JavaScript
Estructuras de datos

CertiDevs

Índice de contenidos

JavaScript proporciona varias **estructuras de datos** para almacenar y organizar la información en un programa.

Las estructuras de datos más comunes en JavaScript incluyen *arrays*, *objetos*, *conjuntos* (Sets) y *mapas* (Maps).

1. Arrays

Los **arrays** son estructuras de datos lineales que almacenan una colección de elementos ordenados.

Se pueden acceder a los elementos por **índice** y pueden contener elementos de diferentes tipos de datos.

1.1. Creación de Arrays

Puedes crear **arrays** utilizando corchetes ([]) y separando los elementos con comas.

```
const numeros = [1, 2, 3, 4, 5];
const frutas = ['manzana', 'plátano', 'cereza'];
const mixto = [1, 'dos', true, { nombre: 'Juan' }];
```

También puedes utilizar el constructor Array para crear arrays.

```
const numeros = new Array(1, 2, 3, 4, 5);
```

1.2. Acceso y Modificación de Elementos

Accede a los elementos del array utilizando el índice entre corchetes.

```
const frutas = ['manzana', 'plátano', 'cereza'];
console.log(frutas[0]); // 'manzana'
```

Modifica los elementos del array asignando un nuevo valor al índice.

```
const frutas = ['manzana', 'plátano', 'cereza'];
frutas[1] = 'uva';
console.log(frutas); // ['manzana', 'uva', 'cereza']
```

1.3. Métodos de Array

Los **arrays** en JavaScript tienen varios métodos útiles para realizar operaciones comunes, como agregar o eliminar elementos, iterar sobre elementos y manipular el array.

- `push` y `pop`: agregar y eliminar elementos al final del array.
- `unshift` y `shift`: agregar y eliminar elementos al principio del array.
- `splice`: agregar, eliminar o reemplazar elementos en cualquier posición del array.
- `slice`: crear una copia superficial de una parte del array.
- `concat`: combinar dos o más arrays.
- `join`: convertir un array en una cadena, uniendo sus elementos con un separador.
- `indexOf` y `lastIndexOf`: buscar la primera y última aparición de un elemento en el array.
- `includes`: verificar si un array contiene un elemento específico.
- `reverse`: invertir el orden de los elementos en el array.
- `sort`: ordenar los elementos de un array de acuerdo con una función de comparación.
- `map`, `filter`, `reduce`, `forEach`, `some`, y `every`: manipular, iterar y realizar comprobaciones en los elementos del array.

1.3.1. Métodos para agregar y eliminar elementos

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

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

const nuevaLongitud = frutas.push('uva', 'pera');
console.log(frutas); // ['manzana', 'plátano', 'cereza', 'uva', 'pera']
console.log(nuevaLongitud); // 5
```

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

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

const elementoEliminado = frutas.pop();
console.log(frutas); // ['manzana', 'plátano']
console.log(elementoEliminado); // 'cereza'
```

`splice()`: cambia el contenido de una matriz eliminando elementos existentes y/o agregando nuevos elementos.

```
const array = [2, 5, 9];

console.log(array);

const index = array.indexOf(5);
if (index > -1) { // only splice array when item is found
  array.splice(index, 1); // 2nd parameter means remove one item only
}
```

```
// array = [2, 9]
console.log(array);
```

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

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

```
const frutas = ['manzana', 'plátano', 'cereza'];
const elementoEliminado = frutas.shift();
console.log(frutas); // ['plátano', 'cereza']
console.log(elementoEliminado); // 'manzana'
```

1.3.2. Métodos para iterar y manipular elementos

`forEach()`: Ejecuta una función para cada elemento del array.

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

numeros.forEach(function (numero) {
    console.log(numero * 2);
});
// Salida:
// 2
// 4
// 6
// 8
// 10
```

`map`: Crea un nuevo array con los resultados de aplicar una función a todos los elementos del array original.

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

const cuadrados = numeros.map(function (numero) {
    return numero * numero;
});
console.log(cuadrados); // [1, 4, 9, 16, 25]
```

`filter`: Crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

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

const pares = numeros.filter(function (numero) {
    return numero % 2 === 0;
});
```

```
console.log(pares); // [2, 4]
```

`reduce`: Aplica una función acumuladora sobre un array, reduciendo sus elementos a un único valor.

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

const suma = numeros.reduce(function (acumulador, numero) {
    return acumulador + numero;
}, 0);
console.log(suma); // 15
```

`some`: Comprueba si al menos un elemento del array cumple con la condición implementada por la función dada. Retorna true si al menos un elemento cumple la condición, de lo contrario retorna false.

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

const hayImpares = numeros.some(function (numero) {
    return numero % 2 !== 0;
});
console.log(hayImpares); // true
```

`every`: Comprueba si todos los elementos del array cumplen con la condición implementada por la función dada. Retorna true si todos los elementos cumplen la condición, de lo contrario retorna false.

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

const todosPositivos = numeros.every(function (numero) {
    return numero > 0;
});
console.log(todosPositivos); // true
```

`find`: Devuelve el primer elemento del array que cumple con la condición implementada por la función dada. Si ningún elemento cumple la condición, devuelve undefined.

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

const primerPar = numeros.find(function (numero) {
    return numero % 2 === 0;
});
console.log(primerPar); // 2
```

2. Conjuntos (Set)

Un **Set** o **conjunto** es una colección de valores únicos, es decir, no permite duplicados.

Los Sets son útiles cuando necesitas almacenar un conjunto de elementos sin repetición o cuando quieres realizar operaciones como la unión, intersección y diferencia entre conjuntos.

2.1. Creación de Sets

Puedes crear un Set utilizando el constructor Set y pasando un iterable (como un array) como argumento.

```
const conjuntoNumeros = new Set([1, 2, 3, 4, 5]);
const conjuntoFrutas = new Set(['manzana', 'plátano', 'cereza']);
```

También se puede crear un Set vacío y agregar elementos utilizando el método add. El método add agrega un elemento al Set. Si el elemento ya existe en el Set, no se agregará nuevamente.

```
const conjuntoNumeros = new Set();
conjuntoNumeros.add(1);
conjuntoNumeros.add(2);
conjuntoNumeros.add(3);
```

2.2. Métodos y Propiedades de Set

`size`: Retorna la cantidad de elementos en el Set.

```
const conjuntoNumeros = new Set([1, 2, 3, 4, 5]);
console.log(conjuntoNumeros.size); // 5
```

`has`: Comprueba si un elemento existe en el Set. Retorna true si el elemento se encuentra en el Set, de lo contrario retorna false.

```
const conjuntoFrutas = new Set(['manzana', 'plátano', 'cereza']);
console.log(conjuntoFrutas.has('manzana')); // true
```

`delete`: Elimina un elemento del Set. Retorna true si el elemento se eliminó correctamente, de lo contrario retorna false.

```
const conjuntoFrutas = new Set(['manzana', 'plátano', 'cereza']);
conjuntoFrutas.delete('manzana');
console.log(conjuntoFrutas); // Set { 'plátano', 'cereza' }
```

`clear`: Elimina todos los elementos del Set.

```
const conjuntoFrutas = new Set(['manzana', 'plátano', 'cereza']);
conjuntoFrutas.clear();
console.log(conjuntoFrutas); // Set {}
```

3. Mapas (Map)

Un **mapa** o Map es una colección de pares clave-valor donde las claves pueden ser de cualquier tipo, no solo cadenas de texto como en los objetos.

3.1. Creación de Maps

Puedes crear un Map utilizando el constructor Map y pasando un iterable (como un array de pares clave-valor) como argumento.

```
const mapa = new Map([
    ['nombre', 'Juan'],
    ['edad', 28],
    ['ocupacion', 'Desarrollador']
]);
```

También puedes crear un Map vacío y agregar elementos utilizando el método set.

```
const mapa = new Map();
mapa.set('nombre', 'Juan');
mapa.set('edad', 28);
mapa.set('ocupacion', 'Desarrollador');
```

`size`: Retorna la cantidad de elementos en el Map.

```
const mapa = new Map([
    ['nombre', 'Juan'],
    ['edad', 28],
    ['ocupacion', 'Desarrollador']
]);
console.log(mapa.size); // 3
```

`get`: Obtiene el valor asociado a una clave en el Map. Retorna undefined si la clave no existe en el Map.

```
const mapa = new Map([
    ['nombre', 'Juan'],
    ['edad', 28],
```

```
    ['ocupacion', 'Desarrollador']
]);
console.log(mapa.get('nombre')); // 'Juan'
```

`forEach`: Ejecuta una función para cada par clave-valor del Map.

```
const mapa = new Map([
    ['nombre', 'Juan'],
    ['edad', 28],
    ['ocupacion', 'Desarrollador']
]);
mapa.forEach(function (valor, clave) {
    console.log(clave + ': ' + valor);
});
// Salida:
// 'nombre: Juan'
// 'edad: 28'
// 'ocupacion: Desarrollador'
```