REPASO JAVASCRIPT 24-25

Fortunato Yekue Nzambi

September 2024

Contents

1 Let vs Var vs Const

Es importante dejar de usar `var` porque asigna un alcance global a las variables, lo que puede causar problemas. En su lugar:

- Usa `const` para variables cuyo valor no cambiará.
- Usa `let` si planeas modificar el valor de la variable.

1.1 Ejemplos:

```
const PI = 3.14;
// PI no se puede modificar

let nombre = 'Juan';
nombre = 'Pedro'; // nombre ahora es 'Pedro'

var edad = 25;
if (true) {
    var edad = 30;
}
console.log(edad); // 30 (porque var es global)
```

2 Template Literals

Los *template literals* permiten trabajar con cadenas de texto de forma dinámica, como concatenar variables dentro de strings.

2.1 Ejemplos:

```
// Concatenar variables
const nombre = 'Juan';
console.log(`Hola, ${nombre}!`);

// Operaciones dentro de template literals
console.log(`2 + 2 es ${2 + 2}`);

// Multilinea con template literals
const mensaje = `Hola,
 Como estas?`;
console.log(mensaje);
```

2.2 Ejercicio: Uso de Template Literals con Formularios

Usando *template literals*, escribe el código para que una página web pida en primera instancia un nombre y una edad mediante un formulario. Al pulsar un botón, se debe ocultar el formulario y mostrar un mensaje como el siguiente: "Hola [nombre del usuario], naciste en el año [año de nacimiento]". Además, debe aparecer debajo del mensaje otro botón para recargar la página y poder empezar de nuevo.

3 Object Literals

Un objeto literal es una variable cuyo valor está encerrado entre llaves. Los objetos en JavaScript se pasan por referencia, lo que significa que cualquier modificación en el objeto original afectará a todas las copias del mismo.

3.1 Ejemplos:

```
// Creacion de un objeto literal
const persona = { nombre: 'Juan', edad: 25 };

// Acceso a propiedades
console.log(persona.nombre); // Juan

// Uso de spread para copiar un objeto
const persona2 = { ...persona, edad: 30 };
console.log(persona2); // { nombre: 'Juan', edad: 30 }
```

4 Funciones Flecha

Las funciones flecha son una forma moderna y simplificada de declarar funciones en JavaScript.

4.1 Ejemplos:

```
// Funcion flecha basica
const sumar = (a, b) => a + b;
console.log(sumar(2, 3)); // 5

// Funcion flecha con una sola linea
const saludo = nombre => `Hola, ${nombre}!`;
console.log(saludo('Juan'));

// Funcion flecha con multiples lineas
const multiplicar = (a, b) => {
```

```
  const resultado = a * b;
  return resultado;
};
console.log(multiplicar(2, 3)); // 6
```

5 Find y Filter

`find()` devuelve el primer elemento que cumple una condición, mientras que `filter()` devuelve todos los elementos que la cumplen.

5.1 Ejemplos:

```
// Uso de find()
const numeros = [1, 2, 3, 4, 5];
const encontrado = numeros.find(num => num > 3);
console.log(encontrado); // 4

// Uso de filter()
const mayoresDeDos = numeros.filter(num => num > 2);
console.log(mayoresDeDos); // [3, 4, 5]

// Uso de find() con objetos
const personas = [{nombre: 'Juan'}, {nombre: 'Ana'}];
const personaEncontrada = personas.find(persona => persona.nombre === 'Ana');
console.log(personaEncontrada); // {nombre: 'Ana'}
```

5.2 Ejercicio: Sistema de Gestión de Empleados

Crea una aplicación web que permita gestionar una lista de empleados en una empresa.

- Crea una estructura HTML que incluya un formulario para agregar nuevos empleados y una lista de empleados existentes. El formulario debe incluir campos para ingresar el nombre, el salario y el departamento del empleado.
- Diseña una interfaz atractiva para la aplicación. Utiliza CSS para dar estilo al formulario y a la lista de empleados. Asegúrate de que la interfaz sea intuitiva y fácil de usar.
- Implementa la funcionalidad principal utilizando objetos en JavaScript. Debes crear un objeto "Empleado" que tenga propiedades para el nombre, el salario y el departamento. Utiliza un array para almacenar todos los empleados.

Implementa las siguientes funciones en JavaScript:

- **Agregar Empleado**: Cuando el usuario complete el formulario y haga clic en el botón "Agregar", crea un nuevo objeto *Empleado* con los datos ingresados y agrégalo al array de empleados. Actualiza la lista de empleados en la interfaz para mostrar el nuevo empleado.
- **Calcular Salario Total**: Calcula y muestra el salario total de todos los empleados en la parte inferior de la lista.
- **Buscar por nombre**: Utilizando el buscador, el usuario introduce un nombre y se muestra el empleado si lo encuentra.

6 Desestructuración de Objetos y Arrays

La desestructuración permite extraer propiedades o elementos de objetos y arrays en variables separadas.

6.1 Ejemplos:

```
// Desestructuracion de objetos
const persona = { nombre: 'Juan', edad: 25 };
const { nombre, edad } = persona;
console.log(nombre, edad); // Juan 25

// Desestructuracion de arrays
const numeros = [1, 2, 3];
const [uno, dos] = numeros;
console.log(uno, dos); // 1 2

// Desestructuracion de retorno de funciones
function retornaArray() {
  return [1, 2, 3];
}
const [a, b] = retornaArray();
console.log(a, b); // 1 2
```

7 Importaciones y Exportaciones

Para trabajar con módulos en JavaScript, es fundamental entender cómo importar y exportar contenido entre archivos JS.

7.1 Ejemplos:

```
// En archivo funciones.js
export const sumar = (a, b) => a + b;
```

```
export const restar = (a, b) => a - b;

// En archivo principal.js
import { sumar, restar } from './funciones';
console.log(sumar(2, 3)); // 5
console.log(restar(5, 2)); // 3

// Exportacion por defecto
export default function multiplicar(a, b) {
  return a * b;
}
```

7.2 Ejercicio: Sistema de Manejo de Inventarios

Supongamos que estás desarrollando un sistema de manejo de inventarios para una tienda en línea. Tienes tres módulos de JavaScript separados que deben trabajar juntos para lograr esto: `productos.js`, `carrito.js` y `facturacion.js`.

- El módulo `productos.js` debe exportar una lista de productos disponibles en la tienda. Cada producto debe tener una imagen, nombre, un precio y una cantidad en stock. Además, debe exportar una función `actualizarStock` que tome un producto y una cantidad, y actualice la cantidad en stock para ese producto.
- El módulo `carrito.js` debe exportar un objeto `carrito` que contiene los productos que un cliente ha agregado al carrito de compras. Debe tener métodos para agregar productos al carrito y calcular el total de la compra. Además, debe exportar una función `vaciarCarrito` para eliminar todos los productos del carrito.
- El módulo `facturacion.js` debe importar el módulo `carrito.js` y calcular el total de la compra, además de generar una factura para el cliente.

8 Promesas y Async/Await

Las promesas permiten manejar tareas asíncronas. `async/await` hace que el código asíncrono sea más legible.

8.1 Ejemplos:

```
// Creacion de una promesa
const promesa = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Hecho!'), 1000);
});
```

```
// Uso de then()
promesa.then(result => console.log(result)); // Hecho!

// Uso de async/await
async function obtenerDatos() {
  const resultado = await promesa;
  console.log(resultado); // Hecho!
}
obtenerDatos();
```

9 Fetch API

La función `fetch()` se utiliza para realizar peticiones HTTP.

9.1 Ejemplos:

```
// Peticion fetch basica
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data));

// Peticion con async/await
async function obtenerDatos() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  console.log(data);
}
obtenerDatos();

// Fetch con manejo de errores
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la respuesta');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

10 Axios

Axios es una biblioteca para realizar peticiones HTTP de manera más sencilla que con `fetch()`.

10.1 Ejemplos:

```
// Peticion basica con Axios
axios.get('https://api.example.com/data')
  .then(response => console.log(response.data));

// Uso de async/await con Axios
async function obtenerDatos() {
  const response = await axios.get('https://api.example.com/data');
  console.log(response.data);
}
obtenerDatos();

// Peticion POST con Axios
axios.post('https://api.example.com/submit', {
  nombre: 'Juan',
  edad: 25
}).then(response => console.log(response.data));
```

11 Ternarios y Null Check

El operador ternario es una forma compacta de escribir condiciones `if-else`.
Para manejar variables no definidas, usamos el operador OR (`||`).

11.1 Ejemplos:

```
// Operador ternario
const edad = 18;
const mensaje = edad >= 18 ? 'Es mayor de edad' : 'Es menor de edad';
console.log(mensaje); // Es mayor de edad

// Null Check
const nombre = undefined || 'Nombre por defecto';
console.log(nombre); // Nombre por defecto

// Encadenar ternarios
const nota = 75;
const resultado = nota >= 90 ? 'A' : nota >= 75 ? 'B' : 'C';
console.log(resultado); // B
```