
JS
const fetch =
const log =
let embed;
function
// Promise.
return
function
return prev.
$(':header').
const children
if ($(children).
$(header).
$(children).
}
return header;
});
return Promise.
});

JavaScript

JavaScript es un lenguaje de programación interpretado y orientado a objetos, ampliamente utilizado para crear interactividad en las páginas web.

Características principales:

- Interactividad
- Compatibilidad
- Multiplataforma
- Sintaxis
- Ecosistema amplio

JavaScript: Salida

- **Consola:** La forma más sencilla de mostrar información es a través de la consola del navegador utilizando **console.log()**. Esto es útil para depuración y desarrollo.
  - Además de console.log(), puedes usar **console.error()** para mostrar errores y **console.warn()** para mostrar advertencias en la consola.
- **Ventanas de alerta:** Se puede usar **alert()** para mostrar un cuadro de diálogo de alerta con un mensaje. Este método detiene la ejecución del código hasta que el usuario cierre la ventana.

JavaScript: Entrada

El método prompt() en JavaScript es una función que se utiliza para mostrar un cuadro de diálogo que solicita al usuario que ingrese información. Este método es parte de la API del navegador y permite la interacción básica con el usuario a través de la interfaz gráfica.

```
let valor = prompt([texto], [valorPorDefecto]);
```

JavaScript: Variables

Las variables son contenedores que permiten almacenar datos.

var Forma más antigua de declarar variables. Su visibilidad está limitada a la función donde se declara. Su uso ha disminuido debido a comportamientos inesperados en algunas situaciones.

let Permite declarar variables con un ámbito de bloque. Es más seguro que var.

const Se utiliza para declarar constantes, es decir, variables cuyo valor no puede cambiar una vez asignado. Al igual que **let**, tiene un **ámbito de bloque**.

JavaScript: Tipos de datos

JavaScript es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar el tipo de variable al momento de su creación. Los tipos de datos más comunes incluyen:

String Representa texto. Se pueden definir con comillas simples, dobles o backticks (template literals).

```
let saludo = "Hola, mundo";
let despedida = 'Adiós, mundo';
let mensaje = `Hoy es ${saludo}`;
```

JavaScript: Tipos de datos

Number Representa números, tanto enteros como decimales.

```
let edad = 30;       // Entero
let temperatura = 36.5; // Decimal
```

Boolean Representa valores lógicos, true (verdadero) o false (falso).

```
let esEstudiante = true;
let esMaestro = false;
```

JavaScript: Operadores

Los operadores son símbolos que se utilizan para realizar operaciones sobre valores o variables.

Operadores aritméticos:

Se utilizan para realizar cálculos matemáticos.

Operador	Descripción	Ejemplo
`+`	Suma	`5 + 3`
`-`	Resta	`5 - 3`
`*`	Multiplicación	`5 * 3`
`/`	División	`5 / 3`
`%`	Módulo (resto)	`5 % 3`

JavaScript: Operadores de comparación

Se utilizan para comparar dos valores.

Operador	Descripción	Ejemplo
`==`	Igualdad (valor)	`5 == '5'`
`===`	Igualdad estricta (valor y tipo)	`5 === 5`
`!=`	Desigualdad (valor)	`5 != '5'`
`!==`	Desigualdad estricta (valor y tipo)	`5 !== 5`
`>`	Mayor que	`5 > 3`
`<`	Menor que	`5 < 3`
`>=`	Mayor o igual que	`5 >= 5`
`<=`	Menor o igual que	`5 <= 3`

JavaScript: Operadores lógicos

Se utilizan para combinar expresiones booleanas.

Operador	Descripción	Ejemplo
`&&`	AND lógico	`true && false`
`		`
`!`	NOT lógico	`!true`

JavaScript: Operadores de asignación

Se utilizan para asignar valores a variables.

Operador	Descripción	Ejemplo
`=`	Asignación simple	`let x = 10;`
`+=`	Asignación con suma	`x += 5;`
`-=`	Asignación con resta	`x -= 2;`
`*=`	Asignación con multiplicación	`x *= 3;`
`/=`	Asignación con división	`x /= 4;`

JavaScript: Arreglos

Array Una colección de elementos, que pueden ser de cualquier tipo de dato, incluida otra colección. Se define utilizando corchetes [].

```
let frutas = ["manzana", "banana", "naranja"];
```

JavaScript: Objetos

Object Representa un conjunto de propiedades, donde cada propiedad tiene una clave (nombre) y un valor. Se define utilizando llaves {}.

```
let persona = {
    nombre: "Juan",
    edad: 30,
    esEstudiante: true
};
```

JavaScript: undefined

undefined es un valor que se asigna automáticamente a una variable que ha sido declarada, pero que no tiene un valor asignado. También se devuelve cuando una función no especifica un valor de retorno o cuando se intenta acceder a una propiedad de un objeto que no existe.

```
let x;  // x es undefined
console.log(x);  // undefined

function saludo() { }
console.log(saludo());  // undefined

let objeto = { nombre: "Juan" };
// undefined (propiedad no existente)
console.log(objeto.edad);
```

JavaScript: null

null es un valor explícitamente asignado para indicar "ausencia de valor". A diferencia de undefined, que se asigna automáticamente, null se usa intencionadamente para señalar que una variable no tiene un valor válido o está vacía.

```
let y = null;
console.log(y);  // null

let objeto2 = { nombre: "Ana", edad: null };
console.log(objeto2.edad);  // null
```

JavaScript: NaN

NaN significa Not-a-Number (No es un número) y es un valor especial en JavaScript que indica que una operación matemática o de conversión ha fallado y no ha producido un número válido. A pesar de lo que su nombre sugiere, el tipo de NaN en JavaScript es number.

- Operaciones Matemáticas No Definidas
- Conversión Fallida de String a Número
- Operaciones con undefined
- Operaciones con Valores No Numéricos

Number.isNaN() Verifica si el valor dado es realmente NaN.

JavaScript: Diferencias entre null y undefined

Característica	undefined	null
Asignación	Automática cuando no hay valor.	Asignado explícitamente por el programador.
Uso	Para variables no definidas o propiedades inexistentes.	Para indicar intencionadamente la ausencia de valor.
Tipo	undefined	object
Comparación estricta	No son iguales (undefined !== null).	Son diferentes (undefined !== null)
Comparación no estricta	Son consideradas iguales (undefined == null).	Se considera que ambas indican "sin valor".

JavaScript: Funciones

Una función puede ser declarada para ser llamada posteriormente. En JavaScript, las funciones son objetos de primera clase, lo que significa que:

- Se pueden asignar a variables.
- Se pueden pasar como argumentos a otras funciones.
- Se pueden devolver desde otras funciones.

```
function sumar(a, b) {
    return a + b;
}
```

JavaScript: Funciones

Las **funciones anónimas** no tienen un nombre, y suelen asignarse a variables o pasarse como argumentos a otras funciones.

```
const multiplicar = function (a, b) {
    return a * b;
};
```

Las **funciones flecha** son una forma más concisa de escribir funciones. No tienen su propio contexto this y son útiles para escribir funciones más breves.

```
const numeros = [1, 2, 3, 4, 5];
const dobles = numeros.map(x => x * 2);
console.log(dobles); // [2, 4, 6, 8, 10]
```

JavaScript: Parámetros de funciones

Los parámetros son variables que se pasan a una función. Estos se colocan dentro de los paréntesis y son separados por “,”.

Se pueden definir con un valor predeterminado para ser usados si no se pasa un valor específico al invocar la función.

Se debe tener en cuenta su posición. Si se desea omitir un parámetro, todos los que le siguen también deben tener valores predeterminados o ser omitidos.

```
function saludo(nombre = "Invitado") {
    return `Hola, ${nombre}!`;
}
```

JavaScript: Llamada de funciones

Para llamar una función se utiliza el nombre seguido de los paréntesis. De esta forma, se obtiene el resultado de la función.

```
function convertirACelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}

let valor = convertirACelsius();
```

Si se accede a una función sin los paréntesis, se retorna la función completa en lugar del resultado.

```
function convertirACelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}

let valor = convertirACelsius;
```

JavaScript: Métodos de Number

- **Number.isInteger(value)** Verifica si el valor es un número entero.
- **Number.isNaN(value)** Verifica si el valor es NaN.
- **Number.parseInt(string)** Convierte una cadena de texto a un número entero.
- **Number.parseFloat(string)** Convierte una cadena de texto a un número de punto flotante.
- **toFixed(digits)** Redondea un número a un número fijo de decimales.
- **toString()** Convierte el número a una cadena.

JavaScript: Métodos de String

- **charAt(index)** Devuelve el carácter en una posición específica.
- **concat(string)** Combina dos o más cadenas.
- **includes(substring)** Verifica si una cadena contiene otra cadena.
- **indexOf(substring)** Devuelve el índice de la primera aparición de una subcadena.
- **replace(substring, newSubstring)** Reemplaza una parte de la cadena por otra.
- **split(separator)** Divide una cadena en un arreglo de subcadenas.
- **toLowerCase(), toUpperCase()** Convierte todos los caracteres a minúsculas y mayúsculas.
- **trim()** Elmina los espacios en blanco al principio y al final.

JavaScript: Métodos de Array

- **push(element)** Agrega un elemento al final del arreglo.
- **pop()** Elimina y devuelve el último elemento del arreglo.
- **shift()** Elimina y devuelve el primer elemento del arreglo.
- **unshift(element)** Agrega un elemento al principio del arreglo.
- **slice(start, end)** Devuelve una copia de una porción del arreglo.
- **splice(start, deleteCount, item1, item2, ...)** Agrega o elimina elementos del arreglo.
- **forEach(callback)** Ejecuta una función para cada elemento del arreglo.

JavaScript: Métodos de Array

- **map(callback)** Crea un nuevo arreglo con los resultados de una función aplicada a cada elemento.
- **filter(callback)** Crea un nuevo arreglo con los elementos que pasen una prueba.
- **reduce(callback, initialValue)** Reduce todos los elementos del arreglo a un solo valor.
- **find(callback)** Devuelve el primer elemento que pase una prueba.
- **findIndex(callback)** Devuelve el índice del primer elemento que pase una prueba.

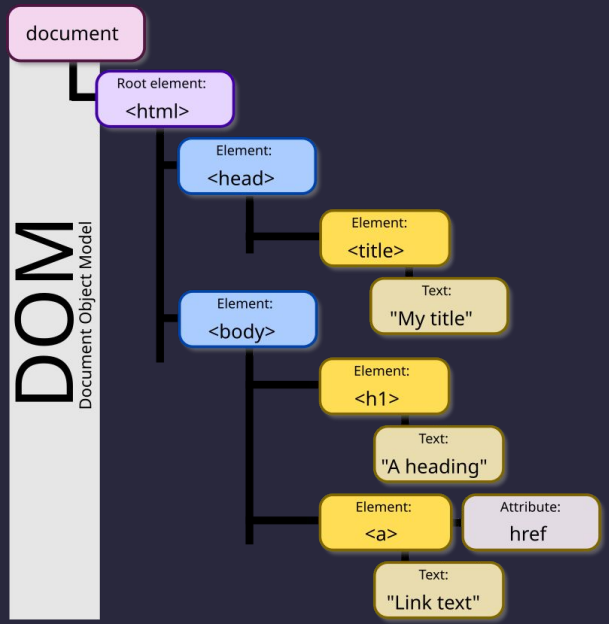
JavaScript: Métodos de Object

- **Object.keys(object)** Devuelve un arreglo con las llaves (propiedades) de un objeto.
- **Object.values(object)** Devuelve un arreglo con los valores de las propiedades del objeto.
- **Object.entries(object)** Devuelve un arreglo de pares [key, value] del objeto.
- **Object.assign(target, source)** Copia las propiedades de un objeto a otro.
- **Object.freeze(object)** Congela un objeto para que no se puedan modificar sus propiedades.
- **Object.seal(object)** Sella un objeto, permitiendo modificar propiedades existentes, pero no agregar o eliminar propiedades.

JavaScript: Manipulación del DOM

El DOM (Document Object Model) es una representación en forma de árbol de un documento HTML.

A través de JavaScript, se puede acceder y manipular este árbol, lo que permite cambiar el contenido, el estilo o la estructura de una página web sin necesidad de recargarla.



JavaScript: Seleccionar elementos

Se pueden seleccionar elementos por medio de los siguientes métodos:

- **document.getElementById()** Selecciona el elemento dependiendo de un id.
- **document.getElementsByClassName()** Selecciona los elementos que coincidan con una clase.
- **document.getElementsByTagName()** Selecciona los elementos que correspondan a una etiqueta.
- **document.querySelector()** Selecciona el primer elemento que coincida con el selector CSS dado.
- **document.querySelectorAll()** Igual que el anterior pero selecciona todos los que coinciden.

JavaScript: Modificar propiedades

Las propiedades son variables del objeto que se pueden leer y modificar directamente con JavaScript.

<input>

- value
- type
- placeholder
- disabled
- checked (para checkbox)

<img>

- src
- alt
- width
- height
- title

Otras propiedades comunes

- textContent
- innerHTML
- style
- id

```
const titulo = document.getElementById('titulo');
titulo.textContent = 'Nuevo texto';
```

JavaScript: Modificar estilos

Se pueden modificar las propiedades CSS de un elemento. Una vez que se seleccionó se accede a la propiedad style y posteriormente a la propiedad de CSS que se quiere modificar.

```
const titulo = document.getElementById('titulo');
titulo.style.color = '#ffffff';
```

Si se quieren asignar varios estilos

```
const contenedorP = document.querySelector('.container p');
const nuevosEstilos = {
    fontStyle :  'italic',
    background :  'salmon',
    color :  'white'
};
Object.assign(contenedorP.style, nuevosEstilos);
```

JavaScript: Modificar clases

- **classList.add()**: Agrega una o más clases al elemento.
- **classList.remove()** Elimina una o más clases del elemento.
- **classList.toggle()** Alterna (añade o elimina) una clase, dependiendo de si está presente o no.
- **classList.contains()** Verifica si un elemento tiene una clase específica.
- **classList.replace()** Reemplaza una clase por otra.

JavaScript: Evento de click

Para manejar evento de click se utiliza **addEventListener('click', function)**. Dentro de la función se puede especificar lo que queremos que suceda cuando se hace click en un elemento.

```
const boton = document.getElementById('miBoton');
boton.addEventListener('click', function () {
    alert('Se hizo click');
});
```

JavaScript: Agregar elementos al DOM

Para agregar un elemento, primero debe ser creado con **document.createElement('etiqueta')**. Después añadir contenido al elemento usando textContent y añadir atributos o propiedades según se necesite.

Después, se puede usar:

- **appendChild()** Añade el elemento como el último hijo de otro elemento.
- **insertBefore()** Inserta el elemento antes de otro nodo especificado.
- **prepend()** Inserta el nuevo elemento como el primer hijo.
- **insertAdjacentHTML()** Inserta texto HTML en una posición específica relativa a un nodo.

JavaScript: Quitar elementos del DOM

Primero se selecciona el elemento con document.getElement…

Después se puede usar:

- **remove()** Elimina el elemento directamente.
- **removeChild()** Elimina un hijo de un elemento padre. Es necesario seleccionar primero al padre.

JavaScript: Evento de teclado

Existen 3 eventos de teclado:

- **keydown**: Se dispara cuando una tecla es presionada.
- **keypress**: Se dispara mientras una tecla es presionada y sostenida (en desuso en muchos navegadores).
- **keyup**: Se dispara cuando una tecla es soltada.

```
document.addEventListener('keydown', () => {
    console.log('Se presionó una tecla');
});
```

JavaScript: Evento de teclado

Propiedades clave de los eventos:

- **event.key** Devuelve la tecla presionada como un string (e.g., 'a', 'Enter').
- **event.code** Devuelve el código físico de la tecla presionada (e.g., 'KeyA', 'ArrowDown').
- **event.altKey, event.ctrlKey, event.shiftKey, event.metaKey** Indican si alguna de estas teclas modificadoras estaba presionada durante el evento (Alt, Ctrl, Shift, Meta).

```
document.addEventListener('keydown', function (event) {
    if (event.altKey && event.key === 's') {
        console.log('Alt + S presionado');
    }
});
```

JavaScript: preventDefault()

Se utiliza para evitar que el comportamiento predeterminado de un evento ocurra. Esto es particularmente útil cuando se desea tomar control sobre el flujo de interacción en una página web, ya que permite modificar la respuesta estándar que el navegador tiene ante ciertos eventos como:

- Prevenir el envío de un formulario
- Prevenir el comportamiento de enlaces
- Prevenir el comportamiento de teclas

```
document.addEventListener('keydown', function (event) {
    if (event.ctrlKey && event.key === 's') {
        event.preventDefault(); //Evita que se guarde la página
        console.log('Ctrl + S presionado');
    }
});
```

JavaScript: API de Local Storage

- Guardar datos: **localStorage.setItem(‘clave’,’valor’)**
- Obtener datos: **localStorage.getItem(‘clave’)** Si no existe la clave, devuelve null.
- Eliminar datos: **localStorage.removeItem('clave')**
- Limpiar todos los datos: **localStorage.clear()**
- Longitud del almacenamiento: **localStorage.length**
- Obtener una clave por índice: **localStorage.key(index)**

Local Storage solo permite almacenar datos en formato de cadena (string). Para almacenar objetos o arrays, es necesario convertirlos a una cadena utilizando **JSON.stringify()** y luego, para recuperar los datos, utilizar **JSON.parse()**.