Profesor: Arle Morales Ortiz

Guía teórico practica de arrays – map - set

Arrays

Los objetos te permiten almacenar colecciones de datos a través de nombres. Pero a menudo necesitamos una colección ordenada, donde tenemos un 1ro, un 2do, un 3er elemento y así sucesivamente. Por ejemplo, necesitamos almacenar una lista de algo: usuarios, bienes, elementos HTML, etc.

un Array comienza y termina con corchetes [ ]

Crea un Array con tres elementos : 'Apple', 'Orange', y 'Plum'.

```
let fruits = ["Apple", "Orange", "Plum"];

console.log(fruits[1]);

fruits[0] = 'pear';

console.log(fruits);
```

Las comas para separación de elementos y es ignorado las comas finales.

Para leer un elemento Array, ponemos un índice entre corchetes (los índices comienzan en cero)

Para cambiar un elemento de Array, asignamos a un Array con un índice:

length - push

```
let fruits = ["Apple", "Orange", "Plum"];
console.log(fruits.length); // 3

fruits[fruits.length] = "grape";
console.log(fruits);// [ 'Apple', 'Orange', 'Plum', 'grape' ]

fruits.push("banana");
console.log(fruits); // [ 'Apple', 'Orange', 'Plum', 'grape', 'banana' ]

fruits.length = 1;
console.log(fruits);// [ 'Apple' ]
```

Cada Array tiene una propiedad .length que se puede usar tanto para leer como para cambiar (!) el número de elementos en un Array.

Si escribimos en el Array en el índice de la longitud, agregamos un elemento:

Otra forma de agregar un elemento es a través del método Array .push()

Podando el Array se eliminan elementos

array.at() - vaciar array

```
let fruits = ["Apple", "Orange", "Plum", "banana"];
console.log( fruits.at( index: -2) ); // Plum

fruits = [];
console.log(fruits); //[]
```

El método Array .at()devuelve el elemento en un índice dado. Admite índices positivos y negativos

Para borrar (vaciar) una matriz, podemos establecerla .length en cero o poner el array vacio.

Profesor: Arle Morales Ortiz

Dentro de un array, un elemento de propagación o difusion consta de tres puntos (...) seguidos de una expresión.

```
people = ["Arle", "Juliana", "valentina", "Ana"];
people2 = ["Diego", "Juan"];

copy = [...people, ...people2,"Pepe"];
console.log(copy);
copy2 = [...people2];
console.log(copy2);

```

copia

for-of: iterando sobre índices

```
people = ["Arle", "Juliana", "valentina", "Ana"];
for (const element of people) {
    console.log(element);
}

people2 = ["Jhon", "Lian"];
for (const element of people2.keys()) {
    console.log(element);
}

for (const [i, e] of people2.entries()) {
    console.log(i, e);
}
```

for-of: iterando sobre [índice, elemento] pares

```
const cities = ["Armenia", "pereira", "Cali"];
console.log(cities.shift()); // Armenia
console.log(cities); // [ 'pereira', 'Cali' ]

const cities2 = ["Manizales", "Popayan", "Medellin"];
console.log(cities2.pop()); // Medellin

const cities3 = [...cities, ...cities2];
console.log(cities3); //[ 'pereira', 'Cali', 'Manizales', 'Popayan' ]

cities3.splice( start: 1, deleteCount: 2);
console.log(cities3);//[ 'pereira', 'Popayan' ]
```

Remueve el primero elemento

Remueve el ultimo elemento

Remueve varios elementos

Profesor: Arle Morales Ortiz

La programación funcional en JavaScript se enfoca en la manipulación de los datos sin cambiar el estado original de los mismos

```
//Map
//El método map() devuelve un nuevo array con los resultados de la llamada a una función
// proporcionada para cada elemento del array original.
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(num => num * num);
console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

1

```
//Filter:
//El método filter() crea un nuevo array con todos los elementos que pasan
// la prueba implementada por la función proporcionada.

const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

2

```
//Reduce:
//El método reduce() ejecuta una función reductora sobre cada elemento del array,
// devolviendo un único valor reducido.

const numbers = [1, 2, 3, 4, 5];
const sumOfNumbers = numbers.reduce((acc : number , curr : number ) => acc + curr);
console.log(sumOfNumbers); // 15
```

3

```
//forEach
//El método forEach() ejecuta una función para cada elemento del array.

const numbers = [1, 2, 3, 4, 5];
numbers.forEach(num => console.log(num * 2)); // Output: 2 4 6 8 10
```

4

```
//Every:
//El método every() comprueba si todos los elementos del array pasan la prueba
// implementada por la función proporcionada.

const numbers = [1, 2, 3, 4, 5];
const allNumbersAreGreaterThanZero = numbers.every(num => num > 0);
console.log(allNumbersAreGreaterThanZero); // true
```

5

Profesor: Arle Morales Ortiz

```
//Some:
//El método some() comprueba si al menos un elemento del array pasa
// la prueba implementada por la función proporcionada.

const numbers = [1, 2, 3, 4, 5];
const someNumbersAreGreaterThanFour = numbers.some(num => num > 4);
console.log(someNumbersAreGreaterThanFour); // true
```

6

Métodos: iteración y transformación

```
const cities = ["Armenia", "Pereira", "Cali"];
cities.forEach(x => console.log(x));

console.log(cities.map(x=> x)); // [ 'Armenia', 'Pereira', 'Cali' ]

const y = cities.find(x => {   // Cali
    if(x == 'Cali'){
        return x;
    } }
);
console.log(y);

const f = cities.filter( x =>{ //[ 'Pereira' ]
    if(x == 'Pereira'){
            return x;
        }
    }
);
console.log(f);
```

1

2

3

4

Profesor: Arle Morales Ortiz

map

Hay tres formas comunes de crear mapas.

```
1  const emptyMap = new Map(); // 1. map vacio
2
3  const animals = new Map( entries: [
4      [1, 'horse'],
5      [2, 'dog'],
6      [3, 'gato']
7  ]);
8
9  const categories = new Map()
10     .set(1, 'shoes')
11     .set(2, 'dresses')
12     .set(3, 'stockings');
13
14 const copy = new Map(categories);
15
16 animals.set(4,"giraffe"); // setear un elemento
17 console.log(animals.get(2)); // traer un elemento -> dog
18 console.log(categories.size); // tamaño del map -> 3
19
20 categories.forEach( callbackfn: x => console.log(x));
21 for (const [key, value] of animals) {
22     console.log(key, value);
23 }
24
25 const objAnimals = Object.fromEntries(categories);
26
27 console.log(objAnimals);
```

1 — 1

2 — 2

3 — 3

copiar

iterar valores

mostrar clave - valor

Convertir mapas a objetos

Los métodos y propiedades son:

- `new Map()` – crea el mapa.
- `map.set(clave, valor)` – almacena el valor asociado a la clave.
- `map.get(clave)` – devuelve el valor de la clave. Será `undefined` si la `clave` no existe en map.
- `map.has(clave)` – devuelve `true` si la `clave` existe en map, `false` si no existe.
- `map.delete(clave)` – elimina el valor de la clave.
- `map.clear()` – elimina todo de map.
- `map.size` – tamaño, devuelve la cantidad actual de elementos.

```
1   // Agregar una entrada al mapa
2   const map = new Map();
3   map.set("clave1", "John Doe");
4   map.set("clave2", "John Doe");
5
6   // Obtener el valor de una clave
7   const value = map.get("clave1");
8   console.log(value); // John Doe
9
10  // Comprobar si una clave existe
11  const exists = map.has("name");
12  console.log(exists); // false
13
14  // Eliminar una entrada
15  map.delete("clave1");
16
17  // Obtener todas las claves
18  const keys = map.keys();
19  console.log(keys); // ['name']
20
21  // Obtener todos los valores
22  const values = map.values();
23  console.log(values); // ['John Doe']
24
25  // Obtener todos los pares de clave-valor
26  const entries = map.entries();
27  console.log(entries); // [['name', 'John Doe']]
28
29  // Eliminar todas las entradas
30  map.clear();
```

1 2 3 4 5 6 7 8

Profesor: Arle Morales Ortiz

Set

Un `Set` es una colección de tipo especial: "conjunto de valores" (sin claves), donde cada valor puede aparecer solo una vez.

Sus principales métodos son:

- `new Set(iterable)` – crea el set. El argumento opcional es un objeto iterable (generalmente un array) con valores para inicializarlo.
- `set.add(valor)` – agrega un valor, y devuelve el set en sí.
- `set.delete(valor)` – elimina el valor, y devuelve `true` si el `valor` existía al momento de la llamada; si no, devuelve `false`.
- `set.has(valor)` – devuelve `true` si el valor existe en el set, si no, devuelve `false`.
- `set.clear()` – elimina todo el continido del set.
- `set.size` – es la cantidad de elementos.

La característica principal es que llamadas repetidas de `set.add(valor)` con el mismo valor no hacen nada. Esa es la razón por la cual cada valor aparece en `Set` solo una vez.

Por ejemplo, vienen visitantes y queremos recordarlos a todos. Pero las visitas repetidas no deberían llevar a duplicados. Un visitante debe ser "contado" solo una vez.

```
let set = new Set();
const set2 = new Set( values: ['red', 'green', 'blue']);
const set3 = new Set()
    .add('red')
    .add('green')
    .add('blue');

let john = { name: "John" };
let pete = { name: "Pete" };
let mary = { name: "Mary" };

set.add(john);
set.add(pete);
set.add(mary);
set.add(john);
set.add(mary);

console.log( set.size ); // 3

for (let user of set) {
    console.log(user.name); // John (luego Pete y Mary)
```

Profesor: Arle Morales Ortiz

```
const mySet = new Set([1, 2, 3, 4, 5]);

//Podemos eliminar elementos de un conjunto usando el método delete():
mySet.delete(3);

//Podemos verificar si un valor existe en un conjunto usando el método has():
console.log(mySet.has(4)); // true

//Podemos iterar sobre los elementos de un conjunto usando un bucle for-of:
for (const value of mySet) {
    console.log(value);
}
//Podemos iterar
mySet.forEach(x => console.log(x))
```

Matrices

Las matrices de dos dimensiones son un tipo de matriz que tiene dos dimensiones, es decir, se puede visualizar como una tabla de datos. Las matrices de dos dimensiones se pueden usar para almacenar datos de una manera organizada y eficiente.

```
let myMatrix = [[1, 2, 3], [4, 5, 6]];

let firstElement = myMatrix[0][2];
console.log(firstElement) // 3

myMatrix.push([2,8,9])

// Iterar sobre la matriz utilizando una función
myMatrix.forEach((element, rowIndex, columnIndex) => {
    console.log(` f${rowIndex}: ${element}`);
});
```