cedia | TMO Training Management Office

info@cedia.org.ec
cedia.edu.ec

CURSO DE JAVASCRIPT

Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia

Ing. María Fernanda Granda J., PhD

26-11-2025

"Conectando ideas, transformando sociedades"

Módulo 5: Arreglos en JavaScript

- Más ejemplos de map, filter y reduce
- Funciones de los arreglos: some, find, findIndex, every, at,slice, spice
- Copiar y clonar arreglos
- Desestructuración de arreglos en JS
- Como usar la sintáxis Spread y el resto de parámetros en JS

Más ejemplos de map, filter, reduce

```
let estudiantes = [
    {   'id': '001',
        'f_nombre': 'Alex',
        'l_nombre': 'B',
        'genero': 'M',
        'casado': false,
        'edad': 22,
        'paga': 250,
        'cursos': ['JavaScript', 'React']
    },
    {   'id': '002',
        'f_nombre': 'Efraín',
        'l_nombre': 'M',
        'genero': 'M',
        'casado': true,
        'edad': 32,
        'paga': 150,
        'cursos': ['JavaScript', 'PWA']
    },
    {   'id': '003',
        'f_nombre': 'Noralma',
        'l_nombre': 'S',
        'genero': 'F',
        'casado': false,
        'edad': 27,
        'paga': 350,
        'cursos': ['Blogging', 'React', 'UX']
    },
    {
        'id': '004',
        'f_nombre': 'Hugolino',
        'l_nombre': 'F',
        'genero': 'M',
        'casado': true,
        'edad': 36,
        'paga': 250,
        'cursos': ['Git', 'React', 'Branding']
    }
];
```

Ejemplo con filter

- Encontrar los estudiantes con género femenino. Entonces la condición de filtro debería ser que el genero sea igual a 'F'.

```
let mujeres= estudiantes.filter(
    (elemento)=>{
        return elemento.genero=='F';
    }
);
console.log(mujeres);
```

```
[
  {
    id: '003',
    f_nombre: 'Noralma',
    l_nombre: 'S',
    genero: 'F',
    casado: false,
    edad: 27,
    pago: 350,
    cursos: [ 'Blogging', 'React', 'UX' ]
  }
]
```

Ejemplo con map

- Vamos a crear un nuevo arreglo de los nombres completos en mayúsculas de todos los elementos en el arreglo de estudiantes.

```
let nombresCompletos= estudiantes.map((elemento,indice) => {
    return {'Estudiante ':'('+indice+') '+elemento['f_nombre'].toUpperCase()+' '+elemento['l_nombre']}
});
console.log(nombresCompletos);
```

```
[
  { 'Estudiante ': '(0) ALEX B' },
  { 'Estudiante ': '(1) EFRAÍN M' },
  { 'Estudiante ': '(2) NORALMA S' },
  { 'Estudiante ': '(3) HUGOLINO F' }
]
```

Ejemplo con reduce

- Aplicar la función **reduce** en el arreglo **estudiantes** para calcular la cantidad total a pagar por todos los estudiantes.

- Iniciamos con el **acumulador con 0**.
- Aplicamos el método reduce en cada uno de los objetos estudiantes. Accedemos la propiedad **paga** y la añadimos al **acumulador**.
- Finalmente regresaremos el acumulador en **total**.

```
let total=estudiantes.reduce(
  (acumulador,estudiantes)=>{
    acumulador+=estudiantes.paga;
    return acumulador;
  },0);

  console.log(total);
```

Determinar si algún elemento cumple la condición

- **some():** Regresa un valor **booleano** (verdadero/falso) basado al menos en un elemento en el arreglo pasando la condición en la función.
- Vamos a ver si ahí hay algún estudiante menor de 30 años de edad.

```
let alguienMenorA30=estudiantes.some((elemento)=>{
    return elemento.edad<30;
});
console.log(alguienMenorA30); //true
```

Determinar el elemento que cumple la condición

- **find():** Regresa el primer elemento encontrado del arreglo que satisface la condición de la función. Si no encuentra regresa **Undefined**.

```
let alguienMenorA30=estudiantes.find((elemento)=>{
    return elemento.edad<30;
});
console.log(alguienMenorA30); //true
```

```
{
  id: '001',
  f_nombre: 'Alex',
  l_nombre: 'B',
  genero: 'M',
  casado: false,
  edad: 22,
  paga: 250,
  cursos: [ 'JavaScript', 'React' ]
}
```

Determinar el elemento que cumple la condición

- **findIndex():** Regresa el primer elemento encontrado del arreglo que satisface la condición de la función. Si no encuentra regresa **-1**.

```
let posicion=estudiantes.findIndex((elemento)=>{
    return elemento.edad<30;
});
console.log(posicion); //true
```

Determinar si los elementos cumplen la condición

- **every():** detecta si cada elemento del arreglo satisface la condición pasada en la función.
- Vamos a encontrar si todos los estudiantes se han registrado en al menos dos cursos.

```
let min2Cursos=estudiantes.every((elemento)=>{
    return elemento.cursos.length>=2;
});
console.log(min2Cursos); //true
```

Acceder a posiciones negativas del Arreglo

- **at():** permite acceder a los elementos usando ambos índices positivo y negativo con un solo método.

```
const numeros = ['1', '2', '3', '4', '5', '6', '7', '8'];
console.log(numeros.at(0)); // 1
console.log(numeros.at(3)); // 4
console.log(numeros.at(-1)); // 8
console.log(numeros.at(-5)); // 4
console.log(numeros.at(-8)); // 1
console.log(numeros.at(10)); // undefined
```

Copiar arreglos

- **slice():** Crea una nueva copia superficial del arreglo original. No cambia el arreglo original.

```
let elementos= new Array(1,2,3,4);
let copia=elementos.slice();
console.log(elementos);
console.log(copia);
console.log(elementos===copia);
```

```
[ 1, 2, 3, 4 ]
[ 1, 2, 3, 4 ]
false
```

Añadir, actualizar y remover elementos en un arreglo

- **splice():** Para añadir necesita pasar la posición donde se quiere añadir, cuántos elementos a borrar empezando con la posición, y con el elemento a añadir.

```
let nombres=['Ana','Nestor','Matías'];
nombres.splice(1,0,"Mariana");
console.log(nombres); // [ 'Ana', 'Mariana', 'Nestor', 'Matías' ]
```

Añadir, actualizar y remover elementos en un arreglo

- **splice():** En este ejemplo elimina el indice 2 y añade un nuevo elemento. En este caso devuelve un arreglo con el elemento eliminado.

```
let nombres=['Ana','Nestor','Matías'];
let eliminado=nombres.splice(2,1,"Mariana");
console.log(eliminado); //['Matías' ]
console.log(nombres); // [ 'Ana', 'Nestor', 'Mariana' ]
```

Acceder al última aparición en el arreglo

- **lastIndexOf():** Encuentra el índice de la última aparición de un elemento en un arreglo. Como indexOf(), lastIndexOf() también regresa -1 cuando el elemento no es encontrado.

```
let numeros=[2,4,6,8,2];
console.log("Pos del primer 2: "+numeros.indexOf(2));  // 0
console.log("Pos del último 2: "+numeros.lastIndexOf(2));  // 4
```

Llenar un arreglo

- **fill(valor):** llena un arreglo con un valor estático.
- Se puede cambiar todos los elementos a valores estáticos o cambiar algunos pocos elementos seleccionados.
- El método fill() cambia el arreglo original.

```
let numeros=[2,4,6,8,10];
numeros.fill(0);
console.log(numeros); // [ 0, 0, 0, 0, 0 ]
```

Llenar un arreglo

- Un ejemplo donde cambiamos desde la pos 1 hasta la pos 2 del arreglo usando el método fill()
  - El 1er argumento es el valor con el que cambiamos.
  - El 2do es el índice de inicio que se va a cambiar. Empieza con 0.
  - El último es para determinar donde dejar de llenar. El valor máximo podria ser numeros.length.

```
let numeros=[2,4,6,8,10];
numeros.fill(0,1,3);
console.log(numeros); // [ 2, 0, 0, 8, 10 ]
```

Método estático: Determinar si es un arreglo

- **Array.isArray(valor):** El método regresa verdadero si el valor que se le pasa es un arreglo.

```
console.log(Array.isArray(['?', '?', '?', '?', '?', '?', '?'])); // returns true
console.log(Array.isArray('?')); // returns false
console.log(Array.isArray({ 'tomate': '?'})); // returns false
console.log(Array.isArray([])); // returns true
```

Desestructuración de arreglos en JS

- Con ECMAScript 6 (ES6), se tiene una nueva sintaxis llamada sintaxis de desestructuración.
- Reduce de escribir un montón de código. Esto le da un gran impulso a la productividad.
- Es útil para ayudar a mantener tu código limpio y conciso.

Desestructuración de arreglos en JS

- Ejemplo de extraer valores de un arreglo usando la sintaxis de desestructuración:

Con desestructuración

```
let [num1, num2, num3]=[10, 20, 30];
console.log(num1, num2, num3);
```

```
10 20 30
```

Normal

```
let numeros=[10, 20, 30];
let num1=numeros[0];
let num2=numeros[1];
let num3=numeros[2];
console.log(num1, num2, num3);
```

Asignar un valor por defecto a una variable

- Cuando no hay valor o es undefined para el elemento arreglo.
- Asignamos por defecto el valor de la variable **y.**

```
let [x, y=5]=['?'];
console.log(x, y);
```

```
? 5
```

Como Saltar el Valor en un Arreglo

- Cuando no interesa todos los elementos de un arreglo.
- En este caso, se puede saltar un valor.

```
let [x, ,z]=[5,'?',8];
console.log(x, z);
```

```
5 8
```

Desestructurando un Arreglo Anidado en JS

- Los arreglos se pueden anidar. Esto significa que un arreglo puede tener otro arreglo como elemento.
- Para acceder a los elementos se puede hacer:

```
let comida=['banana','fresa', 'mango',['zanahoria', 'brocoli', 'espinaca']];
let vegetales=comida[3];
let fruta=comida[2];

console.log("Vegetales: "+vegetales);
console.log("Fruta : "+fruta);
```

```
Vegetales: zanahoria,brocoli,espinaca
Fruta : mango
```

Desestructurando un Arreglo Anidado en JS

- También se puede acceder a un elemento directo:

```
let comida=['banana','fresa', 'mango',['zanahoria', 'brocoli', 'espinaca']];
console.log("Vegetal: "+comida[3][2]);
```

```
Vegetal: espinaca
```

- O se puede hacer esto también:

```
let comida=[,,,[,, 'espinaca']];
console.log("Vegetal: "+comida[3][2]);
```

Como usar la sintáxis Spread y el resto de parámetros en JS

- Desde ES6, podemos usar el ... (si, tres puntos consecutivos) como sintaxis spread y el resto de los parámetros en la desestructuración de arreglo.
- Por el resto del parámetro, el ... aparece en el lado izquierdo de desestructuración.
- Para la sintaxis spread, el ... aparece en el lado derecho de la desestructuración.

Como usar el resto del parámetro en JS

- Podemos organizar los elementos de la izquierda de un arreglo en un nuevo arreglo.
- El resto de los parámetros deben ser la última variable en la sintaxis de desestructuración.

Como usar el resto del parámetro en JS

- Ejemplo, tenemos organizados los dos primeros elementos de un arreglo a las variables **x** y **y**.
- El resto son organizados a la variable **rest** usando el **....**
- La variable **rest** es un nuevo arreglo conteniendo los elementos sobrantes.

```
let [x, y, ...rest] = [3, 6, '?', '?', '?', '?', '?'];

console.log(x); // 3
console.log(y); // 6
console.log(rest); // ["?", "?", "?", "?", "?"]
```

Como usar el operador Spread en JS

- Podemos crear un **clon/copia** del arreglo existente como este:

```
let numeros = [3, 6, 10, 15, 39, 56, 87];
let numerosClonados=[... numeros];
console.log(numerosClonados); // [3, 6, 10, 15, 39, 56, 87]
console.log(numeros===numerosClonados); //false
```

Como combinar dos arreglos

- Usando la sintaxis spread.

```
let pares=[2,4,6,8,10];
let impares=[1,3,5,7,9];
let numeros=[... impares, ... pares];
console.log(numeros);  // [1,3,5,7,9,2,4,6,8,10]
```

cedia | TMO Training Management Office

GRACIAS

cedia.edu.ec
info@cedia.org.ec
(+593) 7-407-9300