Métodos de arreglos

Los métodos de arreglos (arrays) en JavaScript son funciones incorporadas que permiten realizar diversas operaciones en los elementos de un arreglo. Estas operaciones pueden incluir agregar, eliminar, modificar, filtrar y transformar elementos en un arreglo. Los métodos de arreglos son una parte esencial de la programación en JavaScript y son ampliamente utilizados para manipular datos en estructuras de listas.

Los métodos de arreglos simplifican la tarea de trabajar con arreglos al proporcionar una sintaxis clara y concisa para realizar operaciones comunes. Puedes usar estos métodos para realizar tareas como buscar elementos, ordenar arreglos, mapear valores, filtrar elementos según ciertos criterios y mucho más.

Algunos de los métodos de arreglos más comunes en JavaScript son:

push():
Se utiliza para agregar uno o más elementos al final de un array. Es una de las formas más comunes de modificar un array al agregar nuevos elementos a su final. El método push() modifica el array original y devuelve la nueva longitud del array después de agregar los elementos.

sintaxis básica del método push():

```
array.push(elemento);
```

array: es el array al que deseas agregar elementos.
elemento: son los elementos que deseas agregar al final del array.

Ejemplo con código:

```
const frutas = ['manzana', 'plátano', 'pera'];
const nuevaFruta = 'naranja';

frutas.push(nuevaFruta);

console.log(frutas); // Output: ['manzana', 'plátano', 'pera', 'naranja']
```

también puedes usar push() para agregar varios elementos de una vez:

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

numeros.push(...nuevosNumeros);

console.log(numeros); // Output: [1, 2, 3, 4, 5, 6]
```

pop():
El método pop() se utiliza para eliminar el último elemento de un array y devolver ese elemento eliminado. Este método modifica el array original al quitar el último elemento y devuelve el valor del elemento eliminado. Si el array está vacío, pop() devuelve undefined.

Sintaxis básica:

```
// sintaxis
array.pop();
```

array: es el array del que deseas eliminar el último elemento.

Ejemplo con código:

```
const frutas = ['manzana', 'plátano', 'pera'];
const ultimaFruta = frutas.pop();

console.log(ultimaFruta); // Output: 'pera'
console.log(frutas); // Output: ['manzana', 'plátano']
```

Es importante mencionar que pop() modifica el array original al eliminar el elemento. Si deseas eliminar elementos de un array pero mantener una copia del array original sin modificar, puedes hacer una copia del array antes de usar pop() o almacenar el valor eliminado en una variable.

Shift():
El método shift() es una función fundamental en JavaScript que se utiliza para manipular arrays. Este método se utiliza para eliminar el primer elemento de un array y, al mismo

tiempo, modificar el array original. Devuelve el elemento eliminado, lo que permite acceder y utilizar ese valor inmediatamente o almacenarlo en una variable para su posterior procesamiento.

Ejemplo de sintaxis básica:

```
array.shift();
```

array: es el array del que deseas eliminar el primer elemento.

Ejemplo con código:

```
const frutas = ['manzana', 'plátano', 'pera'];
const primeraFruta = frutas.shift();

console.log(primeraFruta); // Output: 'manzana'
console.log(frutas); // Output: ['plátano', 'pera']
```

shift() es especialmente útil cuando necesitas trabajar con elementos de un array en un orden específico, comenzando desde el principio y avanzando gradualmente a través de la lista. Además, puede ser útil en situaciones donde necesitas procesar elementos de manera secuencial.

unshift():
El método unshift() se utiliza para agregar uno o más elementos al principio de un array. Es el opuesto de push(), que agrega elementos al final del array. unshift() modifica el array original y devuelve la nueva longitud del array después de agregar los elementos.

Sintaxis de unshift:

```
106    array.unshift(elemento);
```

array: es el array al que deseas agregar elementos al principio.

Ejemplo con código:

```
const frutas = ['manzana', 'plátano', 'pera'];
const nuevaFruta = 'naranja';

frutas.unshift(nuevaFruta);

console.log(frutas); // Output: ['naranja', 'manzana', 'plátano', 'pera']
```

también puedes usar unshift () para agregar varios elementos de una vez:

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

numeros.unshift(...nuevosNumeros);

console.log(numeros); // Output: [1, 2, 3, 4, 5, 6]
```

concat():
El método concat() en JavaScript se utiliza para combinar dos o más arrays en uno nuevo sin modificar los arrays originales. Este método crea y devuelve un nuevo array que contiene los elementos de los arrays originales en el orden en que se proporcionan como argumentos a concat(). Los arrays originales permanecen sin cambios.

Sintaxis de concat:

```
nuevoArray = array1.concat(array2, array3);
```

nuevoArray: es el nuevo array que se crea al combinar los arrays originales.

array1, array2, array3: son los arrays que deseas combinar.

Ejemplo con código:

```
const frutas1 = ['manzana', 'plátano'];
const frutas2 = ['pera', 'naranja'];
const frutasCombinadas = frutas1.concat(frutas2);

console.log(frutasCombinadas); // Output: ['manzana', 'plátano', 'pera', 'naranja']
```

Una ventaja importante de concat() es que los arrays originales no se modifican, por lo que siguen siendo los mismos después de la operación. Esto es útil si deseas mantener los arrays originales intactos mientras creas un nuevo array que los contiene a todos.

También puedes utilizar el operador spread (...) para lograr el mismo efecto de combinar arrays.

join():
El método join() se utiliza para crear una cadena de caracteres a partir de los elementos de un array. Permite combinar todos los elementos del array en una sola cadena y separarlos por un separador especificado. El array original no se modifica; en su lugar, join() devuelve una nueva cadena basada en los elementos del array.

Sintaxis de join:

```
cadena = array.join(separador);
```

cadena: es la cadena de caracteres resultante que contiene los elementos del array unidos por el separador.

array: es el array cuyos elementos se unirán para formar la cadena.

separador: es un argumento opcional que determina el carácter o cadena que se usará para separar los elementos en la cadena resultante. Si no se proporciona, se asume una coma (",") como separador por defecto.

Ejemplo con código:

```
const frutas = ['manzana', 'plátano', 'pera'];
const cadenaFrutas = frutas.join(', ');

console.log(cadenaFrutas); // Output: 'manzana, plátano, pera'
```

Ejemplo usando otros separadores:

```
const palabras = ['Hola', 'mundo', 'cómo', 'estás'];
const frase = palabras.join(' ');

console.log(frase); // Output: 'Hola mundo cómo estás'
```

Ejemplo sin separador:

```
const numeros = [1, 2, 3, 4];
const cadenaNumeros = numeros.join();

console.log(cadenaNumeros); // Output: '1,2,3,4'
```

join() es útil cuando deseas combinar los elementos de un array en una sola cadena de caracteres, especificando un separador opcional entre los elementos. Esto es útil para formatear cadenas de salida o construir expresiones legibles a partir de los valores en un array sin modificar el array original.

splice():
El método splice() se utiliza para modificar un array al agregar o eliminar elementos en una ubicación específica dentro del array. splice() permite realizar operaciones de inserción, eliminación o reemplazo de elementos en un array. Este método modifica el array original y devuelve un nuevo array con los elementos eliminados (si se han eliminado) o un array vacío si no se eliminaron elementos.

Sintaxis del método splice:

```
array.splice(indice, cantidad, elemento1, elemento2, elementoN);
```

array: es el array en el que deseas realizar las operaciones.

índice: es el índice en el que deseas comenzar a realizar las operaciones.

cantidad: es el número de elementos que deseas eliminar a partir del índice especificado.

elemento1, elemento2, elementoN: son los elementos que deseas agregar al array en la posición especificada.

Ejemplo con código:

Eliminando elementos de un array:

```
const numeros = [1, 2, 3, 4, 5];
const elementosEliminados = numeros.splice(2, 2);

console.log(elementosEliminados); // Output: [3, 4]
console.log(numeros); // Output: [1, 2, 5]
```

En el caso anterior, splice(2, 2) elimina dos elementos del array números a partir del índice 2, lo que resulta en la eliminación de los valores 3 y 4. Los elementos eliminados se almacenan en la variable elementosEliminados.

Agregando elementos a un array con splice:

```
const frutas = ['manzana', 'plátano', 'pera'];
frutas.splice(1, 0, 'naranja', 'uva');

console.log(frutas); // Output: ['manzana', 'naranja', 'uva', 'plátano', 'pera']
```

En este caso, splice(1, 0, 'naranja', 'uva') agrega los elementos 'naranja' y 'uva' al array frutas en la posición 1 (justo después de 'manzana') sin eliminar ningún elemento existente.

Reemplazando elementos en un array:

```
const colores = ['rojo', 'verde', 'azul'];
colores.splice(1, 1, 'amarillo');

console.log(colores); // Output: ['rojo', 'amarillo', 'azul']
```

slice():
El método slice() es una función incorporada en JavaScript que se utiliza para crear una copia superficial de una porción de un arreglo original sin modificar el arreglo original. Proporciona una forma flexible de extraer elementos de un arreglo y crear un nuevo arreglo basado en los índices especificados.

Sintaxis de slice:

```
array.slice([inicio[fin]])
```

array: El arreglo del cual deseas extraer una porción.

inicio (opcional): El índice desde el cual se comenzará a extraer elementos. Si se omite, se inicia desde el índice 0.

fin (opcional): El índice hasta el cual se extraerán elementos.

El método slice() devuelve un nuevo arreglo que contiene una copia superficial de los elementos extraídos del arreglo original. Esto significa que los elementos del nuevo arreglo son referencias a los mismos objetos que los del arreglo original. Si los objetos son mutables, como objetos anidados o arreglos dentro del arreglo original, los cambios realizados en ellos se reflejarán tanto en el arreglo original como en el nuevo, ya que comparten las mismas referencias.

Ejemplo con código de slice:

```
const originalArray = [1, 2, 3, 4, 5];
const slicedArray = originalArray.slice(1, 4);
// En este ejemplo, slice(1, 4) crea una copia superficial de los elementos desde el
// índice 1 (inclusive) hasta el índice 4 (exclusivo) del originalArray, resultando
// en [2, 3, 4].
console.log(slicedArray); // Resultado: [2, 3, 4]
```

map():
El método map() se utiliza para transformar un array creando un nuevo array a partir de la aplicación de una función a cada uno de los elementos del array original. Este método no modifica el array original, sino que devuelve un nuevo array con los resultados de aplicar la función especificada a cada elemento.

Sintaxis del método map:

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

const nuevoArray = arrayOriginal.map((elemento, indice) => {
  // Código para transformar el elemento
  return nuevoValor;
});
```

nuevoArray: es el nuevo array que se crea a partir de los resultados de aplicar la función a cada elemento del array original.

arrayOriginal: es el array original que se va a transformar.

elemento: El elemento actual del array.

índice: El índice del elemento actual en el array.

Ejemplo con código de map:

```
const numeros = [1, 2, 3, 4, 5];
const cuadrados = numeros.map(numero => numero * numero);

console.log(cuadrados); // Output: [1, 4, 9, 16, 25]
```

Otro ejemplo:

```
const personas = [
  { nombre: 'Juan', edad: 30 },
  { nombre: 'María', edad: 25 },
  { nombre: 'Carlos', edad: 35 }
];

// Utilizando map para extraer solo los nombres
const nombres = personas.map(persona => persona.nombre);

console.log(nombres); // Output: ['Juan', 'María', 'Carlos']
```

filter():
El método filter() en JavaScript se utiliza para crear un nuevo array a partir de un array existente, que contiene solo los elementos que cumplan con ciertos criterios especificados en una función de filtro. filter() se utiliza para filtrar elementos de un array basándose en una condición dada y crear un nuevo array con los elementos que satisfagan esa condición. El array original no se modifica en el proceso.

Sintaxis de filter:

```
const nuevoArrayFiltrado = arrayOriginal.filter((elemento, indice) => {
  // Condición para filtrar el elemento
  return true_o_false;
});
```

nuevoArray: es el nuevo array que contiene los elementos filtrados.

arrayOriginal: es el array original del que se van a filtrar los elementos.

(elemento, índice, array) => { ... }: es una función de flecha que se ejecutará para cada elemento del array original

elemento: El elemento actual del array.

índice: El índice del elemento actual en el array.

Ejemplo de uso:

```
const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9];
const pares = numeros.filter(numero => numero % 2 === 0);

console.log(pares); // Output: [2, 4, 6, 8]
```

reduce():
El método reduce() en JavaScript se utiliza para reducir un array a un solo valor aplicando una función acumuladora a cada elemento del array, de izquierda a derecha, para combinarlos en un único resultado. Este método es especialmente útil cuando se desea realizar una operación de agregación en un conjunto de datos, como sumar todos los elementos de un array o concatenarlos en una sola cadena.

Sintaxis de reduce:

```
const valorFinal = arrayOriginal.reduce((acumulador, elemento, indice) => {
  // Código para reducir el array
  return nuevoAcumulador;
}, valorInicial);
```

resultado: es el valor resultante después de aplicar la función acumuladora a todos los elementos del array.

array: es el array en el que se realiza la operación de reducción.

acumulador: El valor acumulado hasta el momento.

elemento: El elemento actual del array.

índice: El índice del elemento actual en el array.

valorInicial (opcional): es el valor inicial del acumulador. Si se proporciona, la reducción comienza con este valor. Si no se proporciona, se toma el primer elemento del array como valor inicial y la reducción comienza desde el segundo elemento.

Ejemplo de uso para reduce:

```
const numeros = [1, 2, 3, 4, 5];
const suma = numeros.reduce((acumulador, numero) => acumulador + numero, 0);

console.log(suma); // Output: 15
```

valorInicial (opcional): es el valor inicial del acumulador. Si se proporciona, la reducción comienza con este valor. Si no se proporciona, se toma el primer elemento del array como valor inicial y la reducción comienza desde el segundo elemento.

sort():
El método sort() se utiliza para ordenar los elementos de un array en su lugar y devolver el array ordenado. Por defecto, sort() ordena los elementos como cadenas de caracteres y los ordena en orden lexicográfico(diccionario). Sin embargo, puedes proporcionar una función de comparación personalizada para ordenar los elementos de acuerdo con tus propios criterios.

Sintaxis de sort:

```
frutas.sort();
```

Ejemplo de uso para sort ordenando alfabéticamente:

```
const frutas = ['manzana', 'plátano', 'naranja', 'uva'];
frutas.sort(); // Ordena las frutas alfabéticamente

console.log(frutas); // Output: ['manzana', 'naranja', 'plátano', 'uva']
```

Ejemplo de uso para sort ordenando números:

```
const numeros = [10, 5, 20, 2, 30];
numeros.sort((a, b) => a - b); // Ordena los números en orden ascendente

console.log(numeros); // Output: [2, 5, 10, 20, 30]
```

creamos una función de comparación (a, b) => a - b que compara dos números y los ordena en orden ascendente. Puedes ajustar la función de comparación para cambiar el orden (por ejemplo, (a, b) => b - a para orden descendente).

Ejemplo usando un array de objetos:

```
const personas = [
  { nombre: 'Juan', edad: 30 },
  { nombre: 'María', edad: 25 },
  { nombre: 'Carlos', edad: 35 }
];

personas.sort((a, b) => a.edad - b.edad); // Ordena las personas por edad

console.log(personas);/* Output:[{ nombre: 'María', edad: 25 },{ nombre: 'Juan', edad: 30 },{ nombre: 'Carlos', edad: 35 }]
```

indexOf():
El método indexOf() en JavaScript se utiliza para buscar la primera aparición de un elemento en un array y devuelve el índice de la posición en la que se encuentra ese elemento. Si el elemento no se encuentra en el array, indexOf() devuelve -1.

Ejemplo de sintaxis de indexOf:

```
array.indexOf(elemento);
```

Ejemplo de uso:

```
const frutas = ['manzana', 'plátano', 'naranja', 'uva'];

const indiceNaranja = frutas.indexOf('naranja');
console.log(indiceNaranja); // Output: 2 (índice de 'naranja' en el array)
```

lastIndexOf(): El método lastIndexOf() en JavaScript es similar a indexOf(), pero en lugar de buscar la primera aparición de un elemento en un array, busca la última aparición de ese elemento y devuelve el índice de la última posición en la que se encuentra ese elemento. Si el elemento no se encuentra en el array, lastIndexOf() devuelve -1.

Sintaxis de lastIndexOf:

```
array.lastIndexOf(elemento);
```

Ejemplo de uso:

```
const colores = ['rojo', 'verde', 'azul', 'verde', 'amarillo'];

const ultimoIndiceVerde = colores.lastIndexOf('verde');
console.log(ultimoIndiceVerde); // Output: 3 (índice de la última aparición de 'verde' en el array)
```

includes():
El método includes() en JavaScript se utiliza para verificar si un array contiene un elemento específico y devuelve true si lo encuentra, o false si no. A diferencia de indexOf() y lastIndexOf(), que devuelven el índice de la posición en la que se encuentra el elemento, includes() simplemente devuelve un valor booleano que indica si el elemento está presente o no en el array.

Sintaxis de includes:

```
array.includes(elemento);
```

Ejemplo de uso:

```
const frutas = ['manzana', 'plátano', 'naranja'];

const contieneNaranja = frutas.includes('naranja');
console.log(contieneNaranja); // Output: true (el array contiene 'naranja')
```

Ejemplo con array de objetos:

```
const personas = [
  { nombre: 'Juan', edad: 30 },
  { nombre: 'María', edad: 25 },
  { nombre: 'Carlos', edad: 35 }
];

const buscarPersona = { nombre: 'María', edad: 25 };
const contienePersona = personas.includes(buscarPersona);
console.log(contienePersona); // Output: true (el array contiene la persona con nombre 'María' y edad 25)
```

find():
El método find() en JavaScript se utiliza para buscar el primer elemento en un array que cumpla con una condición específica y devolver ese elemento. Si ningún elemento cumple con la condición, find() devuelve undefined. find() es útil cuando deseas encontrar un elemento en un array que cumpla con ciertos criterios personalizados.

Sintaxis del método find():

```
array.find((elemento, indice) => {
  // Condición para encontrar el elemento
  return true_o_false;
});
```

Ejemplo de uso:

```
const personas = [
  { nombre: 'Juan', edad: 30 },
  { nombre: 'María', edad: 25 },
  { nombre: 'Carlos', edad: 35 }
];

const personaEncontrada = personas.find(persona => persona.edad === 25);
console.log(personaEncontrada);// Output: { nombre: 'María', edad: 25 }
```

Es importante destacar que find() detiene la búsqueda tan pronto como encuentra el primer elemento que cumple con la condición y no continúa buscando otros elementos. Si necesitas encontrar todos los elementos que cumplan con una condición, puedes usar el método filter().

reverse():
El método reverse() en JavaScript se utiliza para invertir el orden de los elementos en un array en su lugar, es decir, modifica el array original y lo reorganiza de manera inversa. No devuelve un nuevo array con los elementos invertidos, sino que modifica el array existente.

Sintaxis de reverse:

```
array.reverse();
```

Ejemplo de uso:

```
const frutas = ['manzana', 'plátano', 'naranja', 'uva'];

frutas.reverse();
console.log(frutas);// Output: ['uva', 'naranja', 'plátano', 'manzana']
```

Es importante tener en cuenta que reverse() no crea un nuevo array con los elementos invertidos, sino que modifica el array existente. Si necesitas un nuevo array con los

elementos invertidos sin modificar el original, puedes hacerlo creando una copia del array antes de aplicar reverse().

Ahora que ya conocemos los métodos de arreglos vamos a manipular el dom agregando y eliminando elementos.