Recuerden poner a grabar la clase

Clase 12

Funciones de orden Superior

Diplomatura UNTREF

Temario

Functions de order Superior

- forEach
- find
- findLast
- filter
- map
- sort
- group
- some
- flat
- reduce (arrays destructivos vs arrays no destructivos)

Funciones de orden superior

Funciones de orden superior

Funciones de orden superior, también conocidas como funciones de alta orden, son un concepto importante en la programación funcional.

Estas funciones tratan a las funciones como ciudadanos de primera clase, lo que significa que pueden ser pasadas como argumentos a otras funciones, devueltas como valores de otras funciones y almacenadas en variables.

Funciones de orden superior

En JavaScript, las funciones de orden superior pueden realizar varias operaciones útiles y flexibles.
Te permiten escribir código más modular, reutilizable y flexible al tratar las funciones como objetos que pueden manipularse y combinarse de varias formas.

Funciones de orden superior

Tipos de funciones:

Funciones como argumentos: Puedes pasar una función como argumento a otra función. Esto es útil para abstraer ciertos comportamientos y hacer que tu código sea más reutilizable.

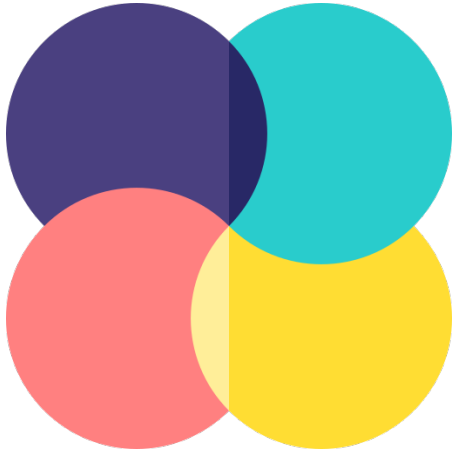
Funciones como valores de retorno: Puedes devolver una función desde otra función. Esto es útil para crear funciones especializadas o para configurar ciertos comportamientos de funciones según el contexto.

Cierre (closures): Las funciones de orden superior permiten crear cierres. Un cierre es una función que "recuerda" el ámbito en el que se creó y puede acceder a las variables de ese ámbito incluso después de que la función que lo creó haya finalizado su ejecución.

Aplicación en Arrays

Aplicación en Arrays

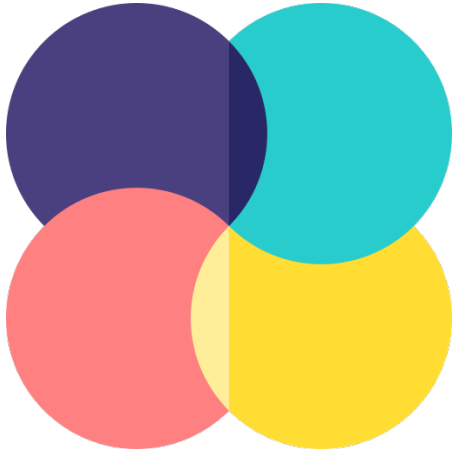
Si bien podemos crear nuestras propias funciones de orden superior para optimizar al máximo nuestro código, también contamos con muchas funciones de orden superior integradas en los arrays JS.



Estas nos permiten exprimir al máximo las capacidades de procesar y obtener resultados efectivos, a través de un mecanismo de abstracción fácil de comprender.

Aplicación en Arrays

La mayoría de las funciones de orden superior, aparecieron en JS desde hace varios años, gracias al modelo colaborativo de compartir conocimientos en Internet a través de foros de discusión.



Originalmente, la iteración de un array de elementos u objetos, se realizaba a través del ciclo for convencional.

Aplicación en Arrays

Ejemplo de iteración de un array de objetos, previo a la existencia del método **f or Each( )** .

Funciones de orden superior

```
const productos = [
  { id: 1, nombre: 'Producto 1', precio: 100 },
  { id: 2, nombre: 'Producto 2', precio: 200 },
  { id: 3, nombre: 'Producto 3', precio: 150 }
];

function forEachProduct() {
  for (let i = 0; i < productos.length; i++) {
    console.log(productos[i].nombre); //imprime el nombre del producto
  }
}
```

Aplicación en Arrays

JQuery también fue parte de la inspiración de estas funciones de orden superior, a partir de la propuesta de su método **$.each()**, para iterar elementos de un objeto u array.

Funciones de orden superior

```
const colores = [
  { 'rojo': '#f00' },
  { 'verde': '#0f0' },
  { 'azul': '#00f' }
];

$.each(colores, function() {
  $.each(this, function(name, value) {
    console.log(`${name} = ${value}`);
  });
});
```

Aplicación en Arrays

Los siguientes ejemplos a repasar, están aplicados sobre este array modelo. Ten presente esto por sí, al repasar este contenido, quieres ir probando el código de cada explicación.

Funciones de orden superior

```
const productos = [
  { id: 1, nombre: 'Producto 1', precio: 100 },
  { id: 2, nombre: 'Producto 2', precio: 200 },
  { id: 3, nombre: 'Producto 3', precio: 150 }
];
```

forEach

forEach

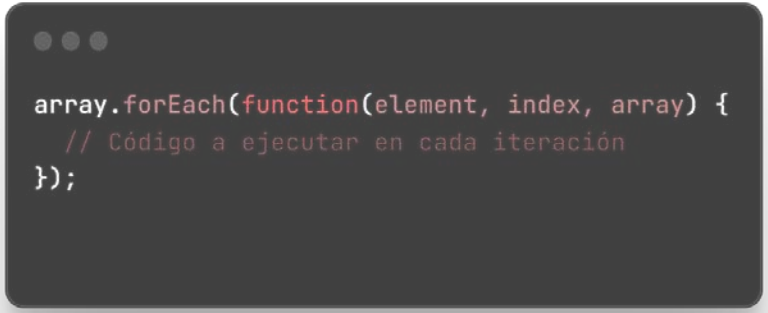
Es una **función de orden superior** que se utiliza con arreglos (arrays) para recorrer cada elemento del arreglo y aplicar una función a cada uno de esos elementos.

forEach es una forma de realizar iteraciones sobre los elementos de un arreglo sin necesidad de utilizar un bucle for tradicional.

Sintaxis de ForEach

La sintaxis básica de forEach es la siguiente:

- element es el elemento actual del arreglo en la iteración actual.
- index es el índice del elemento actual en el arreglo.
- array es el arreglo sobre el que se está iterando.



Sintaxis de ForEach

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Ejemplo aplicado:

En este caso solamente imprime cada número en consola.

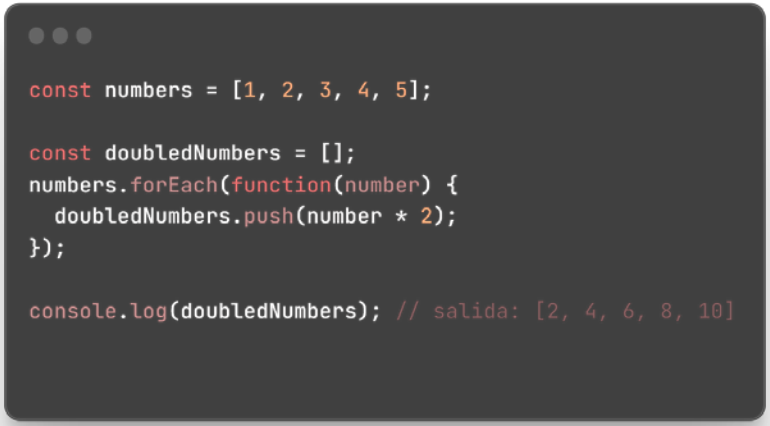
```
const numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number) {
  console.log(number);
});
```

Cuando usar ForEach

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

forEach es útil cuando deseas realizar una operación en cada elemento de un arreglo, como realizar un cálculo, modificar los valores, o realizar cualquier otra acción que deba llevarse a cabo en cada elemento sin necesidad de preocuparte por los detalles de la iteración.

Es importante tener en cuenta que **forEach** no retorna un nuevo arreglo; simplemente realiza una acción en cada elemento del arreglo original.



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

const doubledNumbers = [];
numbers.forEach(function(number) {
  doubledNumbers.push(number * 2);
});

console.log(doubledNumbers); // salida: [2, 4, 6, 8, 10]
```

Cuando usar ForEach

Si deseas generar un nuevo arreglo transformado a partir de los elementos del arreglo original, podrías considerar otras funciones de orden superior como **map**.

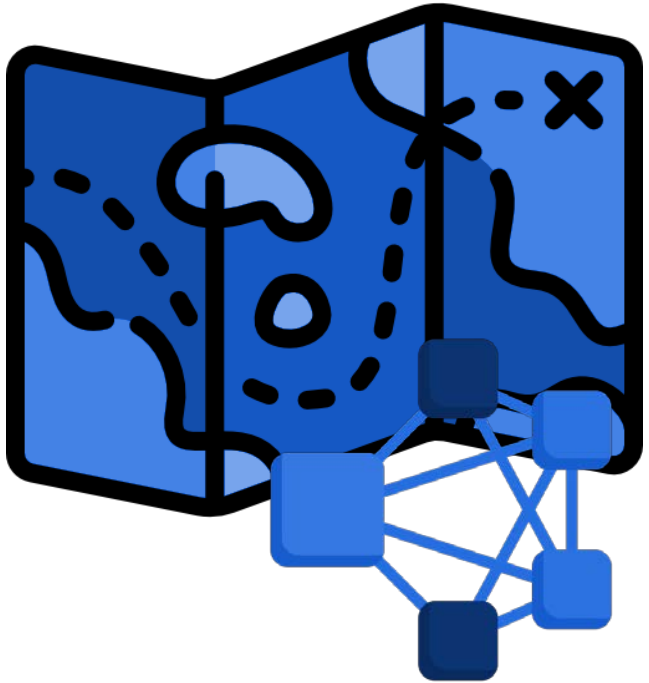
map

map

Es una función de orden superior que se utiliza con arreglos (arrays) para crear un **nuevo arreglo** transformando cada elemento del arreglo original mediante una función proporcionada.

La **función map** toma cada elemento del arreglo original, aplica una función a ese elemento y luego agrega el resultado de la función a un nuevo arreglo.

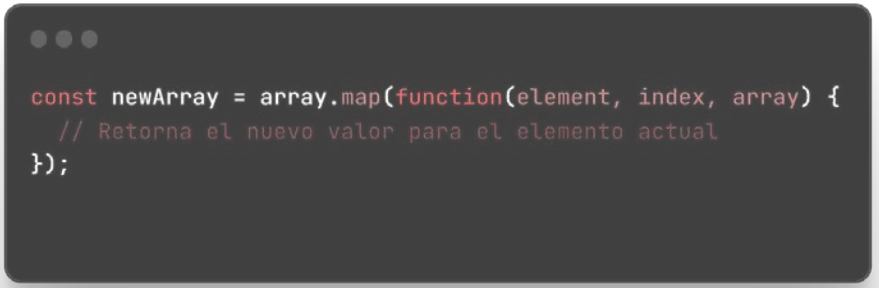
Esto permite crear un nuevo arreglo con los elementos transformados sin modificar el arreglo original.



Sintaxis map

La sintaxis básica de map es la siguiente:

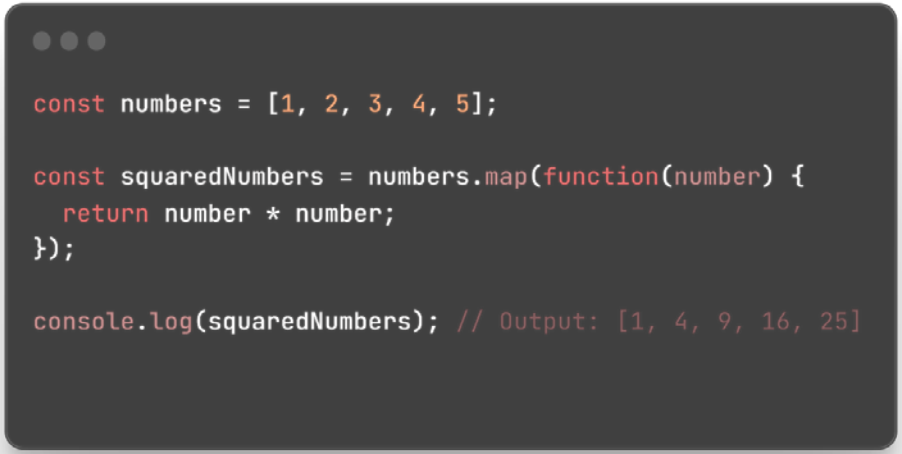
- **element** es el elemento actual del arreglo en la iteración actual.
- **index** es el índice del elemento actual en el arreglo.
- **array** es el arreglo sobre el que se está iterando.



Sintaxis map

Ejemplo aplicado:

La **función pasada a map** toma cada número del arreglo numbers, lo eleva al cuadrado y agrega los valores resultantes al nuevo arreglo squaredNumbers.



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

const squaredNumbers = numbers.map(function(number) {
  return number * number;
});

console.log(squaredNumbers); // Output: [1, 4, 9, 16, 25]
```

Cuando usar map

Una de las **ventajas de map** es que genera un nuevo arreglo sin modificar el arreglo original, lo que facilita la inmutabilidad de los datos y ayuda a mantener un código más limpio y predecible.

la **función map** se utiliza para crear un nuevo arreglo de nombres a partir del arreglo de objetos people.

```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 }
];

const names = people.map(function(person) {
  return person.name;
});

console.log(names); // Output: ['Alice', 'Bob', 'Charlie']
```

find

find

Es una **función de orden superior** que se utiliza con arreglos (arrays) para buscar el primer elemento en el arreglo que cumple con una condición especificada. Una vez que se encuentra el primer elemento que cumple con la condición, la función find retorna dicho elemento y deja de buscar en el resto del arreglo.

Sintaxis find

La sintaxis básica de find es la siguiente:

- element es el elemento actual del arreglo en la iteración actual.
- index es el índice del elemento actual en el arreglo.
- array es el arreglo sobre el que se está iterando.

```
const result = array.find(function(element, index, array) {
  // Retorna true si el elemento cumple con la condición, sino retorna false
});
```

Cuando usar find

find es útil cuando deseas encontrar un único elemento en un arreglo que cumpla con una cierta condición. Si no se encuentra ningún elemento que cumpla con la condición, find retornará undefined.

En este ejemplo se genero un arreglo de objetos y se desea encontrar el primer objeto que cumple con algunas condiciones.
find se utiliza para encontrar la primera persona en el arreglo people que tenga una edad mayor a 25.

```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 }
];

const person = people.find(function(person) {
  return person.age > 25;
});

console.log(person); // Output: { name: 'Bob', age: 30 }
```

findLast

findLast

Es una función de orden superior que opera en arreglos (arrays).

Al igual que Array.prototype.find(), findLast() busca elementos en el arreglo que cumplan con cierta condición, pero en lugar de encontrar el primer elemento que cumple con la condición, busca el último.

```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 }
];

const person = people.find(function(person) {
  return person.age > 25;
});

console.log(person); // Output: { name: 'Bob', age: 30 }
```

Sintaxis findLast

Esto significa que findLast() itera sobre los elementos del arreglo en orden inverso hasta encontrar un elemento que satisfaga la condición.

La sintaxis básica de Array.prototype.findLast() es la siguiente:

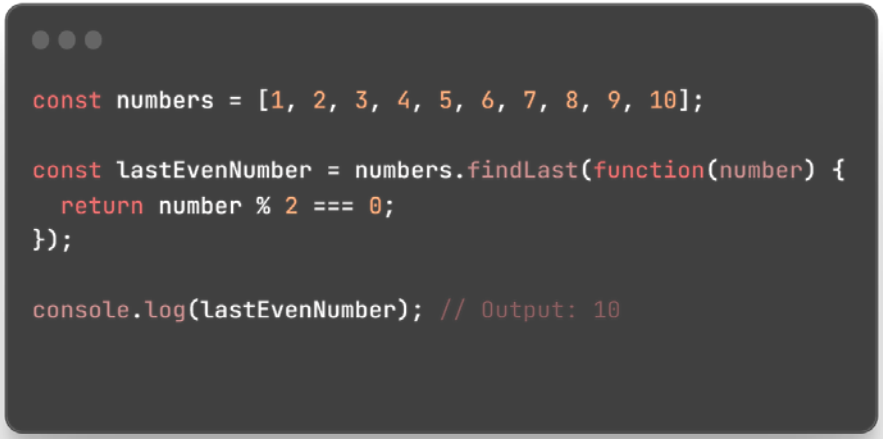
- **element** es el elemento actual del arreglo en la iteración actual.
- **index** es el índice del elemento actual en el arreglo.
- **array** es el arreglo sobre el que se está iterando.

```
const result = array.findLast(function(element, index, array) {
  // Retorna true si el elemento cumple con la condición, sino retorna false
});
```

Cuando usar findLast

En este caso, la función pasada a findLast() busca el último número par en el arreglo numbers y retorna el número 10.

En resumen, Array.prototype.findLast() es una función de orden superior que te permite buscar y encontrar el último elemento que cumple con una condición en un arreglo.



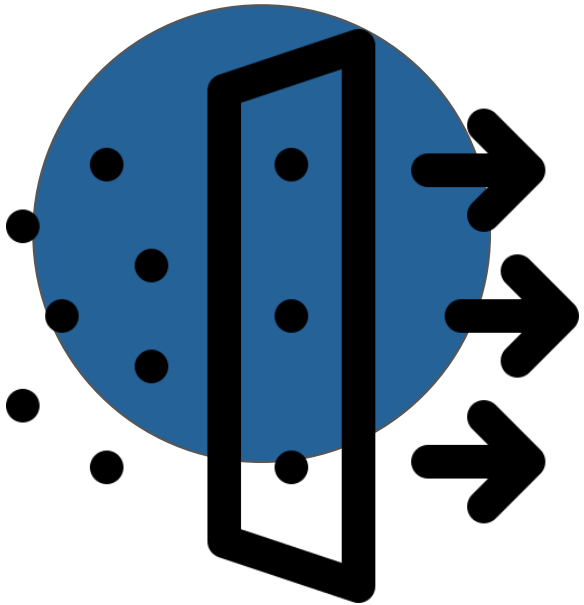
```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const lastEvenNumber = numbers.findLast(function(number) {
  return number % 2 === 0;
});

console.log(lastEvenNumber); // Output: 10
```

filter

Filter

es una función de orden superior que se utiliza con arreglos (arrays) para crear un nuevo arreglo que contiene todos los elementos del arreglo original que cumplen con una condición especificada.

En esencia, filter filtra los elementos del arreglo original según una función de prueba y devuelve un nuevo arreglo que contiene solo los elementos que pasan esa prueba.



Sintaxis Filter

La sintaxis básica de filter es la siguiente:

- element es el elemento actual del arreglo en la iteración actual.
- index es el índice del elemento actual en el arreglo.
- array es el arreglo sobre el que se está iterando.

Aquí tienes un ejemplo simple que muestra cómo usar filter

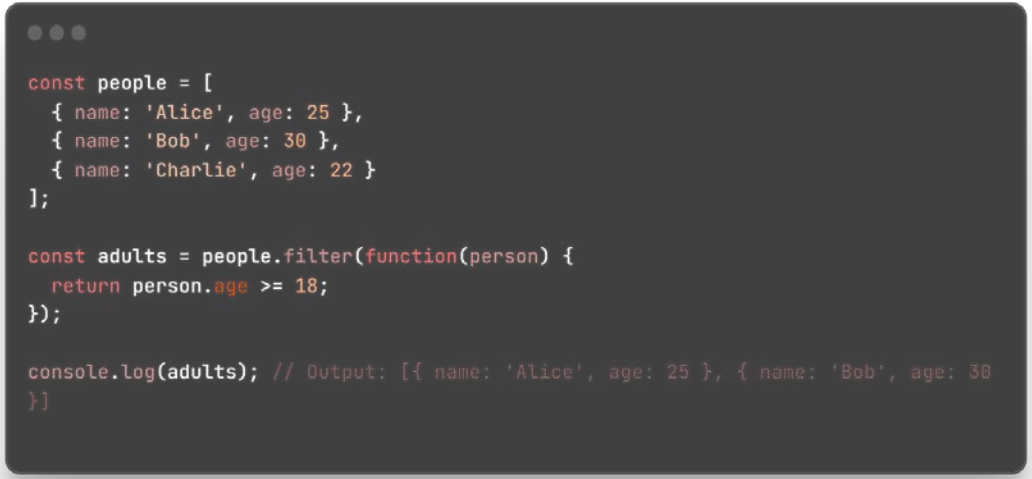
```
const newArray = array.filter(function(element, index, array) {
  // Retorna true si el elemento cumple con la condición, sino retorna false
});
```

Cuando usar Filter

filter filtra los números pares del arreglo numbers y devuelve un nuevo arreglo llamado evenNumbers que contiene solo esos números.

filter es útil cuando deseas crear un subconjunto de elementos del arreglo original que cumplen con una cierta condición.

Puedes usarlo para filtrar elementos basados en propiedades específicas, valores, o cualquier otra condición que puedas definir en la función de prueba.



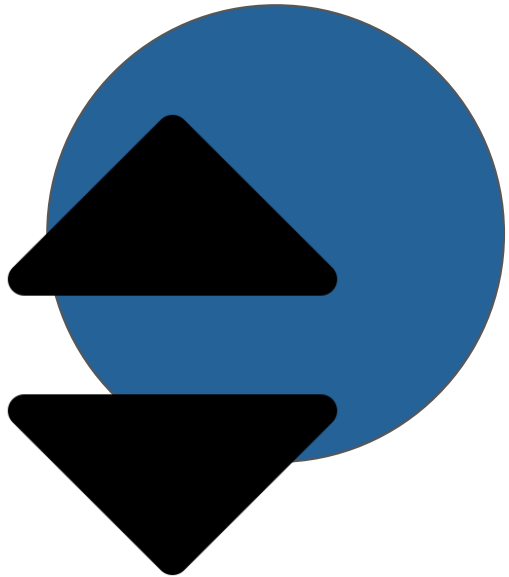
```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 }
];

const adults = people.filter(function(person) {
  return person.age >= 18;
});

console.log(adults); // Output: [{ name: 'Alice', age: 25 }, { name: 'Bob', age: 30
}]
```

sort

sort

Es una función de orden superior que se utiliza con arreglos (arrays) para ordenar los elementos del arreglo según un criterio específico.

La función sort toma una función de comparación (opcional) como argumento que define cómo se deben comparar y ordenar los elementos.



Sintaxis de sort

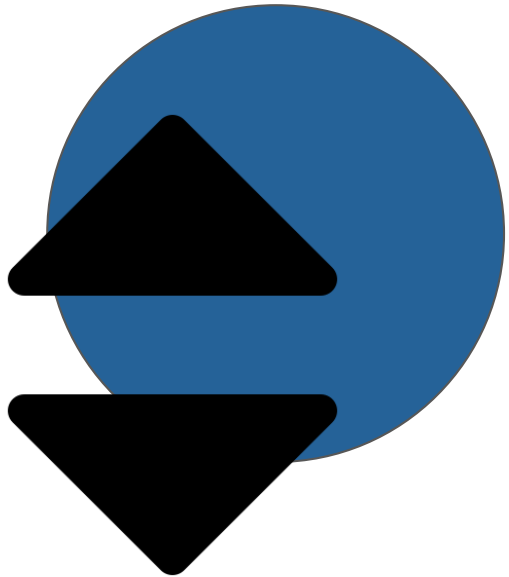
La sintaxis básica de **sort** es la siguiente:

compareFunction es una función que compara dos elementos y devuelve un valor numérico que indica su relación de orden. Es opcional.

Si no se proporciona **compareFunction**, los elementos se ordenarán como cadenas de caracteres en orden lexicográfico.

compareFunction

La función **compareFunction** debe retornar un valor negativo si el primer elemento debe ordenarse antes que el segundo, un valor positivo si el primer elemento debe ordenarse después que el segundo, o cero si los elementos son iguales en términos de orden.



compareFunction

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

En este caso, se utiliza sort para ordenar los números en orden ascendente.

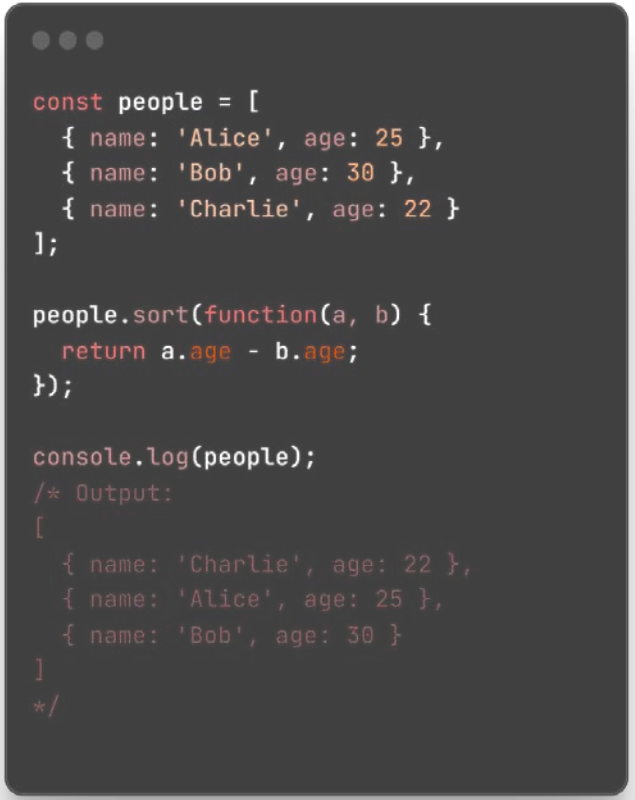
La función de comparación function(a, b) { return a - b; } resta b de a, lo que significa que los números más pequeños estarán al principio del arreglo.

```
const numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];

numbers.sort(function(a, b) {
  return a - b;
});

console.log(numbers); // Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

Cuando utilizar sort

sort es útil cuando deseas reorganizar los elementos de un arreglo en un orden específico, ya sea ascendente o descendente, basado en un criterio específico.

Ten en cuenta que la función sort modifica el arreglo original y no crea un nuevo arreglo.



```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 }
];

people.sort(function(a, b) {
  return a.age - b.age;
});

console.log(people);
/* Output:
[
  { name: 'Charlie', age: 22 },
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 }
]
*/
```

some

Some

Es una función de orden superior que se utiliza con arreglos (arrays) para verificar si al menos un elemento del arreglo cumple con una cierta condición.

La función **some** retorna true si al menos uno de los elementos satisface la condición especificada en la función de prueba, y retorna false si ninguno de los elementos cumple con esa condición.

Sintaxis de Some

La sintaxis básica de some es la siguiente:

- element es el elemento actual del arreglo en la iteración actual.
- index es el índice del elemento actual en el arreglo.
- array es el arreglo sobre el que se está iterando.

```
const result = array.some(function(element, index, array) {
  // Retorna true si el elemento cumple con la condición, sino retorna false
});
```

Cuando usar Some

some es útil cuando deseas determinar si al menos uno de los elementos en el arreglo satisface cierta condición. Puede ser útil para verificar la existencia de elementos que cumplan con ciertas características sin tener que iterar manualmente sobre el arreglo.

Puedes usar some para verificar si hay algún adulto en un arreglo de personas:

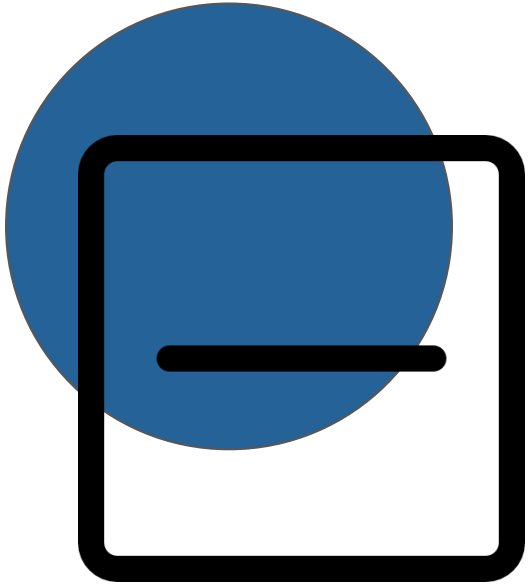
```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 17 },
  { name: 'Charlie', age: 22 }
];

const hasAdult = people.some(function(person) {
  return person.age >= 18;
});

console.log(hasAdult); // Output: true
```

flat

flat

flat no es propiamente una función de orden superior, sino un método de los arreglos (arrays) en JavaScript que se utiliza para "aplanar" arreglos anidados, es decir, convertir un arreglo multidimensional en un arreglo de una sola dimensión.

Este método es especialmente útil cuando trabajas con arreglos que contienen otros arreglos como elementos.



Sintaxis de flat

La sintaxis básica de flat es la siguiente:

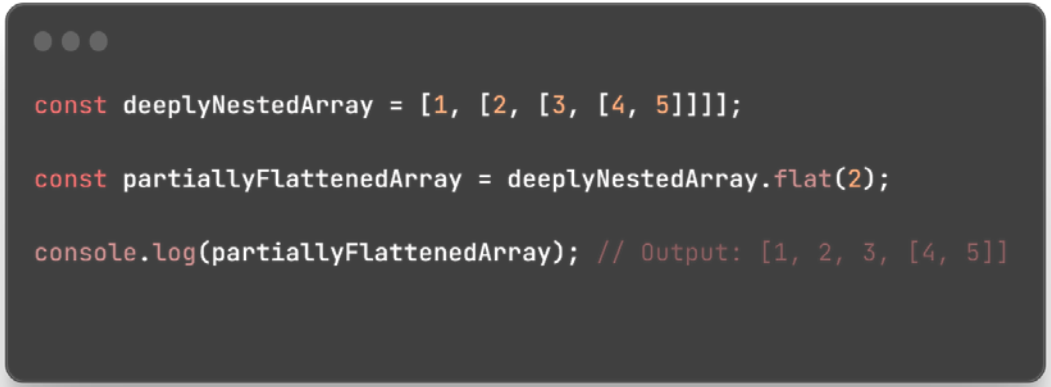
depth es opcional y define la profundidad máxima de anidamiento que se debe aplanar. Si no se proporciona, se asume un valor de 1.

```
const newArray = array.flat([depth]);
```

Cuando usar flat

flat es útil cuando tienes arreglos anidados y deseas trabajar con una estructura de arreglo más simple y lineal.

En el ejemplo, nestedArray es un arreglo anidado que contiene otros arreglos como elementos. Al aplicar flat() sin argumentos, se obtiene un nuevo arreglo flattenedArray donde los arreglos internos se han "aplanado" en una sola dimensión.



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

const partiallyFlattenedArray = deeplyNestedArray.flat(2);

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

Cuando usar flat

Suponiendo que tenemos un array llamado **otrosProductos**, con un set de datos que necesitamos unificar en el array principal.

Podemos agregar este array dentro del principal utilizando **productos.push(otrosProductos)**. Esto nos dará como resultado un array de dos dimensiones.

De transformación: flat()

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Este array multidimensional puede ser unificado generando un nuevo array mediante el método **.flat()**.

En este, podemos especificar el nivel de profundidad que deseamos aplicar sobre el array, por si el array de la segunda dimensión posee un array más como elemento contenido.

```
console.table(productos)
```

VM773:1

(índice)	id	nombre	precio	stock	categor…	0	1
0	1	'TV 55'	1000	30	'Video'		
1	2	'Laptop'	1500	100	'Comput…		
2	3	'iPhone…	800	42	'Telefo…		
3	4	'Tablet'	500	71	'Comput…		
4	5	'Pods'	100	28	'Audio'		
5	6	'MP3 pl…	200	11	'Audio'		
6	7	'Videoc…	300	22	'Video'		
7	8	'Smartw…	250	88	'Comput…		
8	9	'Impres…	150	14	'Acceso…		
9	10	'Altavo…	120	18	'Audio'		
10						{…}	{…}

▶ Array(11)

Funciones de orden superior

```
const productosUnificados = productos.flat(1);

console.table(productosUnificados);
```

De transformación: flat()

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Así unificaremos rápidamente en una sola dimensión, un array de elementos de un mismo tipo.

VM1040:1

(índice)	id	nombre	precio	stock	categoria
0	1	'TV 55'	1000	30	'Video'
1	2	'Laptop'	1500	100	'Computación'
2	3	'iPhone 8'	800	42	'Telefonía'
3	4	'Tablet'	500	71	'Computación'
4	5	'Pods'	100	28	'Audio'
5	6	'MP3 playe…	200	11	'Audio'
6	7	'Videocáma…	300	22	'Video'
7	8	'Smartwatc…	250	88	'Computación'
8	9	'Impresora'	150	14	'Accesorios'
9	10	'Altavoces'	120	18	'Audio'
10	11	'Printer H…	550	7	'Accesorios'
11	12	'Altavoces…	199	8	'Audio'

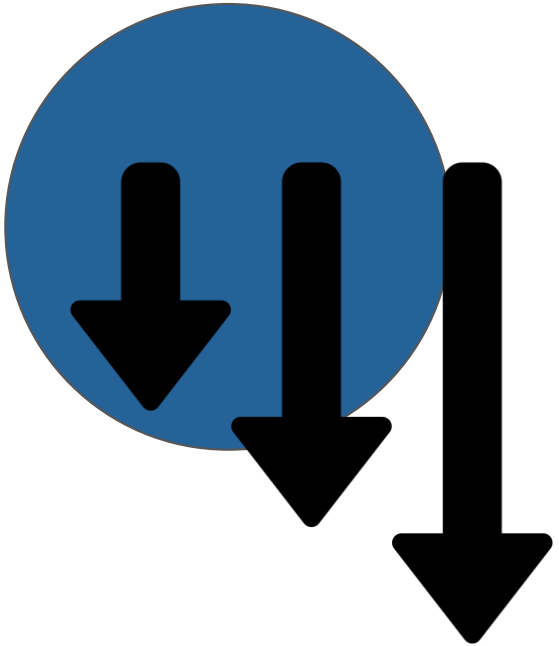
▶ Array(12)

reduce

reduce

Es una función de orden superior que se utiliza con arreglos (arrays) para combinar todos los elementos del arreglo en un solo valor acumulado, utilizando una función reductora.

La **función reductora** toma dos argumentos: el **acumulador** y el **elemento** actual del arreglo, y retorna un valor que se acumula a medida que la función se aplica a cada elemento.



Sintaxis de reduce

La sintaxis básica de reduce es la siguiente:

- **accumulator** es el valor acumulado actual.
- **element** es el elemento actual del arreglo en la iteración actual.
- **index** es el índice del elemento actual en el arreglo.
- **array** es el arreglo sobre el que se está iterando.
- **initialValue** es un valor opcional que se utiliza como el valor inicial del acumulador. Si no se proporciona, el primer elemento del arreglo se usa como valor inicial y la función comienza a ejecutarse a partir del segundo elemento.

```
const result = array.reduce(function(accumulator, element, index, array) {
  // Retorna el valor acumulado después de procesar el elemento actual
}, initialValue);
```

Cuando usar reduce

Reduce es muy flexible y versátil.
Puede usarse para realizar una amplia variedad de operaciones, como sumas, multiplicaciones, concatenaciones, cálculos de promedio, y más.

También es especialmente útil para transformar arreglos en un solo valor o estructura de datos.

Por ejemplo, puedes usar reduce para obtener el promedio de los valores de un arreglo:

```
const values = [10, 20, 30, 40, 50];

const average = values.reduce(function(accumulator, value, index,
array) {
  accumulator += value;
  if (index === array.length - 1) {
    return accumulator / array.length;
  } else {
    return accumulator;
  }
}, 0);

console.log(average); // Output: 30
```

Metodo destructivo

¿Qué es un método destructivo?

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

En JS, se conoce como método destructivo, o **in-place**, a alguna función o método que, al ejecutarse, altera la estructura original del objeto o array donde éste se aplica.

Este concepto lo tenemos que tener presente porque, de necesitar trabajar siempre con un array original, este método puede alterarlo y romper alguna parte lógica necesaria en nuestra aplicación.

Como alternativa y ante la duda, siempre podemos copiar/clonar un array, previo a ejecutar algún método o función destructiva.

¿Dudas o consultas?

¡Muchas Gracias!