Más métodos de arrays

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1 | slice()

.slice()

Este método devuelve (extrae) una copia de una parte del array dentro de un array (subarray). Determina el índice inicial y el final (opcional). El final no está incluido para el nuevo array.

- Si el inicio es un valor negativo, extrae los últimos elementos del array desplazándose desde el final del mismo. Si es omitido, por defecto es 0. Si es mayor a *array.length*, devolverá array vacío.
- Si el final es negativo, realiza un desplazamiento al final. Por ejemplo, *array.slice(3, -1)* extrae desde el cuarto elemento hasta el penúltimo. Si es mayor a *array.length* o si es omitido, extrae hasta el final del array,

```
{}  array.slice(inicio, fin);
       // indicamos los índices de inicio y fin
       // para obtener el nuevo array
```

{código}

```
let numeros = [3, 4, 5, 6, 7];
let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

```
let numeros = [3, 4, 5, 6, 7];
let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

Declaramos la variable **numeros** y almacenamos un array con cinco números.

```
let numeros = [3, 4, 5, 6, 7];
let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

A **slice()** le definimos el índice inicial y el final, el elemento del índice final no estará en el nuevo **subArray**.

La variable **subArray** contendrá el nuevo array creado a partir del array original numeros.

```
let numeros = [3, 4, 5, 6, 7];
let subArray = numeros.slice(0, 3);
```

```
console.log(subArray); // [3,4,5]
```

Mostramos por consola la variable **subArray** que contiene un nuevo array con una copia de los elementos del array original, pero con los índices acotados.

2 | splice()

.splice()

Este método nos sirve para remover y/o agregar elementos de un array.
Recibe tres parámetros:

- **inicio**: el índice del primer elemento (de donde comenzará el cambio).
- **cant** (opcional): un entero que indica la cantidad de elementos a eliminar. Si se omite o si es mayor que (*array.length - inicio*), se eliminarán todos los elementos desde el **inicio**.
- **items** (opcional): indica los elementos que se agregarán en el array, desde inicio. Si se omite splice, solo elimina.

```
{}
array.splice(inicio, cant, item1, item2, ...);
   // indicamos el índice, la cant y los elementos.
```

{código}

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Declaramos la variable **numeros** y almacenamos un array con cinco números.

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Al método **splice()** le definimos el índice inicial y la cantidad de elementos a remover y él/los elementos a agregar.
En este caso comienza del índice 0, elimina 0 elementos, y agrega en la posición 0 el valor 2.

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Mostramos por consola el array original **numeros** y observamos que se agregó en el índice 0 el elemento con valor 2.

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

En este caso comienza en el índice 1 y únicamente elimina 2 elementos.

```
let numeros = [3, 4, 5, 6, 7];
numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Mostramos por consola el array original **numeros** y observamos que fueron eliminados los elementos [3, 4] del array.

3 | sort()

.sort()

Este método nos sirve para ordenar los elementos de un array.
Recibe un callback como parámetro (opcional) que especifica el modo de ordenamiento. Si es omitido, el array es ordenado con el valor de string (Unicode).
Convierte a string a cada elemento.

La función recibe dos parámetros, que son los elementos a comparar, el primero vs. el segundo elemento.

```
{}
array.sort(); //ordenamiento con la posición de valor Unicode

array.sort(callback); // la función como parámetro
```

{código}

```
let marcas = [‘samsung’, ‘xiaomi’, ‘apple’];
marcas.sort();
console.log(marcas);
// [‘apple’, ‘samsung’ , ‘xiaomi’]

let marcas2 = [‘samsung’, ‘Xiaomi’, ‘apple’, ‘3M’];
marcas2.sort();
console.log(marcas);
// [‘3M’, ‘Xiaomi’, ‘apple’, ‘samsung’]
```

```
let marcas = [‘samsung’, ‘xiaomi’, ‘apple’];
marcas.sort();
console.log(marcas);
// [‘apple’, ‘samsung’ , ‘xiaomi’]
```

Declaramos un array de marcas, ordenamos con **.sort()**, luego mostramos por consola y observamos como ordenó el array.

```
let marcas2 = [‘samsung’, ‘Xiaomi’, ‘apple’,
‘3M’];
marcas2.sort();
console.log(marcas);
// [‘3M’, ‘Xiaomi’, ‘apple’, ‘samsung’]
```

```
let marcas = [‘samsung’, ‘xiaomi’,
‘apple’];
marcas.sort();
console.log(marcas);
// [‘apple’, ‘samsung’ , ‘xiaomi’]
```

```
let marcas2 = [‘samsung’, ‘Xiaomi’,
‘apple’, ‘3M’];
marcas2.sort();
console.log(marcas);
// [‘3M’, ‘Xiaomi’, ‘apple’, ‘samsung’]
```

En este caso declaramos un array de marcas, pero una de ellas contiene un número y otra comienza con mayúscula. Ordenamos con **.sort()**, luego mostramos por consola y podemos observar cómo el método ordena el array atendiendo a la posición del valor Unicode.

```
let numeros = [10, 3, 4, 52, 6, 7];
numeros.sort();
console.log(numeros);
// [10, 4, 52, 7, 8, 9]
function compare(a,b){
        return a-b;
    }
    numeros.sort(compare);
    console.log(numeros);
    // [4, 7, 8, 9, 10, 52]
```

Declaramos un array de números y al ordenarlo de forma predeterminada observamos que 9 es mayor que 10 porque “9” > “1”, lo que nos muestra un resultado incorrecto.

```
let numeros = [10, 3, 4, 52, 6, 7];
numeros.sort();
console.log(numeros);
// [10, 4, 52, 7, 8, 9]
function compare(a,b){
  return a-b;
}
numeros.sort(compare);
console.log(numeros);
// [4, 7, 8, 9, 10, 52]
```

Podemos solucionar esto declarando una simple función que define un orden alternativo al predeterminado Unicode. Esta, en este caso, retorna un valor (negativo, cero o positivo).

4 | find()

.find()

Este método devuelve el valor del **primer** elemento de un array que cumple con una función especificada (callback).
Recibe un callback que se ejecuta sobre cada índice del array hasta que encuentre uno que devuelve un valor verdadero, y toma tres parámetros:

- **elemento**: se utiliza un alias para representar el valor del elemento actual que está procesando el array.
- **index** (opcional): posición del elemento actual que se está procesando en el array.
- **array**: que está siendo recorrido.

{}

```
array.find(callback(e, array));
   // indicamos el elemento a buscar
   // y la posición en la que comenzará
```

{código}

```
let criptos = [
    {nombre: 'Bitcoin', simbolo: 'BTC'},
    {nombre: 'Ethereum', simbolo: 'ETH'},
    {nombre: 'Cardano', simbolo: 'ADA'}
];
```

Definimos un array de objetos.

```
function esBitcoin(criptos) {
    return criptos.nombre === 'Bitcoin';
}

console.log(criptos.find(esBitcoin));
// {nombre: 'Bitcoin', simbolo: 'BTC'}
```

```
let criptos = [
    {nombre: 'Bitcoin', simbolo: 'BTC'},
    {nombre: 'Ethereum', simbolo: 'ETH'},
    {nombre: 'Cardano', simbolo: 'ADA'}
];
```

```
function esBitcoin(criptos) {
    return criptos.nombre === 'Bitcoin';
}
```

Definimos una función para buscar coincidencia de una propiedad de los objetos.

```
console.log(criptos.find(esBitcoin));
// {nombre: 'Bitcoin', simbolo: 'BTC'}
```

```
let criptos = [
    {nombre: 'Bitcoin', simbolo: 'BTC'},
    {nombre: 'Ethereum', simbolo: 'ETH'},
    {nombre: 'Cardano', simbolo: 'ADA'}
];

function esBitcoin(criptos) {
    return criptos.nombre === 'Bitcoin';
}

console.log(criptos.find(esBitcoin));
// {nombre: 'Bitcoin', simbolo: 'BTC'}
```

Mostramos por consola, utilizando el método **find()**, y pasamos como parámetro la función **esBitcoin**. Nos devuelve el primer objeto.

```
let criptos = [
    {nombre: 'Bitcoin', simbolo: 'BTC'},
    {nombre: 'Ethereum', simbolo: 'ETH'},
    {nombre: 'Cardano', simbolo: 'ADA'}
];
```

```
Let res = criptos.find(e => e.nombre === 'Bitcoin');
```

En este caso utilizamos arrow functions, donde el valor elemento actual sería el objeto en cuestión.

```
console.log(res);
// {nombre: 'Bitcoin', simbolo: 'BTC'}
```

```
let criptos = [
    {nombre: 'Bitcoin', simbolo: 'BTC'},
    {nombre: 'Ethereum', simbolo: 'ETH'},
    {nombre: 'Cardano', simbolo: 'ADA'}
];

Let res = criptos.find(e => e.nombre === 'Bitcoin');
```

```
console.log(res);
// {nombre: 'Bitcoin', simbolo: 'BTC'}
```

Comprobamos que la salida por consola del resultado es la misma.


DigitalHouse>
Coding School