NUEVO_VALOR';
console.log(app.API_KEY);   // Salida: abcd-1234-efgh (El valor no cambió)

// 2. Intento de enumerar las propiedades
console.log(Object.keys(app)); // Salida: ['nombre'] (API_KEY está oculta)

// 3. Intento de eliminar (falla porque configurable es false)
delete app.API_KEY;
console.log(app.API_KEY);   // Salida: abcd-1234-efgh (No se eliminó)
```

---

## 2. Más Ejemplos con Array.prototype.reduce()

## Descripción

El método **reduce()** aplica una función de *callback* (la función reductora) contra un **acumulador** y cada elemento en el array (de izquierda a derecha) para reducirlo a un **único valor**. Es extremadamente versátil y puede usarse para operaciones más allá de la simple suma.

## Ejemplo 1: Agrupar Elementos (Clustering)

Se utiliza reduce() para transformar un array plano de objetos en un objeto donde las propiedades son las categorías, y los valores son arrays de objetos que pertenecen a esa categoría.

JavaScript

```
const productos = [
  { nombre: 'Manzana', categoria: 'Fruta' },
  { nombre: 'Lechuga', categoria: 'Verdura' },
  { nombre: 'Plátano', categoria: 'Fruta' },
  { nombre: 'Zanahoria', categoria: 'Verdura' }
];
```

```
// Objetivo: Agrupar por la propiedad 'categoria'  
const agrupados = productos.reduce((acumulador, productoActual) => {  
  const categoria = productoActual.categoría;  
  
  // Si la categoría no existe en el acumulador, la inicializa como un array vacío  
  if (!acumulador[categoría]) {  
    acumulador[categoría] = [];  
  }  
  
  // Agrega el producto al array de su categoría  
  acumulador[categoría].push(productoActual.nombre);  
  
  return acumulador;  
}, {}); // Valor inicial: un objeto vacío {}  
  
console.log(agrupados);  
/*  
Salida:  
Fruta: [ 'Manzana', 'Plátano' ],  
Verdura: [ 'Lechuga', 'Zanahoria' ]  
*/
```

## Ejemplo 2: Aplanar un Array (Flattening)

Se utiliza reduce() para convertir un array que contiene otros arrays (array multidimensional) en un array de una sola dimensión.

## JavaScript

```
const colecciones = [
  [1, 2],
  [3, 4, 5],
  [6]
];

// Objetivo: Convertir a [1, 2, 3, 4, 5, 6]
const aplanado = colecciones.reduce((acumulador, arrayActual) => {
  // Concatena el array actual al acumulador
  return acumulador.concat(arrayActual);
}, []); // Valor inicial: un array vacío []

console.log(aplanado); // Salida: [1, 2, 3, 4, 5, 6]

// Nota: Para arrays simples, también se puede usar Array.prototype.flat()
```