BOOTCAMP

Generación T

streambe

El Método slice() en JavaScript

Extracción sin modificación

Crea una copia de una porción del array o string original sin modificarlo.

Sintaxis flexible

Acepta uno o dos parámetros para definir el inicio y fin de la extracción.

Índices negativos

Permite contar desde el final usando valores negativos.

A diferencia de otros métodos, slice() no altera la estructura original. Ideal para crear subconjuntos de datos manteniendo intacta la fuente.

```
const numeros = [1, 2, 3, 4, 5];
const subArray = numeros.slice(1, 3); // [2, 3]

const texto = "JavaScript";
const subTexto = texto.slice(4); // "Script"
```

join() y split() en JavaScript

El Método join()

Une todos los elementos de un array en una cadena de texto.

- Convierte arrays a strings
- Acepta un separador como parámetro
- No modifica el array original

```
const frutas = ["manzana", "pera", "naranja"];

const listado = frutas.join(", "); // "manzana, pera, naranja"
```

El Método split()

Divide un string en un array de substrings.

- Convierte strings a arrays
- Requiere un separador como parámetro
- Retorna un nuevo array

```
const texto = "JavaScript es poderoso";

const palabras = texto.split(" "); // ["JavaScript", "es", "poderoso"]
```

Aplicaciones Prácticas

- Formateo de datos para visualización
- Procesamiento de archivos CSV
- Manipulación de URLs y rutas
- Análisis de texto y tokenización
- Transformación de datos para APIs

Métodos filter() y find() en JavaScript

filter()

Crea un nuevo array con elementos que cumplen una condición. Devuelve múltiples elementos que pasan el test.

find()

Retorna el primer elemento que cumple la condición. Se detiene una vez encuentra una coincidencia.

Sintaxis similar

Ambos métodos aceptan una función callback que evalúa cada elemento del array.

```
// Ejemplo de filter()
const numeros = [1, 2, 3, 4, 5, 6];
const pares = numeros.filter(num => num % 2 === 0);
// pares = [2, 4, 6]

// Ejemplo de find()
const usuarios = [{id: 1, nombre: "Ana"}, {id: 2, nombre: "Carlos"}];
const usuario = usuarios.find(u => u.id === 2);
// usuario = {id: 2, nombre: "Carlos"}
```

Iterar Un Arreglo

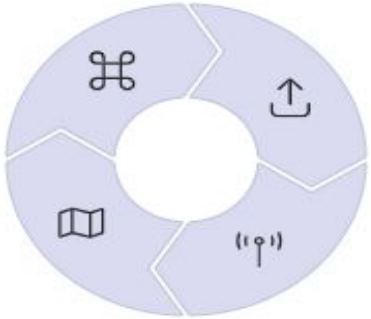
Recorrer cada elemento de un array es una operación fundamental en JavaScript.

for clásico

Utiliza un contador para acceder a cada índice del array.

map()

Crea un nuevo array transformando cada elemento.



for...of

Sintaxis moderna que recorre directamente los valores.

forEach()

Método que ejecuta una función para cada elemento.

```
// Ejemplo con forEach
const numeros = [1, 2, 3, 4];
numeros.forEach(num => console.log(num * 2));
```

El Bucle For en JavaScript

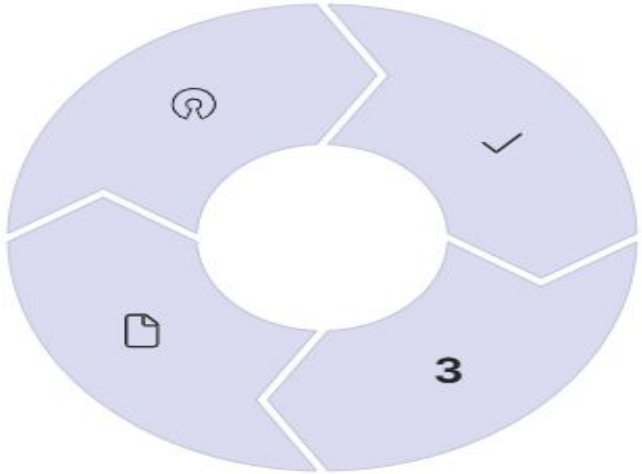
El bucle for es una estructura de control fundamental para iterar sobre colecciones de datos en JavaScript.

Inicialización

Se ejecuta una sola vez al inicio. Define el contador: **let i = 0**

Condición

Se evalúa antes de cada iteración: **i < array.length**



Incremento

Se ejecuta después de cada iteración: **i++**

Bloque de Código

Se ejecuta si la condición es verdadera: **console.log(array[i])**

```
// Ejemplo de bucle for
const frutas = ["manzana", "pera", "naranja"];
for (let i = 0; i < frutas.length; i++) {
  console.log(frutas[i]);
}
```

For Loop Vs. While Loop

For Loop

Todo en una línea: inicialización, condición e incremento.

Ideal para iteraciones con un número conocido de repeticiones.

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

Perfecto para recorrer arrays con índices.

While Loop

Solo evalúa una condición antes de cada iteración.

Mejor para casos donde no sabemos cuántas repeticiones necesitamos.

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

Útil para situaciones dependientes de eventos externos.

Casos de Uso

- For: arrays, strings, operaciones contadas
- While: lectura de archivos, operaciones hasta cumplir condición
- Do...While: ejecución garantizada al menos una vez

El Método forEach() en JavaScript

Ejecución por elemento

forEach() ejecuta una función callback una vez por cada elemento del array.

Parámetros del callback

La función recibe el valor actual, el índice y el array original.

Sin retorno

A diferencia de map(), forEach() siempre devuelve undefined.

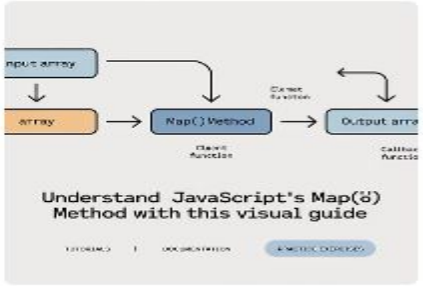
Mutación opcional

Puede modificar el array original si manipulamos sus elementos dentro del callback.

```
// Ejemplo práctico
const frutas = ["manzana", "plátano", "naranja"];
frutas.forEach((fruta, indice) => {
  console.log(`Fruta ${indice + 1}: ${fruta}`);
});
```

El Método map() en JavaScript

El método **map()** transforma cada elemento de un array y devuelve un nuevo array con los resultados.



Transformación

Crea un nuevo array sin modificar el original. Perfecto para transformaciones de datos.

Parámetros

La función callback recibe: valor actual, índice y el array completo.

Retorno

Siempre devuelve un array con la misma longitud que el original.

```
// Ejemplo: duplicar números
const numeros = [1, 2, 3, 4];
const duplicados = numeros.map(numero => numero * 2);
// duplicados = [2, 4, 6, 8]
```

Diferencia entre Map y ForEach

El método map()

Transforma cada elemento y **crea un nuevo array** con los resultados.

- Devuelve un nuevo array
- No modifica el original
- Ideal para transformaciones

Perfecto cuando necesitas los datos transformados para operaciones posteriores.

El método forEach()

Ejecuta una función para cada elemento **sin crear un nuevo array**.

- Siempre devuelve undefined
- Puede modificar el original
- Ideal para operaciones secundarias

Útil cuando solo necesitas ejecutar código para cada elemento sin retornar datos.

¿Cuándo usar cada uno?

- map(): cuando necesitas el resultado
- forEach(): para efectos secundarios
- map() es encadenable con otros métodos
- forEach() es más simple para iteraciones básicas

El Método reduce() en JavaScript

El método **reduce()** combina todos los elementos de un array en un único valor utilizando una función acumuladora.

Acumulación

Procesa los elementos secuencialmente y mantiene un acumulador con los resultados parciales.

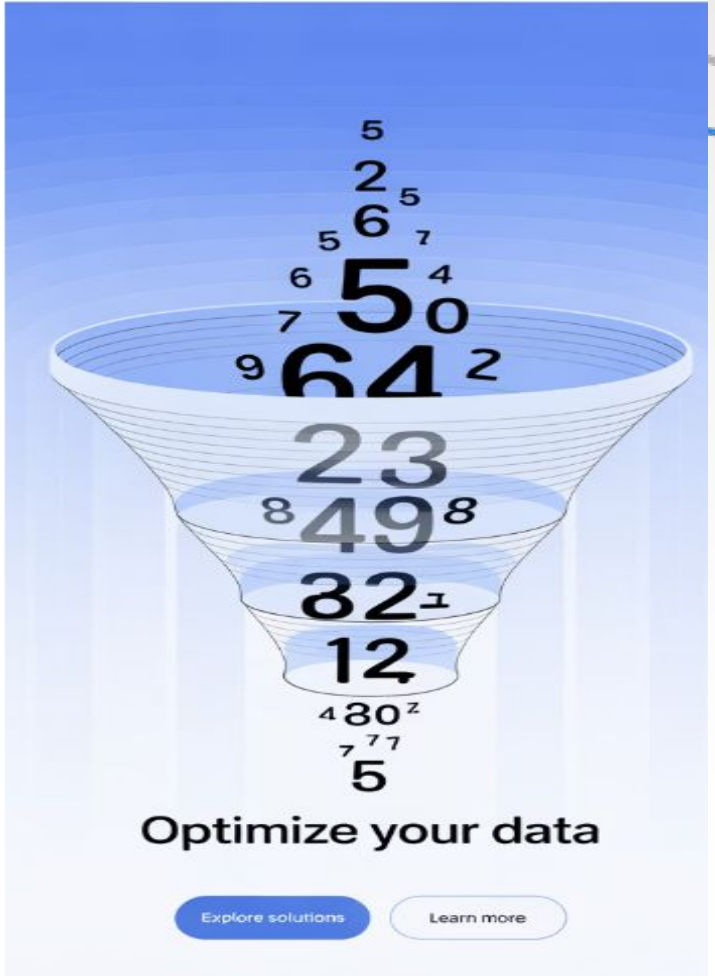
Flexibilidad

Puede producir cualquier tipo de resultado: número, string, objeto o array.

3 Parámetros

La función callback recibe: acumulador, valor actual, índice y array.

```
// Ejemplo: sumar números
const numeros = [1, 2, 3, 4];
const suma = numeros.reduce((acumulador, numero) =>
acumulador + numero, 0);
// suma = 10
```



Muchas gracias.

streambe GENERACIÓN T

Nuestras Redes

www.generaciont.org
www.streambe.com
www.instagram.com/generaciont_ar
www.tiktok.com/@generaciont
generaciont@generaciont.org
Cel: 11 61331747

GENERACIÓN T

streambe