Lagash™

Funciones de coleciones

Lagash University

Alfredo
Felipe
Natalia
Nicolas

filter()

El método filter() crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
var newArray = arr.filter(callback(currentValue[, index[, array]])[, thisArg])
```

Fuente: https://developer.mozilla.org/

Parámetros filter

```
var newArray = arr.filter(callback(currentValue[, index[, array]])[, thisArg])
```

- callback
- currentValue
- index (opcional)
- array (opcional)
- thisArg (opcional)

Ejemplo práctico filter

filter.js

```
1  let arregloNumeros = [1,2,3,4,5,6,7,8,9,10];
2
3  let numerosMayores = arregloNumeros.filter(numero =>
4      numero > 5
5  )
6
7  numerosMayores.forEach(numero => console.log(numero));
8
```

felipe@ZENBOOK: /mnt/c/Users/felip/Desktop/test

```
felipe@ZENBOOK:/mnt/c/Users/felip/Desktop/test$ node filter.js
6
7
8
9
10
felipe@ZENBOOK:/mnt/c/Users/felip/Desktop/test$
```

LAGASH UNIVERSITY

Array.reduce()

El método reduce() aplica una función a un acumulador y a cada valor de una array (de izquierda a derecha) para reducirlo a un único valor.

Array.reduce()

```
2  const numbers = [1, -1, 2, 3];
3
4  let sum = 0;
5  for (let n of numbers)
6    sum += n;
7
8  console.log(sum);
9
```

```
10 const sum = numbers.reduce((accumulator, currentValue) => {
11   return accumulator + currentValue;
12 }, 0);
```

Array.reduce()

```
10  const sum = numbers.reduce((accumulator, currentValue) => {
11    return accumulator + currentValue;
12  }, 0);
```

```
a=0, c=1 => a=1
a=1, c=-1 => a=0
a=0, c=2 => a=2
a=2, c=3 => a=5
```

Array.reduce()

```
let vals = [5, 4, 9, 2, 1];

let biggest = vals.reduce((acc, val) => {
  if (val > acc) {
    acc = val;
  }
  return acc;
});
console.log(biggest);
```

```
let biggest = vals.reduce((a, b) => a > b ? a : b);
console.log(biggest);
```

forEach

El método forEach() ejecuta la función indicada una vez por cada elemento del array

```
arr.forEach(function callback(currentValue, index, array) {
    // tu iterador
}[, thisArg]);
```

Fuente: https://developer.mozilla.org/

Parámetros forEach

```
arr.forEach(function callback(currentValue, index, array) {
    // tu iterador
}[, thisArg]);
```

- callback
- currentValue
- index (opcional)
- array (opcional)
- thisArg (opcional)

Ejemplo práctico forEach

index.js

```
1  let arregloPersonas = ['Alfredo', 'Natalia', 'Felipe', 'Nicolas'];
2
3  arregloPersonas.forEach((elemento,indice)=> {
4      console.log('Elemento en indice ' + indice + ': ' + elemento);
5  });
```

felipe@ZENBOOK: /mnt/c/Users/felip/Desktop/test

```
felipe@ZENBOOK:/mnt/c/Users/felip/Desktop/test$ node index.js
Elemento en indice 0: Alfredo
Elemento en indice 1: Natalia
Elemento en indice 2: Felipe
Elemento en indice 3: Nicolas
```

map

El método map() crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

Sintaxis

```
var nuevo_array = arr.map(function callback(currentValue, index, array) {
    // Elemento devuelto de nuevo_array
}[, thisArg])
```

Ejemplo práctico map

```
1  var numeros = [1, 4, 9];
2  var dobles  = numeros.map(function(num) {
3    return num * 2;
4  });
5
6  // dobles es ahora [2, 8, 18]
7  // numeros sigue siendo [1, 4, 9]
```

for of

La **sentencia `for...of`** crea un bucle que itera a través de los elementos de objetos iterables (incluyendo `Array`, `Map`, `Set`, el objeto arguments, etc.), ejecutando las sentencias de cada iteración con el valor del elemento que corresponda.

Sintáxis for of

```
for (variable of iterable) {
  statement
}
```

Ejemplo práctico for of

```
let nombres=['Natalia','Nicole','Antonia','Isidora'];

for(let i of nombres){
    console.log(i);
}
```

```
PS C:\Users\Usuario\Desktop\ejerciciospresentacion> node index.js
Natalia
Nicole
Antonia
Isidora
```

for in

La sentencia for...in itera sobre todos los elementos de un objeto, en un orden arbitrario. Para cada uno de los elementos, se ejecuta la sentencia especificada

Ejemplo práctico for in

```
var obj = {a: 1, b: 2, c: 3};

for (const prop in obj) {
  console.log(`obj.${prop} = ${obj[prop]}`);
}

// Salida:
// "obj.a = 1"
// "obj.b = 2"
// "obj.c = 3"
```

Funciones para el manejo de colecciones

Funciones para añadir elementos:

- **push()**

arreglo.push(nuevoElemento);

- **unshift()**

arreglo.unshift(nuevoElemento);

Funciones para extraer elementos y eliminar

- **Pop()**

let ultimoElemento: arreglo.pop();

- **Shift()**

let primerElemento = arreglo.shift();

- **Splice()**

arreglo.splice(1,0,'nuevoElemento');

Funciones para el manejo de colecciones

Funciones para extraer elementos:

- **slice(a,b)**

```
let result = [1, 2, 3, 4, 5].slice(1,4);
```

- **slice()**

```
let result = [1, 2, 3, 4, 5].slice();
```

Funciones para recuperar índices de posiciones

- **indexOf(x)**

```
let result = arreglo.indexOf(14);
```

- **lastIndexOf(x)**

```
let result = arreglo.lastIndexOf(14);
```

Funciones para el manejo de colecciones

Funciones para ordener arreglo:

- **reverse()**

Let nuevoArreglo = arreglo.reverse();

- **sort()**

let arreglo = [2,11,111,7].sort();
// arreglo = [11,111,2,7];