Arrays en Javascript. Creación, métodos más comunes, usos y ejemplos.

1. Literal ([])

Es la forma más común y directa para crear arrays. Los elementos se separan por comas dentro de corchetes.

```
const arr = [1, 2, 3, 4]; // Array con 4 elementos
const emptyArr = []; // Array vacío
```

- **Ventajas**: Simple y legible.
- **Usos comunes**: Cuando conocemos los valores iniciales o queremos un array vacío para llenarlo después.

2. Constructor new Array()

Crea un array utilizando el constructor de la clase `Array`. Puede recibir:

- **Un único número**: Crea un array vacío con esa longitud.
- **Varios valores**: Crea un array con esos valores como elementos.

```
const arr = new Array(4); // Array vacío con longitud 4
const arr2 = new Array(1, 2, 3, 4); // Array con elementos 1, 2, 3, 4
```

- **Nota**: Si pasamos un único número, se interpreta como longitud, no como elemento.
- **Desventajas**: Menos legible y más propenso a errores al usar números.

3. Array.of()

Crea un array a partir de los valores pasados como argumentos. Siempre trata los argumentos como elementos, incluso un único número.

```
const arr = Array.of(4); // Array con el número 4 como único elemento
const arr2 = Array.of(1, 2, 3, 4); // Array con 1, 2, 3, 4
```

- **Ventaja**: Evita la confusión del constructor new `Array()`.
- **Usos comunes**: Creación rápida de arrays sin ambigüedad.

4. Array.from()

Crea un array a partir de un objeto iterable (como cadenas o sets) o un objeto con una propiedad `length`. Permite un mapeo opcional para transformar los elementos durante la creación.

```
const arr = Array.from('1234'); // ['1', '2', '3', '4'] (de una cadena)
const arr2 = Array.from([10, 20, 30], x => x / 10); // [1, 2, 3]
const arr3 = Array.from({ length: 5 }, (_, i) => i + 1); // [1, 2, 3, 4, 5]
```

- **Ventaja**: Útil para transformar iterables en arrays o crear arrays personalizados.
- **Usos comunes**: Conversión de iterables o creación de arrays con valores calculados.

5. Spread Operator (...)

Crea un nuevo array copiando elementos de un iterable (como otro array, string, etc.).

```
const arr = [...'abc']; // ['a', 'b', 'c'] (de una cadena)
const arr2 = [...[1, 2, 3], 4, 5]; // [1, 2, 3, 4, 5] (combinación)
```

- **Ventaja**: Sintaxis corta y potente para copiar o combinar arrays.
- **Usos comunes**: Duplicar arrays o combinar arrays y valores.

6. Por asignación dinámica

Permite crear un array vacío y luego añadir elementos en posiciones específicas.

```
const arr = [];
arr[0] = 1; // Añadir en posición 0
arr[1] = 2; // Añadir en posición 1
console.log(arr); // [1, 2]
```

- **Ventaja**: Flexibilidad para modificar el contenido del array dinámicamente.
- **Desventaja**: Menos eficiente y menos legible que las otras formas.

Resumen de casos de uso:

- Usa **literales (`[]`)** para la mayoría de los casos por su simplicidad.

- Usa `Array.of()` para evitar ambigüedades al crear arrays con un solo número.
- Usa `Array.from()` para convertir iterables o crear arrays personalizados.
- Usa **spread (`...`)** para combinar o copiar arrays rápidamente.

Operador Spread (...) en JavaScript

El operador spread (`...`) se utiliza para "descomponer" o "expandir" elementos de un iterable (como arrays, strings, o incluso objetos) en lugares donde se esperan múltiples valores. Es especialmente útil para trabajar con arrays, y permite realizar operaciones como copiar, combinar y transformar datos de manera concisa y legible.

Cómo funciona en arrays

El operador spread expande los elementos de un array (u otro iterable) en su posición correspondiente.

```
const arr = [1, 2, 3];
console.log(...arr); // Salida: 1 2 3 (cada elemento se expande)
```

Casos de uso comunes

1. Crear una copia de un array

El operador spread permite copiar un array sin modificar el original (operación inmutable).

```
const original = [1, 2, 3];
const copy = [...original];
copy.push(4);

console.log(original); // [1, 2, 3] (sin cambios)
console.log(copy);     // [1, 2, 3, 4] (nueva copia modificada)
```

2. Combinar arrays

Puedes combinar varios arrays en uno nuevo de forma sencilla.

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const combined = [...arr1, ...arr2];

console.log(combined); // [1, 2, 3, 4]
```

También puedes insertar elementos adicionales al combinar arrays.

```
const combined = [0, ...arr1, 5, ...arr2];
console.log(combined); // [0, 1, 2, 5, 3, 4]
```

3. Convertir un iterable en un array

El operador spread permite transformar iterables como strings, sets, o incluso objetos tipo arguments en arrays.

```
const str = 'hello';
const charArray = [...str];
console.log(charArray); // ['h', 'e', 'l', 'l', 'o']

const set = new Set([1, 2, 3]);
const arrFromSet = [...set];
console.log(arrFromSet); // [1, 2, 3]
```

4. Eliminar duplicados (con Set)

El operador spread facilita la eliminación de valores duplicados cuando se combina con `Set`.

```
const duplicates = [1, 2, 2, 3, 3, 3];
const unique = [...new Set(duplicates)];
console.log(unique); // [1, 2, 3]
```

Un `Set` en JavaScript es una estructura de datos que permite almacenar **valores únicos** de cualquier tipo, ya sean primitivos u objetos. A diferencia de un array, un `Set` no permite duplicados, lo que lo hace ideal para manejar conjuntos de valores únicos.

Características principales:

1. **Sin duplicados**: No se pueden agregar valores repetidos.
2. **Orden de inserción**: Mantiene el orden en el que se añaden los elementos.
3. **Métodos útiles**: Proporciona métodos para agregar, eliminar y comprobar la existencia de elementos.

Sintaxis básica:

```
let miSet = new Set();
```

Métodos y propiedades:

- `add(valor)`: Agrega un valor al `Set`.
- `delete(valor)`: Elimina un valor del `Set`.

- `has(valor)`: Devuelve `true` si el valor está en el `Set`, de lo contrario, `false`.
- `size`: Devuelve el número de elementos en el `Set`.
- `clear()`: Vacía el `Set`.

Ejemplo de uso:

```
let numeros = new Set();

// Agregar elementos
numeros.add(1);
numeros.add(2);
numeros.add(3);
numeros.add(2); // No se agrega porque ya existe

console.log(numeros); // Output: Set(3) { 1, 2, 3 }

// Verificar si un elemento existe
console.log(numeros.has(2)); // Output: true
console.log(numeros.has(5)); // Output: false

// Eliminar un elemento
numeros.delete(2);
console.log(numeros); // Output: Set(2) { 1, 3 }

// Tamaño del Set
console.log(numeros.size); // Output: 2
```

Un `Set` es útil cuando necesitas eliminar duplicados o trabajar con colecciones de valores únicos en tu aplicación.

5. Uso en funciones

Cuando una función espera múltiples argumentos, el operador spread permite expandir un array para pasar cada elemento como un argumento separado.

```
const numbers = [1, 2, 3];
const sum = (a, b, c) => a + b + c;

console.log(sum(...numbers)); // 6
```

6. Separar elementos de un array

Puedes extraer elementos específicos del inicio, final o cualquier posición usando el spread en combinación con desestructuración.

```
const arr = [1, 2, 3, 4, 5];
const [first, ...rest] = arr;
```

```
console.log(first); // 1
console.log(rest);  // [2, 3, 4, 5]
```

Spread vs. Rest

Aunque parecen similares, el operador spread (`...`) y el operador rest (`...`) tienen propósitos opuestos:

- **Spread (`...`)**: Expande elementos de un iterable.
- **Rest (`...`)**: Agrupa múltiples elementos en una sola variable.

Ejemplo:

```
// Spread
const arr = [1, 2, 3];
const newArr = [...arr, 4]; // Expande 'arr' en otro array
console.log(newArr); // [1, 2, 3, 4]

// Rest
const [first, ...rest] = arr; // Agrupa todo después del primer elemento
console.log(rest); // [2, 3]
```

Limitaciones del operador spread

1. **No copia estructuras anidadas profundamente**:

Si un array contiene objetos u otros arrays, la copia será superficial, no profunda.

```
const nested = [[1], [2]];
const copy = [...nested];
copy[0][0] = 99;

console.log(nested); // [[99], [2]] (el original también cambia)
```

2. **Solo funciona con iterables**:

Si intentas usar spread con un valor no iterable, lanzará un error.

```
const notIterable = 123;
// const result = [...notIterable]; // Error: notIterable is not iterable
```

Resumen de los métodos más habituales para arrays en JavaScript

1. Métodos de adición/eliminación

- `push()`: Añade elementos al final.

```
const arr = [1, 2];
arr.push(3); // [1, 2, 3]
```

- `pop()`: Elimina el último elemento.

```
const arr = [1, 2, 3];
arr.pop(); // [1, 2]
```

- `unshift()`: Añade elementos al inicio.

```
const arr = [2, 3];
arr.unshift(1); // [1, 2, 3]
```

- `shift()`: Elimina el primer elemento.

```
const arr = [1, 2, 3];
arr.shift(); // [2, 3]
```

2. Métodos de iteración

- `forEach()`: Itera sobre cada elemento.

```
[1, 2, 3].forEach(x => console.log(x)); // 1, 2, 3
```

- `map()`: Crea un nuevo array aplicando una función.

```
const doubled = [1, 2, 3].map(x => x * 2); // [2, 4, 6]
```

- `filter()`: Crea un nuevo array con elementos que cumplen una condición.

```
const evens = [1, 2, 3, 4].filter(x => x % 2 === 0); // [2, 4]
```

- `reduce()`: Reduce el array a un único valor.

```
const sum = [1, 2, 3].reduce((acc, x) => acc + x, 0); // 6
```

3. Métodos de búsqueda

- `find()`: Encuentra el primer elemento que cumple una condición.

```
const result = [1, 2, 3].find(x => x > 1); // 2
```

- `findIndex()`: Encuentra el índice del primer elemento que cumple una condición.

```
const index = [1, 2, 3].findIndex(x => x > 1); // 1
```

- `includes()`: Verifica si un elemento está en el array.

```
[1, 2, 3].includes(2); // true
```

4. Métodos de orden y transformación

- `sort()`: Ordena los elementos (por defecto, alfabéticamente).

```
const arr = [3, 1, 2];
arr.sort((a, b) => a - b); // [1, 2, 3]
```

- `reverse()`: Invierte el orden de los elementos.

```
const arr = [1, 2, 3];
arr.reverse(); // [3, 2, 1]
```

- `join()`: Une todos los elementos en un string.

```
const str = [1, 2, 3].join('-'); // "1-2-3"
```

5. Métodos de extracción y manipulación

- `slice()`: Extrae una porción del array sin modificarlo.

```
const arr = [1, 2, 3, 4];
const sub = arr.slice(1, 3); // [2, 3]
```

- `splice()`: Modifica el array eliminando o añadiendo elementos.

```
const arr = [1, 2, 3];
arr.splice(1, 1, 5); // [1, 5, 3] (reemplaza 2 por 5)
```

6. Otros útiles

- `concat()`: Combina arrays.

```
const combined = [1, 2].concat([3, 4]); // [1, 2, 3, 4]
```

- `indexOf()`: Encuentra la primera posición de un elemento.

```
const index = [1, 2, 3].indexOf(2); // 1
```

- `every()`: Verifica si todos los elementos cumplen una condición.

```
[1, 2, 3].every(x => x > 0); // true
```

- `some()`: Verifica si algún elemento cumple una condición.

```
[1, 2, 3].some(x => x > 2); // true
```

Resumen rápido de propósitos:

- **Añadir/Eliminar elementos**: `push`, `pop`, `unshift`, `shift`.

- **Buscar elementos**: `find`, `findIndex`, `includes`, `indexOf`.
- **Modificar estructura**: `map`, `filter`, `sort`, `splice`, `slice`.
- **Comprobar condiciones**: `every`, `some`.
- **Recorrer el array**: `forEach`, `reduce`.
- **Combinar o transformar**: `concat`, `join`.

split() en JavaScript

El método `split()` se utiliza en cadenas (**strings**) para dividirlas en un array de partes, basándose en un separador especificado.

Sintaxis

```
string.split(separador, limite);
```

- `separador`: Especifica cómo dividir la cadena. Puede ser un carácter, palabra, expresión regular o `undefined` (en cuyo caso devuelve el string completo como un array con un único elemento).
- `limite` (opcional): Número máximo de elementos en el array resultante.

Ejemplos básicos

1. Dividir por un carácter

```
const str = "uno,dos,tres";
const arr = str.split(",");
console.log(arr); // ["uno", "dos", "tres"]
```

2. Dividir por espacios

```
const str = "Aprender JavaScript es divertido";
const words = str.split(" ");
console.log(words); // ["Aprender", "JavaScript", "es", "divertido"]
```

3. Limitar el número de partes

```
const str = "uno,dos,tres,cuatro";
const arr = str.split(",", 2);
console.log(arr); // ["uno", "dos"]
```

Usos comunes

1. Convertir una cadena en caracteres individuales

```
const str = "hola";
const chars = str.split("");
console.log(chars); // ["h", "o", "l", "a"]
```

2. Procesar datos estructurados

```
const data = "nombre;apellido;edad";
const fields = data.split(";");
console.log(fields); // ["nombre", "apellido", "edad"]
```

3. Invertir una cadena

```
const str = "JavaScript";
const reversed = str.split("").reverse().join("");
console.log(reversed); // "tpircSavaJ"
```

Importante:

- **No funciona directamente en arrays.** Para trabajar con arrays, utiliza métodos como `join()` o `map()` en combinación con `split()` cuando sea necesario.

A continuación se muestran varios ejemplos que combinan el uso de `split()` con los métodos más comunes de arrays:

1. split() con map()

Ejemplo: Convertir un string de nombres en un array y quitar espacios innecesarios con `trim()`.

```
const nombres = "  Ana,  Luis , Pedro , María  ";
const nombresLimpios = nombres.split(",").map(nombre => nombre.trim());
console.log(nombresLimpios); // ["Ana", "Luis", "Pedro", "María"]
```

2. split() con filter()

Ejemplo: Dividir una lista de palabras y filtrar solo las que tienen más de 3 letras.

```
const palabras = "sol, luna, estrella, sol";
const palabrasFiltradas = palabras.split(",").map(p => p.trim()).filter(p
=> p.length > 3);
console.log(palabrasFiltradas); // ["luna", "estrella"]
```

3. split() con reduce()

Ejemplo: Calcular la longitud total de todas las palabras en un string.

```
const frase = "esto, es, una, prueba";
const longitudTotal = frase.split(",").map(p => p.trim()).reduce((total,
palabra) => total + palabra.length, 0);
console.log(longitudTotal); // 16
```

4. split() con sort()

Ejemplo: Ordenar alfabéticamente una lista de palabras extraídas de un string.

```
const palabras = "sol, estrella, luna, planeta";
const palabrasOrdenadas = palabras.split(",").map(p => p.trim()).sort();
console.log(palabrasOrdenadas); // ["estrella", "luna", "planeta", "sol"]
```

5. split() con reverse()

Ejemplo: Invertir el orden de palabras en un string.

```
const frase = "manzana, pera, plátano";
const palabrasInvertidas = frase.split(",").map(p => p.trim()).reverse();
console.log(palabrasInvertidas); // ["plátano", "pera", "manzana"]
```

6. split() con join()

Ejemplo: Convertir un string en un array y luego reconstruirlo como un string con otro separador.

```
const texto = "rojo, verde, azul, amarillo";
const nuevoTexto = texto.split(",").map(p => p.trim()).join(" | ");
console.log(nuevoTexto); // "rojo | verde | azul | amarillo"
```

7. split() con includes()

Ejemplo: Comprobar si una palabra específica está en una lista.

```
const texto = "  gato, perro, loro, pez ";
const animales = texto.split(",").map(p => p.trim());
console.log(animales.includes("perro")); // true
console.log(animales.includes("tigre")); // false
```

8. split() con every()

Ejemplo: Verificar si todas las palabras en una lista tienen más de 2 caracteres.

```
const texto = "sol, mar, luz";
const todasSonLargas = texto.split(",").map(p => p.trim()).every(p =>
p.length > 2);
console.log(todasSonLargas); // true
```

9. split() con some()

Ejemplo: Verificar si al menos una palabra en la lista contiene la letra "a".

```
const texto = "luz, sol, mar, nube";
const algunaContieneA = texto.split(",").map(p => p.trim()).some(p =>
p.includes("a"));
console.log(algunaContieneA); // true
```

10. split() con indexOf()

Ejemplo: Encontrar la posición de una palabra en la lista.

```
const texto = "cielo, tierra, agua, fuego";
const palabras = texto.split(",").map(p => p.trim());
console.log(palabras.indexOf("agua")); // 2
```

11. split() con slice()

Ejemplo: Extraer las dos primeras palabras de una lista.

```
const texto = "manzana, pera, plátano, uva";
const primerasDos = texto.split(",").map(p => p.trim()).slice(0, 2);
console.log(primerasDos); // ["manzana", "pera"]
```

12. split() con splice()

Ejemplo: Reemplazar una palabra en un array creado a partir de un string.

```
const texto = "uno, dos, tres, cuatro";
const palabras = texto.split(",").map(p => p.trim());
palabras.splice(1, 1, "mil"); // Reemplaza "dos" con "mil"
console.log(palabras); // ["uno", "mil", "tres", "cuatro"]
```

13. split() con concat()

Ejemplo: Combinar un array de palabras con otro array.

```
const texto = "león, tigre, jaguar";
const animales1 = texto.split(",").map(p => p.trim());
const animales2 = ["puma", "pantera"];
const todosLosAnimales = animales1.concat(animales2);
console.log(todosLosAnimales); // ["león", "tigre", "jaguar", "puma",
"pantera"]
```

14. split() con pop() y push()

Ejemplo: Extraer la última palabra y añadir una nueva palabra.

```
const texto = "lápiz, papel, goma, tijeras";
const objetos = texto.split(",").map(p => p.trim());
const ultimoObjeto = objetos.pop(); // Elimina "tijeras"
objetos.push("regla"); // Añade "regla"
console.log(objetos); // ["lápiz", "papel", "goma", "regla"]
console.log(ultimoObjeto); // "tijeras"
```

15. split() con shift() y unshift()

Ejemplo: Eliminar la primera palabra y añadir una nueva al principio.

```
const texto = "carro, moto, bicicleta, avión";
const vehiculos = texto.split(",").map(p => p.trim());
const primerVehiculo = vehiculos.shift(); // Elimina "carro"
vehiculos.unshift("camión"); // Añade "camión"
console.log(vehiculos); // ["camión", "moto", "bicicleta", "avión"]
console.log(primerVehiculo); // "carro"
```

16. split() con trim() directamente

Ejemplo: Limpiar espacios de una cadena antes de dividirla.

```
const texto = "    hola,    mundo,  cómo, estás    ";
const palabras = texto.trim().split(",").map(p => p.trim());
console.log(palabras); // ["hola", "mundo", "cómo", "estás"]
```

Ejercicios de ejemplo

Ejercicio 1: Sistema de gestión de inventario

Descripción: Este ejercicio utiliza un bucle y métodos de arrays (`push`, `splice`, `sort`) para gestionar un inventario de productos. Los resultados se muestran en la página usando elementos creados dinámicamente.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sistema de Inventario</title>
</head>
<body>
    <script>
        // Esperamos a que el DOM esté completamente cargado
        document.addEventListener("DOMContentLoaded", function () {
            // 1. Inicializamos un array con los productos iniciales
            const inventario = ["Manzana", "Pera", "Plátano", "Naranja"];
```

```
            // 2. Creamos un título para el sistema de inventario
            const titulo = document.createElement("h1");
            titulo.textContent = "Sistema de Inventario"; // Texto del
título
            document.body.appendChild(titulo); // Añadimos el título al DOM

            // 3. Creamos un contenedor para mostrar los resultados de las
acciones
            const salida = document.createElement("div");
            salida.style.marginTop = "10px"; // Añadimos un margen superior
para estilo
            document.body.appendChild(salida);

            // 4. Iniciamos un bucle infinito para interactuar con el
usuario
            while (true) {
                // Pedimos al usuario que elija una acción
                const accion = prompt("¿Qué deseas hacer? (añadir,
eliminar, listar, salir)");

                // Si el usuario elige "salir", rompemos el bucle
                if (accion === "salir") break;

                // Si elige "añadir", le pedimos el producto a añadir
                if (accion === "añadir") {
                    const producto = prompt("Introduce el producto:");
                    // Verificamos si el producto no está en el inventario
                    if (!inventario.includes(producto)) {
                        inventario.push(producto); // Añadimos el producto
al final del array
                        salida.textContent = `${producto} añadido al
inventario.`; // Mostramos mensaje
                    } else {
                        salida.textContent = `${producto} ya existe en el
inventario.`; // Producto duplicado
                    }
                }
                // Si elige "eliminar", buscamos y eliminamos el producto
del inventario
                else if (accion === "eliminar") {
                    const producto = prompt("Introduce el producto a
eliminar:");
                    const index = inventario.indexOf(producto); //
Obtenemos el índice del producto
                    if (index !== -1) {
                        inventario.splice(index, 1); // Eliminamos el
producto del array
                        salida.textContent = `${producto} eliminado del
inventario.`; // Confirmamos eliminación
                    } else {
                        salida.textContent = `${producto} no se encuentra
en el inventario.`; // Producto no encontrado
                    }
                }
                // Si elige "listar", mostramos el inventario ordenado
                else if (accion === "listar") {
                    salida.textContent = "Inventario actual: " +
inventario.sort().join(", ");
```

```
                }
                // Si introduce una acción inválida, mostramos un mensaje
de error
                else {
                    salida.textContent = "Acción no válida.";
                }
            }
        });
    </script>
</body>
</html>
```

Ejercicio 2: Análisis de resultados de un examen

Descripción: Este ejercicio utiliza métodos como `filter` y `reduce` para procesar un array de notas. Se calcula la media y se separan las notas aprobadas y suspendidas.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Análisis de Resultados</title>
</head>
<body>
    <script>
        // Esperamos a que el DOM esté completamente cargado
        document.addEventListener("DOMContentLoaded", function () {
            // 1. Definimos el array con las notas de los estudiantes
            const notas = [4, 7, 10, 5, 6, 3, 8];

            // 2. Creamos un título para la página
            const titulo = document.createElement("h1");
            titulo.textContent = "Análisis de Resultados de Examen";
            document.body.appendChild(titulo); // Añadimos el título al DOM

            // 3. Creamos un contenedor para los resultados
            const resultados = document.createElement("div");
            resultados.style.marginTop = "10px";
            document.body.appendChild(resultados);

            // 4. Usamos filter() para separar aprobados y suspendidos
            const aprobadas = notas.filter(nota => nota >= 5); // Notas
mayores o iguales a 5
            const suspendidas = notas.filter(nota => nota < 5); // Notas
menores a 5

            // 5. Calculamos la media de todas las notas con reduce()
            const media = notas.reduce((total, nota) => total + nota, 0) /
notas.length;

            // 6. Creamos y mostramos un párrafo con las notas aprobadas
            const aprobadasP = document.createElement("p");
            aprobadasP.textContent = `Notas aprobadas: ${aprobadas.join(",
")}`;
            resultados.appendChild(aprobadasP);
```

```
            // 7. Creamos y mostramos un párrafo con las notas suspendidas
            const suspendidasP = document.createElement("p");
            suspendidasP.textContent = `Notas suspendidas: $
{suspendidas.join(", ")}`;
            resultados.appendChild(suspendidasP);

            // 8. Creamos y mostramos un párrafo con la media de las notas
            const mediaP = document.createElement("p");
            mediaP.textContent = `Media de notas: ${media.toFixed(2)}`;
            resultados.appendChild(mediaP);

            // 9. Basándonos en la media, mostramos un mensaje final
            const mensaje = document.createElement("p");
            mensaje.textContent = media >= 6 ? "Buen rendimiento" :
"Rendimiento insuficiente";
            document.body.appendChild(mensaje);
        });
    </script>
</body>
</html>
```

Ejercicio 3: Procesador de cadenas

Descripción: Este ejercicio utiliza `split`, `map`, `filter` y `join` para procesar un texto proporcionado por el usuario. Las palabras se convierten en mayúsculas y se filtran las de más de 5 letras.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Procesador de Cadenas</title>
</head>
<body>
    <script>
        // Esperamos a que el DOM esté completamente cargado
        document.addEventListener("DOMContentLoaded", function () {
            // 1. Pedimos al usuario que introduzca una cadena de texto
            const texto = prompt("Introduce una cadena de texto:");

            // 2. Dividimos la cadena en palabras utilizando split() y
eliminamos espacios extra
            const palabras = texto.split(" ").map(p => p.trim());

            // 3. Creamos un título para el procesador de cadenas
            const titulo = document.createElement("h1");
            titulo.textContent = "Procesador de Cadenas";
            document.body.appendChild(titulo);

            // 4. Creamos un contenedor para mostrar los resultados
            const salida = document.createElement("div");
            salida.style.marginTop = "10px";
            document.body.appendChild(salida);
```

```
            // 5. Convertimos las palabras a mayúsculas con map()
            const palabrasMayusculas = palabras.map(palabra =>
palabra.toUpperCase());

            // 6. Filtramos las palabras que tienen más de 5 letras
            const palabrasLargas = palabrasMayusculas.filter(palabra =>
palabra.length > 5);

            // 7. Mostramos las palabras largas si hay resultados
            if (palabrasLargas.length > 0) {
                const resultadoP = document.createElement("p");
                resultadoP.textContent = `Palabras largas: $
{palabrasLargas.join(", ")}`;
                salida.appendChild(resultadoP);
            } else {
                // Si no hay palabras largas, mostramos un mensaje
indicándolo
                const sinResultadoP = document.createElement("p");
                sinResultadoP.textContent = "No hay palabras de más de 5
letras.";
                salida.appendChild(sinResultadoP);
            }
        });
    </script>
</body>
</html>
```

Ejercicio 4: Sistema de estadísticas de números

Descripción: Este ejercicio utiliza métodos como `sort`, `find`, `reduce` y `reverse` para analizar un array de números, mostrando estadísticas como la suma, el orden y el primer número mayor que 20.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Estadísticas de Números</title>
</head>
<body>
    <script>
        // Esperamos a que el DOM esté completamente cargado
        document.addEventListener("DOMContentLoaded", function () {
            // 1. Definimos el array de números
            const numeros = [12, 45, 2, 7, 18, 39];

            // 2. Creamos un título para la página
            const titulo = document.createElement("h1");
            titulo.textContent = "Sistema de Estadísticas de Números";
            document.body.appendChild(titulo);

            // 3. Creamos un contenedor para los resultados
            const resultados = document.createElement("div");
            resultados.style.marginTop = "10px";
```

```
            document.body.appendChild(resultados);

            // 4. Ordenamos los números de menor a mayor con sort()
            const ordenados = [...numeros].sort((a, b) => a - b);

            // 5. Calculamos la suma total de los números con reduce()
            const suma = numeros.reduce((acc, num) => acc + num, 0);

            // 6. Encontramos el primer número mayor que 20 con find()
            const mayorQue20 = numeros.find(num => num > 20);

            // 7. Encontramos el índice del primer número mayor que 20 con
findIndex()
            const indiceMayorQue20 = numeros.findIndex(num => num > 20);

            // 8. Invertimos el orden del array con reverse()
            const invertido = [...numeros].reverse();

            // 9. Creamos y mostramos los resultados en párrafos
            const ordenadosP = document.createElement("p");
            ordenadosP.textContent = `Ordenados: ${ordenados.join(", ")}`;
            resultados.appendChild(ordenadosP);

            const sumaP = document.createElement("p");
            sumaP.textContent = `Suma total: ${suma}`;
            resultados.appendChild(sumaP);

            const mayorP = document.createElement("p");
            mayorP.textContent = `Primer número > 20: ${mayorQue20}`;
            resultados.appendChild(mayorP);

            const indiceP = document.createElement("p");
            indiceP.textContent = `Índice del primer número > 20: $
{indiceMayorQue20}`;
            resultados.appendChild(indiceP);

            const invertidoP = document.createElement("p");
            invertidoP.textContent = `Invertido: ${invertido.join(", ")}`;
            resultados.appendChild(invertidoP);
        });
    </script>
</body>
</html>
```

Ejercicio 5: Juego de palabras y arrays

Descripción: Este ejercicio combina `split`, `splice`, `push` y `slice` para manipular un array de palabras con opciones interactivas para el usuario.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Juego de Palabras</title>
</head>
```

```
<body>
    <script>
        // Esperamos a que el DOM esté completamente cargado
        document.addEventListener("DOMContentLoaded", function () {
            // 1. Creamos un array inicial con palabras
            const palabras = ["gato", "perro", "pez", "loro"];

            // 2. Creamos un título para la página
            const titulo = document.createElement("h1");
            titulo.textContent = "Juego de Palabras y Arrays";
            document.body.appendChild(titulo);

            // 3. Creamos un contenedor para mostrar los resultados
            const salida = document.createElement("div");
            salida.style.marginTop = "10px";
            document.body.appendChild(salida);

            // 4. Pedimos al usuario una palabra nueva para añadir al array
            const nuevaPalabra = prompt("Introduce una nueva palabra:");
            if (!palabras.includes(nuevaPalabra)) {
                palabras.push(nuevaPalabra); // Añadimos la palabra al
array
                const nuevaPalabraP = document.createElement("p");
                nuevaPalabraP.textContent = `Nueva palabra añadida: $
{nuevaPalabra}.`;
                salida.appendChild(nuevaPalabraP);
            } else {
                const existeP = document.createElement("p");
                existeP.textContent = `La palabra "${nuevaPalabra}" ya
existe.`;
                salida.appendChild(existeP);
            }

            // 5. Mostramos las primeras tres palabras con slice()
            const primerasTres = palabras.slice(0, 3);
            const tresPalabrasP = document.createElement("p");
            tresPalabrasP.textContent = `Primeras tres palabras: $
{primerasTres.join(", ")}`;
            salida.appendChild(tresPalabrasP);

            // 6. Pedimos al usuario una palabra para reemplazar la última
palabra
            const reemplazo = prompt("Introduce una palabra para reemplazar
la última:");
            palabras.splice(palabras.length - 1, 1, reemplazo);

            // 7. Mostramos el array final
            const finalArrayP = document.createElement("p");
            finalArrayP.textContent = `Array final: ${palabras.join(", ")}
`;
            salida.appendChild(finalArrayP);
        });
    </script>
</body>
</html>
```