Algoritmos: 6

Arreglos unidimensionales

Un **arreglo unidimensional** en JavaScript es una estructura de datos que permite almacenar una colección de elementos en una sola dimensión, es decir, en una lista lineal. Cada elemento del arreglo tiene una posición específica, denominada *índice*, que comienza en 0 para el primer elemento y aumenta en uno para cada elemento siguiente

```
let frutas = ["manzana", "banano", "pera"];
```

Arreglos bidimensionales o de 2 dimensiones

Un **arreglo bidimensional** en JavaScript es una estructura de datos donde cada elemento se accede mediante dos índices: el primero para el arreglo interior y el segundo para el elemento de ese arreglo interior.

```
let matriz = [[1, 2, 3],[4, 5, 6],[7, 8, 9]];
```

Arreglos de dimensión superior (3 o más)

Para acceder a los elementos de dimensión superior, se usan tantos índices como tenga la dimensión, por ejemplo, si el arreglo es de dimensión 3 se usan tres índices, si es de dimensión 4 se usan 4 índices.

```
let arregloDimension3 = [ [[1, 5, 8], [true]], [[true, false, true],  [true]] ];
console.log(arregloDimension3[0][1][0]);//imprime: true
console.log(arregloDimension3[1][0][1]);//imprime: false
```

Escritura y eliminación de elementos en arreglos unidimensional:

Para escribir o eliminar elementos en un arreglo unidimensional, primero se accede al elemento que se quiere escribir o eliminar mediante su índice, luego usamos el operador de asignación para escribir. Usamos splice para eliminar el elemento.

```
// Creación del arreglo unidimensional
let arregloUnidimensional = [1, 5, 8, true, false, true, true];


// ESCRITURA
arregloUnidimensional[2] = 10; // Cambiamos el valor en el índice 2
console.log(arregloUnidimensional[2]); // Imprime: 10


// ELIMINACIÓN
arregloUnidimensional.splice(1, 1); // Eliminamos el elemento en el índice 1
console.log(arregloUnidimensional); // Imprime: [1, 10, true, false, true, true]
```

Obtención del tamaño de un arreglo

para obtener el tamaño (o la cantidad de elementos) de un arreglo, se usa la propiedad .length. Esta propiedad devuelve un número que representa la cantidad de elementos en el arreglo.

```
const numeros = [1, 2, 3, 4, 5];
const tamaño = numeros.length; console.log(tamaño); // Imprime: 5
```

Recorrido de arreglos unidimensionales

for: El bucle for permite recorrer el arreglo accediendo a cada elemento por su índice.

i representa el índice actual en cada iteración, que va desde 0 hasta array.length - 1.

array[i] accede al valor en cada posición del arreglo.

```
const array = [42, "Hola", true, "Somos", false, 99, "Programadores", 3.14, "Vankversity",
    true];

for (let i = 0; i < array.length; i++) {
    console.log(array[i]); // Muestra cada elemento del arreglo
  }
```

Recorrido de arreglos unidimensionales

forEach: El método forEach ejecuta una función para cada elemento del arreglo. Es más directo y limpio que el for, pero no permite detener el bucle como for (no admite break ni continue).

forEach es más legible y práctico para recorrer elementos, pero no permite control total del flujo como for.

```
let array = [42, "Hola", true, "Somos", false, 99, "Programadores", 3.14, "Vankversity",
    true];

array.forEach((element, index) => {
  console.log(`Índice: ${index}, Elemento: ${element}, Tipo: ${typeof element}`);
});
```

Escritura y eliminación de elementos en arreglos multidimensionales:

Para escribir o eliminar elementos en un arreglo multidimensional, primero se accede al elemento que se quiere escribir o eliminar(tal cual se explica en la primera parte de esta guía), luego usamos el operador de asignación para escribir o usamos splice para eliminar el elemento.

```
// DECLARACION
let arregloDimension3 = [ [[1, 5, 8], [true]], [[true, false, true],  [true]] ];

//ESCRITURA
arregloDimension3[1][0][2] = false;
console.log(arregloDimension3[1][0][2])//imprime: false

//ELIMINACIÓN
arregloDimension3[0][0].splice(1, 1);
console.log(arregloDimension3);//imprime: [ [ [ 1, 8 ], [ true ] ], [ [ true, false, false //], [ true ] ] ]
```

Recorrido de arreglos multidimensionales

for: Para recorrer un arreglo multidimensional, por ejemplo uno bidimensional con ciclo for, se usan dos ciclos anidados y dos índices, i y j, de tal manera que el ciclo for externo recorra los elementos más externos, los cuales son arreglos, y el ciclo for interno recorra los elementos más internos, los cuales no son arreglos.

```
let arregloBidimensional = [[20, 80, 30], [10, 20, 50],[70, 60, 40]];
//El siguiente ciclo recorre los elementos que conforman el arreglo
//arregloBidimensional es decir, [20, 80, 30], [10, 20, 50],[70, 60, 40]
for (let i = 0; i < arregloBidimensional.length; i++) {
 //El siguiente ciclo recorre los elementos de cada elemento del arreglo
 //arregloBidimensional, es decir 20, 80, 30, 10, 20, 50 , 70, 60, 40
 for (let j = 0; j < arregloBidimensional[i].length; j++) {
   //Acá se imprime cada elemento más interno del arreglo
   console.log(arregloBidimensional[i][j]);
 } //La salida seria: 20, 80, 30, 10, 20, 50, 70, 60, 40
}
```

Recorrido de arreglos multidimensionales

forEach: Para recorrer un arreglo multidimensional, por ejemplo uno bidimensional con ciclo forEach, se usan dos ciclos anidados, de tal manera que el ciclo forEach externo recorra los elementos más externos, los cuales son arreglos, y el ciclo forEach interno recorra los elementos más internos, los cuales no son arreglos.

```
let arregloBidimensional = [[20, 80, 30], [10, 20, 50],[70, 60, 40]];
//El siguiente ciclo recorre los elementos que conforman el arreglo
//arregloBidimensional es decir, [20, 80, 30], [10, 20, 50],[70, 60, 40]
arregloBidimensional.forEach(i => {
 //El siguiente ciclo recorre los elementos de cada elemento del arreglo
 //arregloBidimensional, es decir 20, 80, 30, 10, 20, 50 , 70, 60, 40
 i.forEach(j => {
   //Acá se imprime cada elemento más interno del arreglo
   console.log(j);
 })// salida: 20, 80, 30, 10, 20, 50, 70, 60, 40
})
```

Métodos de arreglos en javascript

La clase Array en JavaScript incluye numerosos métodos que permiten manipular, transformar y recorrer arreglos de forma flexible. Aquí está un resumen de los métodos más útiles, organizados por tipo de operación

Métodos de manipulación de elementos

push(elemento): Agrega uno o más elementos al final del arreglo y devuelve la nueva longitud.

```
let frutas = ["manzana", "pera"];
let nuevaLongitud = frutas.push("naranja", "plátano");
console.log(frutas); // ["manzana", "pera", "naranja", "plátano"]
console.log(nuevaLongitud); // 4
```

Métodos de manipulación de elementos

pop(): Elimina el último elemento del arreglo y lo devuelve.

```
let frutas = ["manzana", "pera", "naranja"];
let ultimaFruta = frutas.pop();
console.log(frutas); // ["manzana", "pera"]
console.log(ultimaFruta); // "naranja"
```

Métodos de manipulación de elementos

shift(): Elimina el primer elemento del arreglo y lo devuelve.

```
let frutas = ["manzana", "pera", "naranja"];
let primeraFruta = frutas.shift();
console.log(frutas); // ["pera", "naranja"]
console.log(primeraFruta); // "manzana"
```

Métodos de manipulación de elementos

unshift(elemento): Agrega uno o más elementos al inicio del arreglo y devuelve la nueva longitud.

```
let frutas = ["pera", "naranja"];
let nuevaLongitud = frutas.unshift("manzana", "plátano");
console.log(frutas); // ["manzana", "plátano", "pera", "naranja"]
console.log(nuevaLongitud); // 4
```

Métodos de manipulación de elementos

splice(inicio, cantidad, elemento1, elemento2, ...): Agrega, elimina o reemplaza elementos a partir de un índice específico.

```
let frutas = ["manzana", "pera", "naranja", "plátano"];

// Eliminar 1 elemento en el índice 1 (quita "pera")
let eliminados = frutas.splice(1, 1);

console.log(frutas); // ["manzana", "naranja", "plátano"]
console.log(eliminados); // ["pera"]

// Agregar 2 elementos en el índice 2
frutas.splice(2, 0, "mango", "uva");

console.log(frutas); // ["manzana", "naranja", "mango", "uva", "plátano"]

// Reemplazar 1 elemento en el índice 0
frutas.splice(0, 1, "kiwi");
console.log(frutas); // ["kiwi", "naranja", "mango", "uva", "plátano"]
```

Métodos de manipulación de elementos

forEach(): permite ejecutar una función para cada elemento de un arreglo. Es útil para realizar operaciones o tareas sin la necesidad de devolver un nuevo arreglo, como haría map().

```
let nombres = ["Ana", "Luis", "Carlos", "María"];

nombres.forEach(nombre => {
    console.log(`Hola, ${nombre}!`);
});
//Salida:
//Hola, Ana!
//Hola, Luis!
//Hola, Carlos!
//Hola, María!
```

Métodos de manipulación de elementos

map(): crea un nuevo arreglo aplicando una función a cada elemento del arreglo original. Es útil cuando quieres transformar o modificar elementos de un arreglo y obtener un nuevo arreglo con los valores transformados, sin cambiar el original.

```
let palabras = ["sol", "montaña", "mar", "playa", "cielo"];

let palabrasLargas = palabras.filter(palabra => palabra.length > 4);

console.log(palabrasLargas); // ["montaña", "playa"]
```

Métodos de manipulación de elementos

reduce():aplica una función a cada elemento del arreglo (de izquierda a derecha) para reducir el arreglo a un solo valor. Este método es muy útil para acumular o combinar valores de un arreglo en uno solo, como sumarlos o concatenarlos.

```
let numeros = [1, 2, 3, 4, 5];

let sumaTotal = numeros.reduce((acumulador, numero) => acumulador + numero, 0);

console.log(sumaTotal); // 15
```

Métodos de manipulación de elementos

find():se utiliza para buscar el primer elemento de un arreglo que cumple con una condición específica, definida en una función de callback. Si encuentra un elemento que cumple la condición, lo devuelve; si no, devuelve undefined.

```
let numeros = [5, 12, 8, 130, 44];

let primerMayorQueDiez = numeros.find(num => num > 10);

console.log(primerMayorQueDiez); // 12
```

Métodos de manipulación de elementos

findIndex():se utiliza para encontrar el índice del primer elemento de un arreglo que cumple con una condición especificada en una función de callback. Si se encuentra un elemento que cumple con la condición, devuelve su índice; si no, devuelve -1.

```
let numeros = [5, 12, 8, 130, 44];

let indice = numeros.findIndex(num => num > 10);

console.log(indice); // 1
```

Métodos de manipulación de elementos

includes():se utiliza para determinar si un arreglo incluye un elemento específico, devolviendo true si lo contiene y false si no. Este método es sensible a mayúsculas y minúsculas cuando se trabaja con cadenas.

```
let frutas = ["manzana", "pera", "naranja", "plátano"];

let tieneManzana = frutas.includes("manzana");
let tieneUva = frutas.includes("uva");

console.log(tieneManzana); // true
console.log(tieneUva);     // false
```

Métodos de manipulación de elementos

join():se utiliza para unir todos los elementos de un arreglo en una cadena, separando los elementos con un delimitador especificado. Si no se proporciona un delimitador, el método utilizará una coma (,) por defecto.

```
let frutas = ["manzana", "pera", "naranja", "plátano"];

let cadenaFrutas = frutas.join(", ");

console.log(cadenaFrutas); // "manzana, pera, naranja, plátano"
```

Métodos de manipulación de elementos

sort(): se utiliza para ordenar los elementos de un arreglo en su lugar y devuelve el arreglo ordenado. Por defecto, el método ordena los elementos como cadenas (en orden alfabético), por lo que es importante proporcionar una función de comparación si se desea ordenar números o utilizar un orden específico.

```
let numeros = [5, 3, 8, 1, 2];

numeros.sort((a, b) => a - b); // Orden ascendente

console.log(numeros); // [1, 2, 3, 5, 8]
```

Métodos de manipulación de elementos

reverse(): se utiliza para invertir el orden de los elementos de un arreglo en su lugar. Este método modifica el arreglo original y devuelve una referencia al mismo.

```
let numeros = [1, 2, 3, 4, 5];

numeros.reverse();

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

```
let letras = ["a", "b", "c", "d", "e"];

letras.reverse();

console.log(letras); // ["e", "d", "c", "b", "a"]
```

Algoritmos: 6

VankVersity