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

1-12-2025

“Conectando ideas, transformando sociedades”

Módulo 6: Estructuras de Control

- Estructuras condicionales.
- Estructuras de bucle.
- Estructuras manipuladoras de objetos

Practicar

- Tenemos el siguiente arreglo de productos en una tienda:

```
let productos = [
  { nombre: "Laptop", precio: 1000, stock: 5, categoria: "tecnología" },
  { nombre: "Mouse", precio: 25, stock: 50, categoria: "accesorios" },
  { nombre: "Teclado", precio: 45, stock: 30, categoria: "accesorios" },
  { nombre: "Celular", precio: 800, stock: 10, categoria: "tecnología" }
];
```

Practicar

1. Filtrar los productos que tienen un stock menor a 20.
2. Crear un nuevo arreglo con el nombre en mayúsculas de todos los productos (map()).
3. Crear un nuevo arreglo con el nombre en minúsculas de todos los productos (map()).
4. Calcular el valor total del inventario (precio * stock por producto, luego suma todo con reduce).
5. Determinar si algún producto cuesta menos de 30 dólares (some()).
6. Usar find() para localizar el primer producto en la categoría "tecnología".
7. Verificar si todos los productos tienen stock mayor a 0 (every()).
8. Eliminar un producto del arreglo usando splice() (por ejemplo, el del índice 2).
9. Usar fill() para crear un arreglo de 4 posiciones con el texto "sin stock".
10. Comprobar con Array.isArray() si productos es un arreglo.

Practicar

11. Agrega un producto llamado "Tablet" con precio 600, stock 15, categoría "tecnología“
12. Calcular el producto más caro.
13. Calcular el producto más económico.
14. Listar los nombres de los productos con más de 10 unidades en stock.
15. Contar cuántos productos hay por categoría
16. Ordenar los productos de menor a mayor precio
17. Aplicar un descuento del 10% a todos los productos
18. Agrupar los productos por categoría en un objeto
19. Crear un arreglo con solo los nombres y precios
20. Buscar el índice del producto llamado "Mouse“

Practicar

21. Crear un nuevo array con productos agotados (stock === 0)
22. Calcular el promedio de precios de todos los productos
23. Mostrar un listado numerado con nombre y precio
24. Eliminar todos los productos de una categoría específica
25. Crear una función que calcule el stock total disponible
26. Crear una función que devuelva los nombres de los productos ordenados alfabéticamente
27. Contar cuántos productos tienen un precio superior a un valor dado
28. Simular una búsqueda por nombre exacto (ignorando mayúsculas/minúsculas)
29. Filtrar productos por una categoría dada (función dinámica)
30. Aplicar desestructuración para obtener los dos primeros productos y almacenar el resto en otra variable.

Condicional if

- A veces necesitamos que, bajo condiciones diferentes, se ejecuten acciones diferentes.
- Para esto podemos usar la sentencia if – else if - else

```
if (condición) {
...
} else if (condición) {
...
} else {
...
}
```

Operadores Lógicos

- En las condiciones se pueden concatenar varias condiciones con los operadores lógicos

Operador	Uso	Descripción
AND Lógico (`&&`)	`expr1 && expr2`	Devuelve `expr1` si se puede convertir a `false`; de lo contrario, devuelve `expr2`. Por lo tanto, cuando se usa con valores booleanos, `&&` devuelve `true` si ambos operandos son `true`; de lo contrario, devuelve `false`.
OR lógico (`||`)	`expr1 || expr2`	Devuelve `expr1` si se puede convertir a `true`; de lo contrario, devuelve `expr2`. Por lo tanto, cuando se usa con valores booleanos, `||` devuelve `true` si alguno de los operandos es `true`; si ambos son falsos, devuelve `false`.
NOT lógico (`!`)	`!expr`	Devuelve `false` si su único operando se puede convertir a `true`; de lo contrario, devuelve `true`.

Valores falsos

- Cualquier valor que no esté definido como falso se considerará verdadero en JavaScript.
- Ejemplos de expresiones que se pueden convertir a false son aquellos que se evalúan como null, 0, NaN, la cadena vacía ("") o undefined.
- Aquí hay una lista de valores falsos :
  - false
  - 0 (cero)
  - -0 (cero negativo)
  - 0n (BigInt cero)
  - "", '', `` (cadena vacía)
  - Null (nulo)
  - Undefined (no definido)
  - NaN (no es un número)

Evaluación de Corto Circuito

- Debido a que las expresiones lógicas se evalúan de izquierda a derecha, se prueban para una posible evaluación de "cortocircuito" utilizando las siguientes reglas:
  - false && anything se evalúa en cortocircuito como false.
  - true || anything se evalúa en cortocircuito como true.
- La parte anything de las expresiones anteriores no se evalúa, por lo que los efectos secundarios de hacerlo no surten efecto.

Ejemplos

- Algunos ejemplos del operador AND.

```
var a1 = true && true; // t && t devuelve true
var a2 = true && false; // t && f devuelve false
var a3 = false && true; // f && t devuelve false
var a4 = false && 3 == 4; // f && f devuelve false
var a5 = "Cat" && "Dog"; // t && t devuelve Dog
var a6 = false && "Cat"; // f && t devuelve false
var a7 = "Cat" && false; // t && f devuelve false
```

Ejemplos

- Algunos ejemplos del operador OR.

```
var o1 = true || true; // t || t devuelve true
var o2 = false || true; // f || t devuelve true
var o3 = true || false; // t || f devuelve true
var o4 = false || 3 == 4; // f || f devuelve false
var o5 = "Cat" || "Dog"; // t || t devuelve Cat
var o6 = false || "Cat"; // f || t devuelve Cat
var o7 = "Cat" || false; // t || f devuelve Cat
```

Ejemplos

- Algunos ejemplos del operador NOT.

```
var n1 = !true; // !t devuelve false
var n2 = !false; // !f devuelve true
var n3 = !"Cat"; // !t devuelve false
```

Operador Ternario o Condicional

- Es posible ejecutar en una sola instrucción un if con un else. Si estas empezando no te recomiendo usarla, pero no la olvides.
- Posee una particularidad muy interesante: devuelve el valor.

```
const edad = 32;
let ciudadano;

if (edad >= 18) {
    ciudadano = "Puede votar";
} else {
    ciudadano = "No puede votar";
}

console.log(ciudadano); //Puede votar
```

```
const edad = 32;
const ciudadano = edad >= 18 ? "Puede votar" : "No puede votar";
console.log(ciudadano);
```

```
switch (variable) {
    case 0:
        ...
        break;
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    default:
        ...
        break;
}
```

Switch

- Se comporta como un condición cuyo valor es igualado en todos los casos.
- Las sentencias switch pueden tener una sintaxis más limpia que las sentencias if else complicadas.

```
const diaSemana = "martes";
switch (diaSemana) {
  case "lunes":
    console.log("Hoy es lunes.");
    break;
  case "martes":
    console.log("Hoy es martes.");
    break;
  case "miercoles":
    console.log("Hoy es miércoles.");
    break;
  case "jueves":
    console.log("Hoy es jueves.");
    break;
  case "viernes":
    console.log("Hoy es viernes.");
    break;
  case "sabado":
    console.log("Hoy es sábado.");
    break;
  case "domingo":
    console.log("Hoy es domingo.");
    break;
  default:
    console.log("No es un día de la semana válido.");
}
```

Operador de Coalescencia Nulo

- En ciertos momentos encontraremos valores nulos.
- Para evitar problemas, lo idóneo, es dar un valor por defecto.
- Dentro de JavaScript disponemos de una herramienta que devuelve el lado derecho si el izquierdo es null o undefined.

```
const variable1 = "Agua" || "Alternativa"
// "Agua"

const variable2 = null || "Alternativa"
// "Alternativa"
```

Operador de Coalescencia Nulo

- Actualmente se recomiendo utilizar ??, una evolución de ||, ya que devuelve en ciertas circunstancias falsos negativos con "" o 0.
- Ejemplo: todo lo presente en el siguiente código será retornado como Alternativa:

```
0 || "Alternativa"
"" || "Alternativa"
false || "Alternativa"
undefined || "Alternativa"
null || "Alternativa"
```

Operador de Coalescencia Nulo

- Actualmente se recomiendo utilizar ??, una evolución de ||, ya que devuelve en ciertas circunstancias falsos negativos con "" o 0.
- Ejemplo: todo lo presente en el siguiente código de la izquierda será retornado como Alternativa.

```
0 || "Alternativa"
"" || "Alternativa"
false || "Alternativa"
undefined || "Alternativa"
null || "Alternativa"
```

```
0 ?? "Alternativa" // 0
"" ?? "Alternativa" // ""
false ?? "Alternativa" // false
undefined ?? "Alternativa" // "Alternativa"
null ?? "Alternativa" // "Alternativa"
```

Encadenamiento Opcional

- Si intento acceder a un valor que no existe, obtendremos **undefined**.

```
const perfil = {
    nombre: "Miguel",
    edad: 45,
    activo: true,
    direccion: {
        calle: "falsa",
        numero: 123
    }
};

perfil.edad // 45
perfil.nombre // "Miguel"
perfil.apellidos // undefined
```

- No hay ningún impedimento si trabajamos con un solo nivel de profundidad.
- Pero cuando queremos obtener un valor a más profundidad, y no existe, nos devolverá un **error** que **parará la ejecución**.

```
perfil.direccion.calle // "falsa"
perfil.comentarios.nombre // Uncaught TypeError
```

Encadenamiento Opcional

- Podemos **capturar el error** y procesarlo o dar un undefined.
- Para lograrlo podremos utilizar un interrogante para marcarlo como opcional.

```
perfil.direccion.calle // "falsa"
perfil.comentarios?.nombre // undefined
```

- Y, si lo mezclamos con un Operador de coalescencia (??), podemos incluso obtener un valor por defecto.

```
perfil.direccion.calle ?? "Sin calle" // "falsa"
perfil.comentarios?.nombre ?? "Sin comentarios" // "Sin comentarios"
```

Bucle de iteración de valor: For-of

- De una manera sencilla y amena recorreremos un array, o lista, para obtener su contenido

```
for (variable of array/objeto) { ... }
```

- Ejemplo:

```
let listaFrameworks = ['Angular', 'React', 'Vue', 'Ember', 'Elm'];

for (const framework of listaFrameworks) {
    console.log(framework);
}
//Angular
//React
//Vue
//Ember
//Elm
```

Bucle de iteración de clave: For-in

- La diferencia radica en la palabra **in** en lugar de **of**, por el resto la estructura es igual

```
for (variable in array/objeto) { ... }
```

- Su objetivo será la de proporcionarnos las posiciones que ocupa cada elemento. Un ejemplo.

```
let listaFrameworks = ['Angular', 'React', 'Vue', 'Ember', 'Elm'];

for (const posicion in listaFrameworks) {
    console.log(posicion);
}
//0
//1
//2
//3
//4
```

Bucle de iteración de clave y valor: forEach

- Permite tener lo mejor del for-in y for-of. Su estructura difiere a lo anteriormente mencionado.

```
array.forEach(function callback(valor, posicion, arrayUsado) { ... });
```

- Devuelve la posición y el valor de cada elemento del array.

```
let listaFrameworks = ['Angular', 'React', 'Vue', 'Ember', 'Elm'];
listaFrameworks.forEach((valor, posicion, array) => {
    console.log(posicion + '-' + valor);
});
//0-Angular
//1-React
//2-Vue
//3-Ember
//4-Elm
```

Bucle de iteración de clave y valor: forEach

- El argumento array no es más una copia del que estamos recorriendo.
- Ejemplo para ver que ocurre cuando cambiamos lo que se muestra.

```
let listaFrameworks = ['Angular', 'React', 'Vue', 'Ember', 'Elm'];
listaFrameworks.forEach((valor, posicion, array) => {
    console.log(array);
});
//['Angular', 'React', 'Vue', 'Ember', 'Elm']
//['Angular', 'React', 'Vue', 'Ember', 'Elm']
//['Angular', 'React', 'Vue', 'Ember', 'Elm']
//['Angular', 'React', 'Vue', 'Ember', 'Elm']
//['Angular', 'React', 'Vue', 'Ember', 'Elm']
```

Bucle tradicional con For

- Sin depender de un array podemos recorrer un rango de números a nuestra elección.

```
for ([expresionInicial]; [condicion]; [expresionIncremento])
  sentencia
```

- Ejemplo:

```
// Bucle for tradicional
for (let i = 0; i <= 3; i += 1) {
    console.log(i);
}
//0
//1
//2
//3
```

```
// Decrecer del 10 al 0
for (let i = 10; i > 0; i -= 1) {
    console.log(i);
}
// Intervalo de 5 entre el 0 a 100
for (let i = 0; i <= 100; i += 5) {
    console.log(i);
}
```

Bucle tradicional con While

- Este bucle es potente a la par que peligroso.
- Solo admite un condicional, por lo que si nuestro software no llega a cambiar esa variable tendremos un bucle infinito.
- Ejemplo:

```
let num = 30;
while (num <= 50) {
    num += 1;
    console.log(num);
}
```

cedia | TMO Training Management Office

GRACIAS

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