Propiedades y Métodos de los Strings en JavaScript

PROPIEDADES de String

1. length

➢ Descripción: Devuelve el número total de caracteres de una cadena, incluidos los espacios y símbolos.

➢ Ejemplo:

```
let texto = "Hola mundo";
console.log(texto.length); // 10
```

➢ Explicación: La cadena "Hola mundo" tiene 10 caracteres: 4 letras para "Hola", 1 espacio, y 5 para "mundo".

2. constructor

➢ Descripción: Devuelve la función constructora del objeto String. No es útil en la mayoría de los casos prácticos.

➢ Ejemplo:

```
let texto = "Hola";
console.log(texto.constructor); // [Function: String]
```

➢ Explicación: Indica que el valor fue creado por el constructor String.

3. prototype *(solo accesible desde String como clase, no desde instancias)*

➢ Descripción: Propiedad que define los métodos compartidos por todas las instancias del tipo String.

➢ Ejemplo:

```
console.log(String.prototype);
```

➢ Explicación: Muestra todos los métodos definidos para los strings, como charAt, split, toUpperCase, etc.

4. __proto__ *(heredada de Object)*

➢ **Descripción:**

Permite acceder al prototipo del objeto, donde se definen sus métodos y propiedades heredadas.

➢ **Ejemplo:**

```
let texto = "Hola";
console.log(texto.__proto__ === String.prototype); // true
```

➢ **Explicación: Confirma que las instancias de string heredan de String.prototype.**

TABLA RESUMEN

Propiedad	¿Qué hace?	¿Ejemplo?
length	Cuenta los caracteres	"Hola mundo".length → 10
constructor	Devuelve la función constructora (String)	"Hola".constructor → [Function]
prototype	Define los métodos compartidos de String (no en instancias)	String.prototype.toUpperCase()
__proto__	Prototipo del objeto (heredado de Object)	"Hola".__proto__ === String.prototype

MÉTODOS de String

1. charAt(index). Devuelve el carácter en la posición indicada.

Ejemplo:

```
let texto = "Hola";
console.log(texto.charAt(1)); // "o"
```

2. **charCodeAt(index).** Devuelve el código Unicode del carácter en la posición indicada.

Ejemplo:

```
let texto = "Hola";
console.log(texto.charCodeAt(0)); // 72 ("H")
```

3. **at(index).** Permite acceder al carácter por posición positiva o negativa.

Ejemplo:

```
let texto = "Hola";
console.log(texto.at(-1)); // "a"
```

4. **concat(cadena2, ...).** Concatena una o más cadenas.

Ejemplo:

```
let saludo = "Hola";
console.log(saludo.concat(" mundo", "!")); // "Hola mundo!"
```

5. **includes(subcadena).** Verifica si una cadena contiene otra.

Ejemplo:

```
let frase = "Bienvenido al curso";
console.log(frase.includes("curso")); // true
```

6. **endsWith(subcadena).** Verifica si la cadena termina con la subcadena indicada.

Ejemplo:

```
let archivo = "documento.pdf";
console.log(archivo.endsWith(".pdf")); // true
```

7. **startsWith(subcadena).** Verifica si la cadena comienza con la subcadena indicada.

Ejemplo:

```
let frase = "Hola mundo";
console.log(frase.startsWith("Hola")); // true
```

8. **indexOf(subcadena).** **Devuelve la posición de la primera aparición de una subcadena.**

Ejemplo:

```
let texto = "Hola mundo";
console.log(texto.indexOf("mundo")); // 5
```

9. **lastIndexOf(subcadena).** **Devuelve la última posición en que aparece una subcadena.**

Ejemplo:

```
let texto = "ab abc ab";
console.log(texto.lastIndexOf("ab")); // 6
```

10. **padStart(longitud, relleno).** **Rellena el inicio de la cadena hasta alcanzar una longitud.**

Ejemplo:

```
let numero = "5";
console.log(numero.padStart(3, "0")); // "005"
```

11. **padEnd(longitud, relleno).** **Rellena el final de la cadena hasta alcanzar una longitud.**

Ejemplo:

```
let id = "12";
console.log(id.padEnd(5, "*")); // "12***"
```

12. **repeat(n).** **Repite la cadena n veces.**

Ejemplo:

```
let eco = "ja ";
console.log(eco.repeat(3)); // "ja ja ja "
```

13. **replace(valor, nuevo).** **Reemplaza una parte por otra (solo la primera coincidencia).**

Ejemplo:

```
let saludo = "Hola mundo";
console.log(saludo.replace("mundo", "amigo")); // "Hola amigo"
```

14. replaceAll(valor, nuevo). Reemplaza todas las coincidencias.

Ejemplo:

```
let texto = "gato gato gato";
console.log(texto.replaceAll("gato", "perro")); // "perro perro perro"
```

15. slice(inicio, fin). Devuelve una porción de la cadena.

Ejemplo:

```
let mensaje = "JavaScript";
console.log(mensaje.slice(0, 4)); // "Java"
```

16. substring(inicio, fin). Similar a slice, pero no acepta índices negativos.

Ejemplo:

```
let lenguaje = "JavaScript";
console.log(lenguaje.substring(4, 10)); // "Script"
```

17. substr(inicio, longitud). Obsoleto pero aún funciona. Devuelve parte de la cadena.

Ejemplo:

```
let palabra = "JavaScript";
console.log(palabra.substr(0, 4)); // "Java"
```

18. split(separador). Divide una cadena en un array según el separador.

Ejemplo:

```
let frase = "rojo,verde,azul";
console.log(frase.split(",")); // ["rojo", "verde", "azul"]
```

19. toLowerCase(). Convierte toda la cadena a minúsculas.

Ejemplo:

```
let saludo = "HOLA";
console.log(saludo.toLowerCase()); // "hola"
```

20. toUpperCase(). Convierte toda la cadena a mayúsculas.

Ejemplo:

```
let saludo = "hola";
console.log(saludo.toUpperCase()); // "HOLA"
```

21. trim(). Elimina los espacios al principio y al final.

Ejemplo:

```
let entrada = "   texto limpio   ";
console.log(entrada.trim()); // "texto limpio"
```

22. trimStart() / trimEnd(). Elimina solo al inicio o al final.

Ejemplo:

```
let entrada = "  hola  ";
console.log(entrada.trimStart()); // "hola  "
console.log(entrada.trimEnd());   // "  hola"
```

23. valueOf(). Devuelve el valor primitivo de un objeto String.

Ejemplo:

```
let str = new String("Hola");
console.log(str.valueOf()); // "Hola"
```

24. toString(). Convierte un objeto String en una cadena simple.

Ejemplo:

```
let str = new String("Texto");
console.log(str.toString()); // "Texto"
```

Propiedades y Métodos de Números en JavaScript

PROPIEDADES de Number

Number.MAX_VALUE

Valor numérico más grande representable.

Ejemplo:
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308

Number.MIN_VALUE

Valor numérico positivo más pequeño representable.

Ejemplo:
console.log(Number.MIN_VALUE); // 5e-324

Number.NaN

Representa un valor que no es un número.

Ejemplo:
console.log(Number.NaN);

Number.NEGATIVE_INFINITY

Representa el infinito negativo.

Ejemplo:
console.log(Number.NEGATIVE_INFINITY); // -Infinity

Number.POSITIVE_INFINITY

Representa el infinito positivo.

Ejemplo:
console.log(Number.POSITIVE_INFINITY); // Infinity

Number.EPSILON

Mínima diferencia entre dos números representables.

Ejemplo:
console.log(Number.EPSILON); // 2.220446049250313e-16

Number.MAX_SAFE_INTEGER

Mayor entero seguro representable.

Ejemplo:

```
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
```

Number.MIN_SAFE_INTEGER

Menor entero seguro representable.

Ejemplo:

```
console.log(Number.MIN_SAFE_INTEGER); // -9007199254740991
```

MÉTODOS de Instancias numéricas

toString([radix])

Convierte el número en una cadena, opcionalmente en una base numérica.

Ejemplo:

```
let num = 255;
console.log(num.toString());    // "255"
console.log(num.toString(16)); // "ff"
```

toFixed(n)

Devuelve una cadena con el número con n decimales.

Ejemplo:

```
let precio = 9.456;
console.log(precio.toFixed(2)); // "9.46"
```

toExponential(n)

Devuelve una cadena con el número en notación exponencial.

Ejemplo:

```
let x = 123000;
console.log(x.toExponential(2)); // "1.23e+5"
```

toPrecision(n)

Devuelve una cadena con el número con n cifras significativas.

Ejemplo:

```
let y = 123.456;
console.log(y.toPrecision(4)); // "123.5"
```

valueOf()

Devuelve el valor primitivo del número.

Ejemplo:

```
let numero = new Number(10);
console.log(numero.valueOf()); // 10
```

toLocaleString()

Devuelve una cadena con formato local del número.

Ejemplo:

```
let dinero = 1234567.89;
console.log(dinero.toLocaleString('es-ES', { style: 'currency', currency: 'EUR' })); // "1.234.567,89 €"
```

MÉTODOS Estáticos de Number

Number.isNaN(valor)

Verifica si el valor es NaN.

Ejemplo:

```
console.log(Number.isNaN(NaN));      // true
console.log(Number.isNaN("hola"));   // false
```

Number.isFinite(valor)

Verifica si el valor es un número finito.

Ejemplo:

```
console.log(Number.isFinite(100));      // true
console.log(Number.isFinite(Infinity)); // false
```

Number.isInteger(valor)

Verifica si el valor es un número entero.

Ejemplo:

```
console.log(Number.isInteger(5));    // true
console.log(Number.isInteger(5.2));  // false
```

Number.isSafeInteger(valor)

Verifica si el número es un entero seguro.

Ejemplo:

```
console.log(Number.isSafeInteger(9007199254740991)); // true
```

Number.parseFloat(cadena)

Convierte una cadena a número flotante.

Ejemplo:
console.log(Number.parseFloat("3.14")); // 3.14

Number.parseInt(cadena, base)

Convierte una cadena a número entero, con base opcional.

Ejemplo:
console.log(Number.parseInt("10", 2)); // 2

Resumen en Tabla

Propiedad / Método	¿Qué hace?	Ejemplo
length	X No aplica en números	–
MAX_VALUE	Mayor número representable	Number.MAX_VALUE
toFixed(n)	Redondea a n decimales (como string)	3.1415.toFixed(2) → "3.14"
isNaN(x)	Verifica si el valor no es un número	Number.isNaN("hola") → false
toLocaleString()	Formatea con separadores/locales	.toLocaleString('es-ES') → "1.000,00"
toString(base)	Convierte a texto (opcional base numérica)	255.toString(16) → "ff"

Propiedades y Métodos de Fechas en JavaScript

JavaScript utiliza el **objeto Date para manejar fechas y horas**. A continuación se presentan sus métodos más importantes con ejemplos.

Métodos Get (obtener información de la fecha)

getFullYear()

Devuelve el año completo.

```
let fecha = new Date("2024-04-15");
console.log(fecha.getFullYear()); // 2024
```

getMonth()

Devuelve el mes (0 = enero, 11 = diciembre).

```
let fecha = new Date("2024-04-15");
console.log(fecha.getMonth()); // 3 (abril)
```

getDate()

Devuelve el día del mes.

```
let fecha = new Date("2024-04-15");
console.log(fecha.getDate()); // 15
```

getDay()

Devuelve el día de la semana (0 = domingo, 6 = sábado).

```
let fecha = new Date("2024-04-15");
console.log(fecha.getDay()); // 1 (lunes)
```

getHours(), getMinutes(), getSeconds(), getMilliseconds()

Devuelven la hora, minutos, segundos o milisegundos.

```
let fecha = new Date("2024-04-15T14:25:30.123");
console.log(fecha.getHours());       // 14
console.log(fecha.getMinutes());     // 25
console.log(fecha.getSeconds());     // 30
console.log(fecha.getMilliseconds()); // 123
```

getTime()

Devuelve el timestamp desde 1970 en milisegundos.

```
let fecha = new Date("2024-04-15");
console.log(fecha.getTime()); // Ejemplo: 1713139200000
```

getTimezoneOffset()

Diferencia con UTC en minutos.

```
let fecha = new Date();
console.log(fecha.getTimezoneOffset()); // Ejemplo: -120
```

Métodos Set (modificar la fecha)

setFullYear(año)

Establece el año.

```
let fecha = new Date();
fecha.setFullYear(2030);
console.log(fecha.getFullYear()); // 2030
```

setMonth(mes)

Establece el mes (0 a 11).

```
let fecha = new Date();
fecha.setMonth(11);
console.log(fecha.getMonth()); // 11 (diciembre)
```

setDate(día)

Establece el día del mes.

```
let fecha = new Date();
fecha.setDate(25);
console.log(fecha.getDate()); // 25
```

setHours(), setMinutes(), setSeconds(), setMilliseconds()

Establecen hora, minutos, etc.

```
let fecha = new Date();
fecha.setHours(18);
console.log(fecha.getHours()); // 18
```

setTime(timestamp)

Establece la fecha con milisegundos desde 1970.

```
let fecha = new Date();
fecha.setTime(0);
console.log(fecha.toISOString()); // "1970-01-01T00:00:00.000Z"
```

Métodos de formato de fecha

toString()

Cadena legible localmente.

```
let fecha = new Date();
console.log(fecha.toString());
```

toDateString()

Solo parte de la fecha.

```
console.log(new Date().toDateString());
```

toTimeString()

Solo parte de la hora.

```
console.log(new Date().toTimeString());
```

toISOString()

Formato ISO 8601.

```
console.log(new Date().toISOString());
```

toLocaleDateString(), toLocaleTimeString(), toLocaleString()

Formato local de fecha/hora.

```
let fecha = new Date();
console.log(fecha.toLocaleDateString('es-ES')); // "30/4/2024"
console.log(fecha.toLocaleTimeString('es-ES')); // "14:25:00"
```

Propiedades y Métodos de Campos tipo Email en JavaScript

Los campos de tipo 'email' se usan en formularios HTML para ingresar direcciones de correo electrónico. A continuación, se detallan sus propiedades y métodos con ejemplos en JavaScript.

Propiedades del campo email

value

Contiene el valor actual del campo.

```
let email = document.querySelector("#correo");
console.log(email.value); // "usuario@dominio.com"
```

type

Devuelve el tipo de campo.

```
console.log(email.type); // "email"
```

name

Nombre del campo.

```
console.log(email.name); // "correo"
```

placeholder

Texto de ayuda cuando el campo está vacío.

```
<input type="email" placeholder="ejemplo@correo.com">
```

required

Indica si el campo es obligatorio.

```
<input type="email" required>
console.log(email.required); // true
```

readOnly

Impide edición por parte del usuario.

```
<input type="email" readonly>
```

disabled

Desactiva el campo.

```
email.disabled = true;
```

multiple

Permite varios correos separados por coma.

```
<input type="email" multiple>
console.log(email.multiple); // true
```

pattern

Permite establecer una expresión regular personalizada.

```
<input type="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$">
```

autocomplete

Controla si se autocompleta.

```
<input type="email" autocomplete="on">
```

Métodos del campo email

checkValidity()

Verifica si el valor cumple las reglas de validación.

```
if (!email.checkValidity()) {
  alert("Correo inválido");
}
```

reportValidity()

Muestra el mensaje de error del navegador.

```
email.reportValidity();
```

setCustomValidity(mensaje)

Define un mensaje de error personalizado.

```
email.setCustomValidity("Debes ingresar un correo válido");
```

focus() / blur()

Pone o quita el foco del campo.

```
email.focus();
```

select()

Selecciona el texto dentro del campo.

```
email.select();
```

form

Referencia al formulario padre.

console.log(email.form); // <form>...</form>

Ejemplo completo de uso

```
<!-- Formulario con campo de tipo email -->
<form id="miFormulario">
  <label>Correo:
    <!-- Campo de entrada tipo email -->
    <input
      type="email"            <!-- Solo acepta correos válidos -->
      id="correo"             <!-- ID único para acceder por JavaScript -->
      name="correo"           <!-- Nombre del campo en el formulario -->
      required                <!-- Hace que sea obligatorio rellenarlo -->
      placeholder="email@ejemplo.com"  <!-- Texto de ayuda cuando está vacío -->
      multiple>               <!-- Permite ingresar varios correos separados por coma -->
  </label>

  <!-- Botón para enviar el formulario -->
  <button type="submit">Enviar</button>
</form>

<script>
  // Obtenemos el campo de email por su ID
  const email = document.getElementById("correo");

  // Obtenemos el formulario por su ID
  const form = document.getElementById("miFormulario");

  // Escuchamos el evento 'submit' del formulario
  form.addEventListener("submit", function(e) {

    // Verificamos si el campo de email no es válido
    if (!email.checkValidity()) {
      // Muestra el mensaje de error del navegador automáticamente
      email.reportValidity();

      // Evita que el formulario se envíe
      e.preventDefault();
    } else {
      // Si el correo es válido, muestra su valor en una alerta
      alert("Correo válido: " + email.value);

    }

  });
</script>
```

Propiedades y Métodos del Campo tipo Teléfono en JavaScript

El campo de tipo 'tel' en HTML permite introducir números de teléfono. Es representado como un **objeto HTMLInputElement en JavaScript** y admite varias propiedades y métodos útiles.

Propiedades del input type="tel"

value

Devuelve o establece el número de teléfono como cadena.

```
<input type="tel" id="telefono">
<script>
  const tel = document.getElementById("telefono");
  console.log(tel.value); // "+34600111222"
</script>
```

type

Devuelve el tipo del campo (siempre será 'tel').

```
console.log(tel.type); // "tel"
```

name

Nombre del campo, usado en formularios.

```
<input type="tel" name="movil">
```

disabled

Indica si el campo está deshabilitado.

```
tel.disabled = true;
```

form

Referencia al formulario al que pertenece el campo.

```
console.log(tel.form);
```

required

Especifica si el campo es obligatorio.

```
<input type="tel" required>
```

pattern

Permite establecer una expresión regular para validar el formato del teléfono.

```
<input type="tel" pattern="\+?[0-9]{9,15}">
```

placeholder

Texto de ejemplo visible cuando el campo está vacío.

```
<input type="tel" placeholder="+34 600 111 222">
```

Métodos del input type="tel"

checkValidity()

Verifica si el campo cumple las restricciones (pattern, required).

```
if (!tel.checkValidity()) {
  console.log("Número no válido");
}
```

reportValidity()

Muestra el mensaje de error del navegador si no es válido.

```
tel.reportValidity();
```

setCustomValidity(mensaje)

Establece un mensaje personalizado de error de validación.

```
tel.setCustomValidity("Introduce un número de teléfono válido");
```

focus()

Coloca el foco sobre el input.

```
tel.focus();
```

blur()

Quita el foco del campo.

```
tel.blur();
```

select()

Selecciona el contenido del campo (si editable).

```
tel.select();
```

addEventListener()

Escucha eventos como 'input', 'change', 'focus', etc.

```
tel.addEventListener("input", () => {
  console.log("Nuevo valor:", tel.value);
});
```

Ejemplo completo de uso

```
<input type="tel" id="telefono" name="movil" placeholder="+34 600 111 222" pattern="\+?[0-9]{9,15}"
required>

<script>
  const tel = document.getElementById("telefono");

  // Mostrar valor actual
  console.log("Teléfono actual:", tel.value);

  // Validar al cambiar el número
  tel.addEventListener("change", () => {
    if (!tel.checkValidity()) {
      tel.setCustomValidity("Introduce un número válido con entre 9 y 15 dígitos.");
      tel.reportValidity();
    } else {
      tel.setCustomValidity(""); // Limpia el mensaje si es válido
      console.log("Número válido:", tel.value);
    }
  });
</script>
```

Propiedades y Métodos del tipo Button en JavaScript

Propiedades del botón (HTMLButtonElement)

type

Devuelve o establece el tipo de botón: 'button', 'submit', 'reset'.

```
let boton = document.getElementById("miBoton");
console.log(boton.type); // "button"
```

value

Valor del botón, útil en formularios.

```
<button id="btn" value="enviar">Enviar</button>
let btn = document.getElementById("btn");
console.log(btn.value); // "enviar"
```

name

Nombre del botón, utilizado al enviar formularios.

```
<button id="b1" name="accion">Aceptar</button>
let b1 = document.getElementById("b1");
console.log(b1.name); // "accion"
```

disabled

Desactiva el botón si se establece en true.

```
let boton = document.getElementById("miBoton");
boton.disabled = true;
```

form

Referencia al formulario al que pertenece el botón.

```
console.log(boton.form);
```

formAction

Define la URL donde se enviará el formulario.

```
<button type="submit" formAction="/otra-url">Enviar</button>
```

formMethod

Define el método de envío del formulario: 'get' o 'post'.

```
<button type="submit" formMethod="post">Enviar</button>
```

formTarget

Define el destino donde se abre la respuesta del formulario.

```
<button type="submit" formTarget="_blank">Enviar</button>
```

formEnctype

Define el tipo de codificación para el formulario.

```
<button type="submit" formEnctype="multipart/form-data">Subir</button>
```

autofocus

Establece el foco automáticamente al cargar la página.

```
<button id="inicio" autofocus>Comenzar</button>
```

Métodos del botón

click()

Simula un clic en el botón.

```
let boton = document.getElementById("miBoton");
boton.click();
```

focus()

Coloca el foco en el botón.

```
boton.focus();
```

blur()

Quita el foco del botón.

```
boton.blur();
```

addEventListener()

Asigna eventos al botón (por ejemplo, clic).

```
boton.addEventListener("click", function() {
  alert("Has hecho clic");
});
```

Ejemplo completo

```html
<!-- Botón en HTML -->
<button id="miBoton" type="button" value="aceptar">Haz clic</button>

<script>
  // Obtenemos una referencia al botón usando su ID
  const boton = document.getElementById("miBoton");

  // Mostramos por consola el tipo del botón (puede ser 'button', 'submit' o 'reset')
  console.log(boton.type); // "button"

  // Mostramos el valor del botón (atributo value en HTML)
  console.log(boton.value); // "aceptar"

  // Activamos el botón (por si estuviera desactivado previamente)
  boton.disabled = false;

  // Asignamos una función al evento "click" del botón
  boton.addEventListener("click", () => {
    // Esta función se ejecutará cada vez que se haga clic en el botón
    alert("¡Botón presionado!"); // Muestra una ventana emergente
  });

  // Usamos setTimeout para simular un clic automático después de 2 segundos (2000 milisegundos)
  setTimeout(() => {
    boton.click(); // Llama al método click() del botón como si el usuario lo pulsara
  }, 2000);
</script>
```

Código para usar (el mismo del ejemplo):

<!-- Botón en HTML -->

```
<button id="miBoton" type="button" value="aceptar">Haz clic</button>

<script>
 // Obtenemos una referencia al botón usando su ID
 const boton = document.getElementById("miBoton");

 // Mostramos por consola el tipo del botón (puede ser 'button', 'submit' o 'reset')
 console.log(boton.type); // "button"

 // Mostramos el valor del botón (atributo value en HTML)
 console.log(boton.value); // "aceptar"

 // Activamos el botón (por si estuviera desactivado previamente)
 boton.disabled = false;

 // Asignamos una función al evento "click" del botón
 boton.addEventListener("click", () => {
  // Esta función se ejecutará cada vez que se haga clic en el botón
  alert("¡Botón presionado!"); // Muestra una ventana emergente
 });

 // Usamos setTimeout para simular un clic automático después de 2 segundos (2000 milisegundos)
 setTimeout(() => {
```

```
   boton.click(); // Llama al método click() del botón como si el usuario lo pulsara
  }, 2000);
</script>
```

Propiedades y Métodos del Campo tipo Color en JavaScript

El campo de tipo 'color' en HTML representa un selector de color. Se accede mediante JavaScript como un objeto HTMLInputElement con tipo 'color'. Estas son sus principales propiedades y métodos con ejemplos:

Propiedades del input type="color"

value

Devuelve o establece el valor hexadecimal del color seleccionado.

```
<input type="color" id="colorInput">
<script>
  const color = document.getElementById("colorInput");
  console.log(color.value); // "#ff0000"
</script>
```

type

Devuelve el tipo del campo (siempre será 'color').

```
console.log(color.type); // "color"
```

name

Nombre del campo usado en formularios.

```
<input type="color" name="fondo">
```

disabled

Indica si el campo está desactivado.

```
color.disabled = true;
```

form

Referencia al formulario al que pertenece el campo.

```
console.log(color.form);
```

required

Establece si el campo es obligatorio.

```
<input type="color" required>
```

defaultValue

Valor inicial del campo.

```
console.log(color.defaultValue);
```

Métodos del input type="color"

checkValidity()

Verifica si el campo es válido según las restricciones.

```
if (!color.checkValidity()) {
  console.log("Color no válido");
}
```

reportValidity()

Muestra el mensaje de error del navegador si no es válido.

```
color.reportValidity();
```

setCustomValidity(mensaje)

Define un mensaje personalizado para errores de validación.

```
color.setCustomValidity("Debes elegir un color válido");
```

focus()

Coloca el foco sobre el input de color.

```
color.focus();
```

blur()

Quita el foco del campo.

```
color.blur();
```

select()

No tiene efecto visible en campos color, pero existe como método heredado.

```
color.select();
```

addEventListener()

Permite escuchar eventos como 'input' o 'change'.

```
color.addEventListener("change", () => {
  console.log("Nuevo color:", color.value);
});
```

Ejemplo completo de uso

```
<input type="color" id="colorInput" name="fondo" value="#00ff00">

<script>
  const color = document.getElementById("colorInput");

  // Mostrar color inicial
  console.log("Color actual:", color.value);

  // Validar al cambiar color
  color.addEventListener("change", () => {
    if (!color.checkValidity()) {
      color.setCustomValidity("Selecciona un color válido");
      color.reportValidity();
    } else {
      console.log("Color válido:", color.value);
    }
  });
</script>
```

Propiedades y Métodos del Campo tipo Range en JavaScript

El campo de tipo 'range' en HTML permite al usuario seleccionar un valor numérico dentro de un rango definido. Este campo se representa como un objeto HTMLInputElement en JavaScript.

Propiedades del input type="range"

value

Devuelve o establece el valor actual del control deslizante.

```
<input type="range" id="rango" min="0" max="100" value="50">
<script>
  const rango = document.getElementById("rango");
  console.log(rango.value); // "50"
</script>
```

min

Devuelve o establece el valor mínimo permitido.

```
console.log(rango.min); // "0"
```

max

Devuelve o establece el valor máximo permitido.

```
console.log(rango.max); // "100"
```

step

Define la cantidad que cambia el valor cuando se arrastra.

```
rango.step = "5";
```

type

Devuelve el tipo del campo, que será 'range'.

```
console.log(rango.type); // "range"
```

name

Nombre del campo en formularios.

```
<input type="range" name="nivel">
```

disabled

Desactiva el control si es true.

```
rango.disabled = true;
```

form

Referencia al formulario al que pertenece el input.

```
console.log(rango.form);
```

required

Indica si el campo es obligatorio.

```
<input type="range" required>
```

Métodos del input type="range"

checkValidity()

Verifica si el valor del rango es válido.

```
if (!rango.checkValidity()) {
  console.log("Valor no válido");
}
```

reportValidity()

Muestra el mensaje de error si no es válido.

```
rango.reportValidity();
```

setCustomValidity()

Establece un mensaje personalizado de error de validación.

```
rango.setCustomValidity("Selecciona un valor permitido");
```

focus()

Coloca el foco en el campo.

```
rango.focus();
```

blur()

Quita el foco del campo.

```
rango.blur();
```

select()

Método heredado, no tiene efecto visible en inputs tipo range.

```
rango.select();
```

addEventListener()

Escucha eventos como 'input' o 'change'.

```
rango.addEventListener("input", () => {
 console.log("Valor actual:", rango.value);
});
```

Ejemplo completo de uso

```
<input type="range" id="rango" name="nivel" min="0" max="100" value="50" step="10">

<script>
 const rango = document.getElementById("rango");

 // Mostrar el valor inicial
 console.log("Valor inicial:", rango.value);

 // Escuchar cambios y mostrar el nuevo valor
 rango.addEventListener("input", () => {
  console.log("Nuevo valor:", rango.value);
 });

 // Validar rango si es requerido
 rango.addEventListener("change", () => {
  if (!rango.checkValidity()) {
   rango.setCustomValidity("Selecciona un número válido entre 0 y 100.");
   rango.reportValidity();
  } else {
   rango.setCustomValidity(""); // Limpia el error si es válido
  }
 });
</script>
```

Propiedades y Métodos del Campo tipo Checkbox en JavaScript

El campo de tipo 'checkbox' en HTML permite seleccionar o deseleccionar una opción. Se representa como un objeto HTMLInputElement en JavaScript con múltiples propiedades útiles.

Propiedades del input type="checkbox"

checked

Indica si el checkbox está marcado (true) o no (false).

```
<input type="checkbox" id="acepto">
<script>
  const chk = document.getElementById("acepto");
  console.log(chk.checked); // true o false
</script>
```

value

Devuelve o establece el valor que se enviará si está marcado.

```
<input type="checkbox" id="chk" value="si">
<script>
  const chk = document.getElementById("chk");
  console.log(chk.value); // "si"
</script>
```

name

Nombre del campo, usado en formularios.

```
<input type="checkbox" name="terminos">
```

type

Devuelve el tipo del input (siempre será 'checkbox').

```
console.log(chk.type); // "checkbox"
```

disabled

Si está en true, el checkbox está desactivado.

```
chk.disabled = true;
```

form

Referencia al formulario al que pertenece.

```
console.log(chk.form);
```

required

Indica si es obligatorio marcarlo.

```
<input type="checkbox" required>
```

Métodos del input type="checkbox"

checkValidity()

Verifica si se cumplen las restricciones de validación.

```
if (!chk.checkValidity()) {
  console.log("Debe marcarse esta casilla");
}
```

reportValidity()

Muestra el mensaje de error del navegador si no es válido.

```
chk.reportValidity();
```

setCustomValidity()

Establece un mensaje de error personalizado.

```
chk.setCustomValidity("Debes aceptar los términos.");
```

focus()

Coloca el foco sobre el checkbox.

```
chk.focus();
```

blur()

Quita el foco del checkbox.

```
chk.blur();
```

click()

Simula un clic (cambia el estado del checkbox).

```
chk.click();
```

addEventListener()

Permite escuchar eventos como 'change' o 'click'.

```
chk.addEventListener("change", () => {
  console.log("Marcado:", chk.checked);
});
```

Ejemplo completo de uso

```
<input type="checkbox" id="acepto" name="terminos" required>

<script>
  const chk = document.getElementById("acepto");

  // Mostrar si está marcado inicialmente
  console.log("¿Marcado?:", chk.checked);

  // Escuchar cambios
  chk.addEventListener("change", () => {
    if (!chk.checked) {
      chk.setCustomValidity("Debes aceptar los términos para continuar.");
    } else {
      chk.setCustomValidity("");
    }
    chk.reportValidity();
  });
</script>
```

Propiedades y Métodos del Campo tipo Radio en JavaScript

Los campos de tipo 'radio' en HTML permiten seleccionar solo una opción dentro de un grupo. En JavaScript, son objetos de tipo HTMLInputElement y pueden manipularse como cualquier otro input.

Propiedades del input type="radio"

checked

Devuelve true si el radio está seleccionado.

```
<input type="radio" id="op1" name="grupo" value="A">
<script>
  const radio = document.getElementById("op1");
  console.log(radio.checked); // true o false
</script>
```

value

Devuelve o establece el valor asociado a ese radio.

```
<input type="radio" id="r2" value="Opción2">
<script>
  const r2 = document.getElementById("r2");
  console.log(r2.value); // "Opción2"
</script>
```

name

Nombre del grupo. Radios con el mismo nombre son parte del mismo grupo.

```
<input type="radio" name="opcion">
```

type

Devuelve 'radio'.

```
console.log(radio.type); // "radio"
```

disabled

Desactiva el radio si se establece en true.

```
radio.disabled = true;
```

form

Devuelve el formulario al que pertenece el radio.

```
console.log(radio.form);
```

required

Indica si se debe seleccionar una opción del grupo.

```
<input type="radio" required>
```

Métodos del input type="radio"

checkValidity()

Verifica si se cumplen las restricciones.

```
if (!radio.checkValidity()) {
  console.log("Selecciona una opción");
}
```

reportValidity()

Muestra mensaje de validación del navegador.

```
radio.reportValidity();
```

setCustomValidity()

Establece un mensaje de error personalizado.

```
radio.setCustomValidity("Debes seleccionar una opción");
```

focus()

Coloca el foco en el radio.

```
radio.focus();
```

blur()

Quita el foco del radio.

```
radio.blur();
```

click()

Simula un clic en el botón de opción.

```
radio.click();
```

addEventListener()

Permite reaccionar a eventos como 'change'.

```
radio.addEventListener("change", () => {
  console.log("Radio seleccionado:", radio.value);
});
```

Ejemplo completo de uso

```
<form id="formulario">
<label><input type="radio" name="grupo" value="A" id="op1" required> Opción A</label>
<label><input type="radio" name="grupo" value="B" id="op2"> Opción B</label>
</form>

<script>
  const radio1 = document.getElementById("op1");
  const radio2 = document.getElementById("op2");

  // Mostrar si están seleccionados
  console.log("¿A seleccionado?:", radio1.checked);
  console.log("¿B seleccionado?:", radio2.checked);

  // Escuchar cambios
  radio1.addEventListener("change", () => {
    console.log("Seleccionaste A");
  });

  radio2.addEventListener("change", () => {
    console.log("Seleccionaste B");
  });
</script>
```

Campo de Tipo Password en JavaScript (Actualizado a 2025)

Los campos de tipo `password` son elementos de formulario utilizados para introducir contraseñas. Estos elementos ocultan los caracteres introducidos y tienen propiedades, métodos y eventos específicos que se pueden manipular con JavaScript.

Propiedades del campo password

value

Obtiene o establece el valor del campo.

```
const pass = document.getElementById("clave");
console.log(pass.value); // muestra la contraseña introducida
```

type

Devuelve 'password' o puede cambiarse a 'text' para mostrar.

```
pass.type = "text"; // mostrar temporalmente la contraseña
```

placeholder

Texto de ayuda cuando el campo está vacío.

```
pass.placeholder = "Introduce tu contraseña";
```

maxLength

Longitud máxima permitida en caracteres.

```
pass.maxLength = 12;
```

minLength

Longitud mínima requerida.

```
pass.minLength = 6;
```

required

Especifica si el campo es obligatorio.

```
pass.required = true;
```

readOnly

Impide editar el campo.

```
pass.readOnly = true;
```

disabled

Inhabilita el campo.

```
pass.disabled = true;
```

name

Nombre que se envía con el formulario.

```
pass.name = "usuarioClave";
```

Métodos aplicables

focus()

Coloca el cursor en el campo.

```
pass.focus();
```

blur()

Quita el foco del campo.

```
pass.blur();
```

select()

Selecciona el contenido del campo.

```
pass.select();
```

setCustomValidity()

Establece un mensaje de error personalizado.

```
pass.setCustomValidity("Debe tener al menos 6 caracteres");
```

checkValidity()

Verifica si el campo cumple las restricciones.

```
if (!pass.checkValidity()) {
  alert("Contraseña no válida");
}
```

Eventos del campo password

input

Cuando se introduce texto en el campo.

```
pass.addEventListener("input", () => {
  console.log("Valor actual:", pass.value);
});
```

change

Cuando el campo cambia y pierde el foco.

```
pass.addEventListener("change", () => {
  console.log("Cambio confirmado");
});
```

focus

Cuando el campo recibe el foco.

```
pass.addEventListener("focus", () => {
  console.log("Campo enfocado");
});
```

blur

Cuando el campo pierde el foco.

```
pass.addEventListener("blur", () => {
  console.log("Campo desenfocado");
});
```

Propiedades y Métodos del Elemento Textarea en JavaScript

El elemento <textarea> permite introducir texto en múltiples líneas. Es útil para comentarios, descripciones y contenido largo. En JavaScript se accede como un HTMLTextAreaElement.

Propiedades del <textarea>

value

Devuelve o establece el contenido del textarea.

```
<textarea id="comentario">Hola</textarea>
<script>
  const txt = document.getElementById("comentario");
  console.log(txt.value); // "Hola"
</script>
```

name

Nombre usado al enviar formularios.

```
<textarea name="opinion"></textarea>
```

placeholder

Texto de ayuda cuando está vacío.

```
<textarea placeholder="Escribe aquí..."></textarea>
```

disabled

Desactiva el campo si es true.

```
txt.disabled = true;
```

readOnly

Hace que el campo no se pueda editar.

```
txt.readOnly = true;
```

required

Obliga a rellenarlo antes de enviar.

```
<textarea required></textarea>
```

maxlength

Límite de caracteres permitidos.

```
<textarea maxlength="200"></textarea>
```

rows

Número de filas visibles.

```
<textarea rows="4"></textarea>
```

cols

Número de columnas visibles.

```
<textarea cols="50"></textarea>
```

Métodos del <textarea>

focus()

Coloca el cursor en el textarea.

```
txt.focus();
```

blur()

Quita el foco del textarea.

```
txt.blur();
```

select()

Selecciona todo el contenido.

```
txt.select();
```

setCustomValidity()

Define un mensaje personalizado de error.

```
txt.setCustomValidity("Este campo es obligatorio");
```

checkValidity()

Verifica si el contenido es válido.

```
if (!txt.checkValidity()) {
  console.log("No es válido");
}
```

reportValidity()

Muestra mensaje de error del navegador.

```
txt.reportValidity();
```

addEventListener()

Escucha eventos como 'input' o 'change'.

```
txt.addEventListener("input", () => {
  console.log("Nuevo texto:", txt.value);
});
```

Ejemplo completo de uso

```
<textarea id="comentario" name="opinion" rows="4" cols="50" placeholder="Escribe aquí..."
required></textarea>

<script>
  const txt = document.getElementById("comentario");

  // Mostrar texto actual
  console.log("Contenido:", txt.value);

  // Validar y mostrar mensaje si no está rellenado
  txt.addEventListener("blur", () => {
    if (!txt.checkValidity()) {
      txt.setCustomValidity("Este campo no puede estar vacío.");
      txt.reportValidity();
    } else {
      txt.setCustomValidity("");
    }
  });

  // Detectar cambios
  txt.addEventListener("input", () => {
    console.log("Escribiendo:", txt.value);
  });
</script>
```

Propiedades y Métodos del Campo tipo Lista (select) en JavaScript

El campo de tipo 'select' en HTML **crea una lista desplegable de opciones.** En JavaScript, se manipula como un objeto HTMLSelectElement, con múltiples propiedades y métodos.

Propiedades del <select>

value

Devuelve o establece el valor de la opción seleccionada.

```
<select id="lista">
<option value="1">Uno</option>
<option value="2">Dos</option>
</select>


<script>
  const lista = document.getElementById("lista");
  console.log(lista.value); // "1" si la opción Uno está seleccionada
</script>
```

name

Nombre del campo usado en formularios.

```
<select name="opcion"></select>
```

length

Número total de opciones en la lista.

```
console.log(lista.length);
```

selectedIndex

Índice de la opción seleccionada.

```
console.log(lista.selectedIndex); // 0 o 1
```

options

Colección de todas las opciones.

```
console.log(lista.options[0].text); // "Uno"
```

disabled

Indica si el select está deshabilitado.

```
lista.disabled = true;
```

form

Referencia al formulario al que pertenece.

```
console.log(lista.form);
```

multiple

Permite seleccionar más de una opción.

```
<select multiple></select>
```

required

Campo obligatorio para el envío.

```
<select required></select>
```

Métodos del <select>

add()

Agrega una nueva opción a la lista.

```
let nueva = new Option("Tres", "3");
lista.add(nueva);
```

remove()

Elimina una opción según el índice.

```
lista.remove(1); // Elimina la segunda opción
```

item()

Devuelve el nodo opción según el índice.

```
console.log(lista.item(0).value);
```

namedItem()

Devuelve la opción con un nombre específico.

```
console.log(lista.namedItem("opcion1"));
```

checkValidity()

Verifica si se cumple la validación.

```
if (!lista.checkValidity()) {
  console.log("Debes seleccionar una opción válida");
}
```

reportValidity()

Muestra mensaje del navegador si no es válido.

```
lista.reportValidity();
```

setCustomValidity()

Define un mensaje personalizado de error.

```
lista.setCustomValidity("Selecciona una opción de la lista.");
```

focus()

Coloca el foco en la lista.

```
lista.focus();
```

blur()

Quita el foco de la lista.

```
lista.blur();
```

addEventListener()

Escucha eventos como 'change' o 'input'.

```
lista.addEventListener("change", () => {
  console.log("Seleccionado:", lista.value);
});
```

Ejemplo completo de uso

```
<select id="lista" name="opciones" required>
<option value="">Selecciona una opción</option>
<option value="1">Uno</option>
<option value="2">Dos</option>
</select>

<script>
  const lista = document.getElementById("lista");

  // Mostrar opción seleccionada
  console.log("Valor actual:", lista.value);
```

```
  // Escuchar cambio
  lista.addEventListener("change", () => {
    if (!lista.checkValidity()) {
      lista.setCustomValidity("Debes seleccionar una opción.");
      lista.reportValidity();
    } else {
      lista.setCustomValidity("");
      console.log("Seleccionado:", lista.value);
    }
  });
</script>
```

Propiedades y Métodos del Tipo Array en JavaScript

Los arrays en JavaScript son objetos que permiten almacenar múltiples valores en una sola variable. Son muy utilizados por su flexibilidad y gran cantidad de métodos incorporados.

Propiedades del Array

length

Devuelve el número de elementos en el array.

```
const arr = [1, 2, 3];
console.log(arr.length); // 3
```

Métodos del Array

push()

Agrega uno o más elementos al final.

```
let arr = [1, 2];
arr.push(3);
console.log(arr); // [1, 2, 3]
```

pop()

Elimina el último elemento y lo devuelve.

```
let arr = [1, 2, 3];
arr.pop(); // devuelve 3
console.log(arr); // [1, 2]
```

shift()

Elimina el primer elemento y lo devuelve.

```
let arr = [1, 2, 3];
arr.shift(); // devuelve 1
console.log(arr); // [2, 3]
```

unshift()

Agrega elementos al principio.

```
let arr = [2, 3];
arr.unshift(1);
console.log(arr); // [1, 2, 3]
```

concat()

Combina arrays o valores.

```
let a = [1, 2];
let b = [3, 4];
let c = a.concat(b);
console.log(c); // [1, 2, 3, 4]
```

join()

Une los elementos con un separador.

```
let arr = ["a", "b", "c"];
console.log(arr.join("-")); // "a-b-c"
```

slice()

Devuelve una parte sin modificar el original.

```
let arr = [1, 2, 3, 4];
let sub = arr.slice(1, 3);
console.log(sub); // [2, 3]
```

splice()

Agrega o elimina elementos.

```
let arr = [1, 2, 3];
arr.splice(1, 1, "a");
console.log(arr); // [1, "a", 3]
```

indexOf()

Devuelve el índice de un valor o -1.

```
let arr = [1, 2, 3];
console.log(arr.indexOf(2)); // 1
```

includes()

Verifica si contiene un valor.

```
let arr = [1, 2, 3];
console.log(arr.includes(2)); // true
```

find()

Devuelve el primer valor que cumple condición.

```
let arr = [1, 2, 3];
let found = arr.find(x => x > 1);
console.log(found); // 2
```

findIndex()

Índice del primer elemento que cumple condición.

```
let arr = [1, 2, 3];
console.log(arr.findIndex(x => x > 1)); // 1
```

map()

Crea nuevo array aplicando función a cada elemento.

```
let nums = [1, 2, 3];
let dobles = nums.map(x => x * 2);
console.log(dobles); // [2, 4, 6]
```

filter()

Filtra elementos según condición.

```
let nums = [1, 2, 3, 4];
let pares = nums.filter(x => x % 2 === 0);
console.log(pares); // [2, 4]
```

reduce()

Reduce el array a un solo valor.

```
let nums = [1, 2, 3];
let suma = nums.reduce((a, b) => a + b, 0);
console.log(suma); // 6
```

sort()

Ordena los elementos.

```
let arr = [3, 1, 2];
arr.sort();
console.log(arr); // [1, 2, 3]
```

reverse()

Invierte el orden de los elementos.

```
let arr = [1, 2, 3];
arr.reverse();
console.log(arr); // [3, 2, 1]
```

every()

Verifica si todos cumplen condición.

```
let arr = [2, 4, 6];
console.log(arr.every(x => x % 2 === 0)); // true
```

some()

Verifica si alguno cumple condición.

```
let arr = [1, 3, 4];
console.log(arr.some(x => x % 2 === 0)); // true
```

flat()

Aplana arrays anidados.

```
let arr = [1, [2, [3]]];
console.log(arr.flat(2)); // [1, 2, 3]
```

fill()

Rellena el array con un valor.

```
let arr = [1, 2, 3];
arr.fill(0);
console.log(arr); // [0, 0, 0]
```

Propiedades y Métodos del Tipo Function en JavaScript

En JavaScript, las funciones son objetos de primera clase, lo que significa que se pueden almacenar en variables, pasar como argumentos y devolver desde otras funciones. A continuación se detallan sus propiedades y métodos más importantes.

Propiedades del Tipo Function

name

Devuelve el nombre de la función.

```
function saludar() {
  return "Hola";
}
console.log(saludar.name); // "saludar"
```

length

Número de parámetros que acepta la función.

```
function sumar(a, b) {
  return a + b;
}
console.log(sumar.length); // 2
```

prototype

Accede al prototipo del objeto creado con una función constructora.

```
function Persona(nombre) {
  this.nombre = nombre;
}
Persona.prototype.saludar = function() {
  return "Hola " + this.nombre;
};
const p = new Persona("Ana");
console.log(p.saludar()); // "Hola Ana"
```

Métodos del Tipo Function

call()

Llama a la función con un contexto y argumentos individuales.

```
function saludar(nombre) {
  console.log("Hola " + nombre);
}
saludar.call(null, "Luis"); // "Hola Luis"
```

apply()

Llama a la función con un contexto y argumentos en array.

```
function sumar(a, b) {
  return a + b;
}
console.log(sumar.apply(null, [2, 3])); // 5
```

bind()

Crea una nueva función con contexto y argumentos predefinidos.

```
const persona = {
  nombre: "Ana",
  saludar: function() {
    console.log("Hola " + this.nombre);
  }
};
const saludo = persona.saludar.bind(persona);
saludo(); // "Hola Ana"
```

toString()

Devuelve el código fuente de la función como cadena.

```
function demo() {
  return true;
}
console.log(demo.toString());
```

Propiedades adicionales de funciones

arguments

Obsoleto. Array-like con los argumentos pasados a la función. No funciona con funciones flecha.

```
function demo(a, b) {
  console.log(arguments.length); // 2
}
demo(1, 2);
```

caller

Obsoleto. Referencia a la función que llamó a la actual. No disponible en modo estricto.

```
function f() {
  console.log(f.caller); // Muestra quién llamó a f
}
function g() {
  f();
}
g();
```

displayName

No estándar. Permite personalizar el nombre mostrado en herramientas de depuración.

```
function hola() {}
hola.displayName = "Saludo personalizado";
console.log(hola.displayName); // "Saludo personalizado"
```

constructor

Referencia al constructor `Function`. Todas las funciones lo tienen por herencia.

```
function ejemplo() {}
console.log(ejemplo.constructor === Function); // true
```

async

Solo en funciones declaradas como async, indica que devuelven una Promesa.

```
async function fetchData() {
  return "data";
}
console.log(fetchData().then); // función .then (es una Promise)
```

Funciones Flecha en JavaScript (Arrow Functions)

Las funciones flecha (arrow functions) fueron introducidas en ES6 (2015) y ofrecen una sintaxis más concisa para escribir funciones. Además, no vinculan su propio `this`, lo cual las hace útiles en muchos contextos.

¿Qué es una función flecha?

Es una forma breve de definir funciones anónimas. Se usan con el operador `=>`.

Ejemplos básicos con explicación línea a línea

```
// Función tradicional
function suma(a, b) {
  return a + b;
}

// Función flecha equivalente
const sumaFlecha = (a, b) => a + b;

// Uso con un parámetro y retorno implícito
const cuadrado = x => x * x;

// Sin parámetros
const saludar = () => console.log("Hola mundo");

// Bloque con return explícito
const resta = (a, b) => {
  return a - b;
};
```

Características clave

- No tiene su propio `this`: hereda el `this` del contexto donde se define.

- No tiene `arguments`, `super` ni puede ser usada como constructor.

- No tiene `new.target` ni puede usarse con `new`.

- Ideal para callbacks, métodos cortos, o expresiones simples.

Diferencias con funciones normales

Aspecto	Función normal	Función flecha
this	Cambia según contexto de invocación	Heredado del entorno donde se define
arguments	Disponible	No disponible
constructor	Puede usarse con `new`	No puede usarse con `new`
uso en callbacks	Correcto	Más limpio y claro

Propiedades y Métodos

Las funciones flecha también tienen acceso a:

- `.length`: número de parámetros definidos.

- `.name`: nombre asignado si se usa en una constante o variable.

- No tienen `.prototype` ni pueden ser instanciadas.

```
const sumar = (a, b) => a + b;
console.log(sumar.length); // 2
console.log(sumar.name);   // "sumar"
console.log(typeof sumar.prototype); // "undefined"
```

Casos de uso habituales

```
// En funciones de array
const nums = [1, 2, 3];
const dobles = nums.map(n => n * 2);
console.log(dobles); // [2, 4, 6]

// En funciones de orden superior
setTimeout(() => {
  console.log("Mensaje tras 1 segundo");
}, 1000);
```

Propiedades y Métodos del Tipo Object en JavaScript (Actualizado a 2025)

En JavaScript, todos los objetos heredan de `Object`, lo que los hace muy versátiles. A continuación, se listan las propiedades y métodos más relevantes de los objetos, con ejemplos actualizados hasta ECMAScript 2025.

Propiedades estáticas de Object

Object.prototype

Contiene todas las propiedades heredables por objetos.

```
console.log(Object.prototype.toString.call([])); // "[object Array]"
```

Métodos estáticos de Object

Object.assign()

Copia propiedades enumerables de uno o más objetos a un destino.

```
const destino = { a: 1 };
const fuente = { b: 2 };
Object.assign(destino, fuente);
console.log(destino); // { a: 1, b: 2 }
```

Object.create()

Crea un nuevo objeto con un prototipo especificado.

```
const persona = { saludar() { return "Hola"; } };
const ana = Object.create(persona);
ana.nombre = "Ana";
console.log(ana.saludar()); // "Hola"
```

Object.defineProperty()

Define o modifica una propiedad con configuración específica.

```
const obj = {};
Object.defineProperty(obj, "edad", { value: 30 });
console.log(obj.edad); // 30
```

Object.defineProperties()

Define múltiples propiedades a la vez.

```
const obj = {};
Object.defineProperties(obj, {
  nombre: { value: "Luis" },
  edad: { value: 40 }
});
console.log(obj.nombre); // "Luis"
```

Object.entries()

Devuelve un array de pares clave-valor.

```
const obj = { a: 1, b: 2 };
console.log(Object.entries(obj)); // [["a",1],["b",2]]
```

Object.fromEntries()

Convierte un array de pares en un objeto.

```
const arr = [["a", 1], ["b", 2]];
console.log(Object.fromEntries(arr)); // { a: 1, b: 2 }
```

Object.freeze()

Congela un objeto (no se puede modificar).

```
const obj = { x: 1 };
Object.freeze(obj);
obj.x = 2;
console.log(obj.x); // 1
```

Object.is()

Compara si dos valores son el mismo valor (como === pero más preciso).

```
console.log(Object.is(NaN, NaN)); // true
```

Object.isFrozen()

Verifica si un objeto está congelado.

```
const obj = Object.freeze({ a: 1 });
console.log(Object.isFrozen(obj)); // true
```

Object.isSealed()

Verifica si está sellado (no se pueden añadir/eliminar propiedades).

```
const obj = Object.seal({ b: 2 });
console.log(Object.isSealed(obj)); // true
```

Object.keys()

Devuelve un array con las claves del objeto.

```
const obj = { x: 10, y: 20 };
console.log(Object.keys(obj)); // ["x", "y"]
```

Object.values()

Devuelve un array con los valores del objeto.

```
const obj = { x: 10, y: 20 };
console.log(Object.values(obj)); // [10, 20]
```

Object.preventExtensions()

Evita agregar nuevas propiedades al objeto.

```
const obj = {};
Object.preventExtensions(obj);
obj.nueva = 1;
console.log(obj.nueva); // undefined
```

Object.seal()

Sella el objeto (no se pueden añadir ni borrar propiedades).

```
const obj = { x: 1 };
Object.seal(obj);
obj.x = 2;
delete obj.x;
console.log(obj.x); // 2
```

Object.getOwnPropertyDescriptor()

Devuelve los detalles de una propiedad.

```
const obj = { x: 42 };
console.log(Object.getOwnPropertyDescriptor(obj, "x"));
/*
{
  value: 42,
  writable: true,
  enumerable: true,
  configurable: true
}
*/
```

Object.getOwnPropertyDescriptors()

Devuelve todos los descriptores.

```
const obj = { x: 1 };
console.log(Object.getOwnPropertyDescriptors(obj));
```

Object.getOwnPropertyNames()

Devuelve un array con los nombres de propiedades (incluidas no enumerables).

```
const obj = Object.create({}, {
  oculto: { value: 1, enumerable: false }
});
console.log(Object.getOwnPropertyNames(obj)); // ["oculto"]
```

Object.getPrototypeOf()

Devuelve el prototipo de un objeto.

```
const obj = {};
console.log(Object.getPrototypeOf(obj) === Object.prototype); // true
```

Object.setPrototypeOf()

Establece el prototipo de un objeto.

```
const obj = {};
const proto = { saluda: () => "Hola" };
Object.setPrototypeOf(obj, proto);
console.log(obj.saluda()); // "Hola"
```

Métodos adicionales y de prototipo de Object

Object.hasOwn()

Devuelve true si el objeto tiene la propiedad especificada (propiedad propia). Introducido en ES2022.

```
const obj = { a: 1 };
console.log(Object.hasOwn(obj, "a")); // true
console.log(Object.hasOwn(obj, "b")); // false
```

Object.prototype.hasOwnProperty()

Devuelve true si el objeto tiene la propiedad como propia (no heredada).

```
const obj = { a: 1 };
console.log(obj.hasOwnProperty("a")); // true
```

Object.prototype.isPrototypeOf()

Verifica si el objeto aparece en la cadena de prototipos de otro objeto.

```
function Persona() {}
const p = new Persona();
console.log(Persona.prototype.isPrototypeOf(p)); // true
```

Object.prototype.propertyIsEnumerable()

Verifica si la propiedad es enumerable.

```
const obj = { a: 1 };
console.log(obj.propertyIsEnumerable("a")); // true
```

Object.prototype.toLocaleString()

Devuelve una representación localizada del objeto.

```
const obj = { a: 1 };
console.log(obj.toLocaleString()); // "[object Object]"
```

Object.prototype.toString()

Devuelve una cadena representando el objeto.

```
const arr = [];
console.log(arr.toString()); // "" (cadena vacía)
console.log(Object.prototype.toString.call(arr)); // "[object Array]"
```

Object.prototype.valueOf()

Devuelve el valor primitivo del objeto.

```
const obj = { a: 1 };
console.log(obj.valueOf()); // { a: 1 }
```

Propiedades y Métodos del Tipo RegExp en JavaScript (Actualizado a 2025)

El tipo `RegExp` en JavaScript representa una expresión regular, una poderosa herramienta para trabajar con patrones de texto. A continuación se listan sus propiedades y métodos más importantes, actualizados hasta ECMAScript 2025, con ejemplos.

Propiedades del objeto RegExp

flags

Devuelve los flags usados en la expresión regular.

```
const regex = /abc/gi;
console.log(regex.flags); // "gi"
```

global

Indica si tiene el flag 'g' (búsqueda global).

```
const regex = /abc/g;
console.log(regex.global); // true
```

ignoreCase

Indica si tiene el flag 'i' (ignorar mayúsculas/minúsculas).

```
const regex = /abc/i;
console.log(regex.ignoreCase); // true
```

multiline

Indica si tiene el flag 'm' (coincidencia en múltiples líneas).

```
const regex = /^abc/m;
console.log(regex.multiline); // true
```

dotAll

Indica si el punto (.) coincide con caracteres de nueva línea (flag 's').

```
const regex = /a.b/s;
console.log(regex.dotAll); // true
```

unicode

Indica si se interpreta como Unicode (flag 'u').

```
const regex = /\u{61}/u;
console.log(regex.unicode); // true
```

sticky

Indica si la búsqueda es 'pegajosa' (flag 'y').

```
const regex = /abc/y;
console.log(regex.sticky); // true
```

source

Devuelve el patrón como cadena (sin delimitadores ni flags).

```
const regex = /hola mundo/;
console.log(regex.source); // "hola mundo"
```

lastIndex

Posición en la cadena donde comienza la próxima búsqueda.

```
const regex = /a/g;
regex.lastIndex = 2;
console.log(regex.lastIndex); // 2
```

Los **flags** son letras que modifican el comportamiento de una expresión regular en JavaScript. **Se colocan al final del patrón (después de la segunda barra inclinada)** y permiten ajustar cómo se realiza la búsqueda.

Resumen Tabla de Flags

Flag	Nombre	Descripción
g	**global**	Busca todas las coincidencias (no se detiene en la primera).
i	**ignoreCase**	Ignora la diferencia entre mayúsculas y minúsculas.
m	**multiline**	Permite que ^ y $ coincidan con los inicios y finales de cada línea, no solo del texto completo.
s	**dotAll**	Hace que el punto (.) también coincida con saltos de línea (\n).
u	**unicode**	Interpreta el patrón como texto Unicode completo (permite \u{XXXX}).
y	**sticky**	Coincide solo desde la posición indicada en lastIndex.

Ejemplos de uso de cada flag

g – global

```
const regex = /a/g;
console.log("banana".match(regex)); // ['a', 'a', 'a']
```

i – ignore case

```
const regex = /hola/i;
console.log(regex.test("HOLA")); // true
```

m – multiline

```
const regex = /^Hola/m;
console.log("Adiós\nHola mundo".match(regex)); // ["Hola"]
```

s – dotAll

```
const regex = /a.b/s;
console.log(regex.test("a\nb")); // true
```

u – unicode

```
const regex = /\u{1F600}/u; // 
console.log(regex.test("")); // true
```

y – sticky

```
const regex = /ab/y;
regex.lastIndex = 2;
console.log(regex.exec("xxabyyab")); // null
```

Métodos del objeto RegExp

test()

Devuelve true si hay una coincidencia con el patrón.

```
const regex = /abc/;
console.log(regex.test("abcdef")); // true
```

exec()

Devuelve un array con los resultados o null.

```
1. const regex = /a(b)c/;
2. const result = regex.exec("abc");
3. console.log(result[0]); // "abc"
4. console.log(result[1]); // "b"
```

Explicación de exec línea a línea

Línea 1

Se define una expresión regular `/a(b)c/` que busca la secuencia 'a', luego 'b' (capturada en grupo), y finalmente 'c'. El grupo de captura se define con paréntesis: `(b)`.

Línea 2

Se aplica el método `.exec()` a la cadena 'abc'. Esto devuelve un array con la coincidencia completa y cualquier grupo capturado.

Línea 3

`result[0]` contiene la coincidencia completa del patrón, que es 'abc'.

Línea 4

`result[1]` contiene el primer grupo de captura, que es solo la letra 'b' (lo que estaba entre paréntesis en la expresión regular).

Resultado de `exec()` en este caso:

Índice	Contenido
result[0]	Coincidencia completa: "abc"
result[1]	Primer grupo capturado: "b"

Métodos Especiales y Symbol.* del Tipo RegExp en JavaScript

Los métodos **Symbol.*** permiten a los objetos **RegExp** personalizar cómo interactúan con **métodos del tipo String, como match, replace, search y split**. A continuación se explican estos métodos y el método toString(), con ejemplos y explicaciones línea por línea.

[Symbol.match]()

Usado internamente por String.prototype.match(). Devuelve todas las coincidencias.

```
const regex = /a./g;
console.log("abc ab".match(regex)); // ["ab", "ab"]
```

• regex busca una 'a' seguida de cualquier carácter.
• El flag 'g' indica búsqueda global.
• .match(regex) devuelve todas las coincidencias encontradas.
• Resultado: ['ab', 'ab']

[Symbol.replace]()

Usado internamente por String.prototype.replace(). Reemplaza coincidencias por otra cadena.

```
const regex = /dog/;
console.log("my dog".replace(regex, "cat")); // "my cat"
```

• regex busca la palabra 'dog'.
• .replace reemplaza la coincidencia con 'cat'.
• Resultado: 'my cat'

[Symbol.search]()

Usado internamente por String.prototype.search(). Devuelve el índice de la primera coincidencia.

```
const regex = /dog/;
console.log("my dog".search(regex)); // 3
```

• Busca la posición donde aparece 'dog' en la cadena.
• 'dog' comienza en el índice 3.
• Resultado: 3

[Symbol.split]()

Usado internamente por String.prototype.split(). Divide una cadena según la expresión.

```
const regex = /,/;
console.log("a,b,c".split(regex)); // ["a", "b", "c"]
```

• Busca las comas como separador.
• Divide la cadena en cada coincidencia.
• Resultado: ['a', 'b', 'c']

toString()

Devuelve la expresión regular como una cadena (con delimitadores y flags).

```
const regex = /hola/i;
console.log(regex.toString()); // "/hola/i"
```

- Convierte la expresión regular a texto.
- Incluye delimitadores '/' y el flag 'i'.
- Resultado: '/hola/i'

Código Síncrono vs Asíncrono en JavaScript (Actualizado a 2025)

JavaScript es un lenguaje de ejecución única (single-threaded), pero puede manejar tareas asíncronas gracias al event loop y Web APIs del navegador o motor de ejecución. A continuación se explica la diferencia entre código síncrono y asíncrono con ejemplos detallados.

¿Qué es código SÍNCRONO?

Es aquel que se ejecuta línea a línea. Cada instrucción se completa antes de pasar a la siguiente.

```
console.log("Inicio");
console.log("Proceso");
console.log("Fin");
```

La consola mostrará:
Inicio
Proceso
Fin

¿Qué es código ASÍNCRONO?

Permite ejecutar tareas en segundo plano sin bloquear el hilo principal. JavaScript continúa con el resto del código.

```
console.log("Inicio");

setTimeout(() => {
  console.log("Tarea asíncrona");
}, 1000);

console.log("Fin");
```

Resultado en consola:
Inicio
Fin
Tarea asíncrona

Promesas (Promise) en JavaScript - Actualizado a 2025

Una Promise (promesa) es un objeto que representa la terminación (o el fracaso) de una operación asíncrona. Fue introducida en ECMAScript 2015 (ES6) y es la base para trabajar con código asíncrono moderno en JavaScript, especialmente junto con async/await.

¿Qué es una Promise?

Una Promise es un objeto que puede estar en uno de los siguientes estados:

- **pending**: estado inicial, ni cumplida ni rechazada.

- **fulfilled**: la operación se completó con éxito.

- **rejected**: la operación falló.

```
const promesa = new Promise((resolve, reject) => {
  // Simulación de operación asíncrona
  setTimeout(() => {
    resolve("Operación exitosa");
    // o: reject("Ocurrió un error");
  }, 1000);
});
```

Propiedades de una Promise

No tiene propiedades públicas como objetos tradicionales. Su estado se gestiona internamente y se accede a través de métodos.

Métodos de Promise

then()

Registra una función que se ejecuta cuando la promesa se cumple.

```
promesa.then(resultado => {
  console.log("Éxito:", resultado);
});
```

catch()

Registra una función que se ejecuta si la promesa se rechaza.

```
promesa.catch(error => {
  console.error("Error:", error);
});
```

finally()

Se ejecuta cuando la promesa se cumple o se rechaza.

```
promesa.finally(() => {
  console.log("La promesa ha finalizado (éxito o error).");
});
```

Promise.resolve()

Devuelve una promesa ya cumplida con un valor.

```
const p = Promise.resolve(42);
p.then(v => console.log(v)); // 42
```

Promise.reject()

Devuelve una promesa ya rechazada con un error.

```
const p = Promise.reject("Error crítico");
p.catch(e => console.error(e));
```

Promise.all()

Devuelve una promesa que se resuelve cuando todas las promesas se cumplen.

```
Promise.all([Promise.resolve(1), Promise.resolve(2)])
  .then(values => console.log(values)); // [1, 2]
```

Promise.allSettled()

Devuelve una promesa que se resuelve cuando todas se completan (con éxito o fallo).

```
Promise.allSettled([Promise.resolve(1), Promise.reject("fallo")])
  .then(results => console.log(results));
```

Promise.any()

Devuelve la primera promesa que se cumple (ignora rechazos).

```
Promise.any([Promise.reject("fallo"), Promise.resolve("ok")])
  .then(result => console.log(result)); // "ok"
```

Promise.race()

Devuelve la primera promesa que se resuelve o se rechaza.

```
Promise.race([
  new Promise(res => setTimeout(() => res("1"), 500)),
  new Promise(res => setTimeout(() => res("2"), 100))
]).then(v => console.log(v)); // "2"
```

Propiedades y Métodos del Elemento <img> en JavaScript (Actualizado a 2025)

El elemento `<img>` en HTML representa una imagen. En JavaScript, puede manipularse como un objeto `HTMLImageElement`, heredando propiedades y métodos útiles para controlar sus atributos, estado de carga, dimensiones y eventos.

Propiedades del elemento <img>

src

Especifica la ruta de la imagen.

```
const img = document.getElementById("miImagen");
img.src = "imagen.jpg";
```

alt

Texto alternativo que se muestra si la imagen no se puede cargar.

```
img.alt = "Descripción de la imagen";
```

width

Ancho visual de la imagen (en píxeles).

```
img.width = 300;
```

height

Alto visual de la imagen (en píxeles).

```
img.height = 200;
```

naturalWidth

Ancho original de la imagen (sin escala).

```
console.log(img.naturalWidth);
```

naturalHeight

Alto original de la imagen.

```
console.log(img.naturalHeight);
```

complete

Devuelve true si la imagen ha terminado de cargarse.

```
console.log(img.complete);
```

currentSrc

La URL actual de la imagen cargada (resolviendo srcset si existe).

```
console.log(img.currentSrc);
```

crossOrigin

Controla CORS para imágenes externas.

```
img.crossOrigin = "anonymous";
```

decoding

Indica cómo debe decodificarse la imagen: 'sync', 'async' o 'auto'.

```
img.decoding = "async";
```

isMap

Indica si la imagen forma parte de un mapa de imagen.

```
img.isMap = true;
```

loading

Permite lazy-loading ('lazy', 'eager').

```
img.loading = "lazy";
```

referrerPolicy

Controla qué encabezado Referer se envía con la solicitud.

```
img.referrerPolicy = "no-referrer";
```

sizes

Define tamaños para imágenes responsivas.

```
img.sizes = "50vw";
```

srcset

Define versiones alternativas de la imagen para diferentes resoluciones.

```
img.srcset = "small.jpg 500w, large.jpg 1000w";
```

useMap

Especifica el uso de un <map> con la imagen.

```
img.useMap = "#mapa";
```

Métodos del objeto <img>

addEventListener()

Escucha eventos como 'load' o 'error'.

```
img.addEventListener("load", () => {
  console.log("Imagen cargada");
});
```

removeEventListener()

Elimina un event listener asignado.

```
function cuandoCarga() {
  console.log("Imagen lista");
}
img.addEventListener("load", cuandoCarga);
img.removeEventListener("load", cuandoCarga);
```

decode()

Devuelve una promesa que se resuelve cuando la imagen está decodificada.

```
img.decode().then(() => {
  console.log("Imagen decodificada y lista para mostrar");
});
```

click()

Simula un clic sobre la imagen.

```
img.click();
```

setAttribute()

Establece atributos HTML como 'title', 'alt', etc.

```
img.setAttribute("alt", "Nueva descripción");
```

getAttribute()

Obtiene un atributo HTML.

```
console.log(img.getAttribute("src"));
```

Mapas de Imagen en JavaScript

Un mapa de imagen en HTML define regiones interactivas dentro de una imagen. Estas regiones se pueden enlazar a URLs o responder a eventos usando JavaScript. Los mapas de imagen se crean con elementos <map> y <area>.

Estructura básica de un mapa de imagen

```
<img src="imagen.jpg" usemap="#miMapa" alt="Mapa" />

<map name="miMapa">
<area shape="rect" coords="34,44,270,350" href="https://ejemplo.com" alt="Zona 1" />
<area shape="circle" coords="400,200,50" href="https://otra.com" alt="Zona 2" />
</map>
```

- `usemap="#miMapa"`: enlaza la imagen con el mapa.
- `<area>` define una zona activa.
- `shape`: puede ser rect (rectángulo), circle (círculo) o poly (polígono).
- `coords`: define las coordenadas para esa forma.
- `href`: a dónde lleva al hacer clic.

Ejemplo completo con JavaScript

```
<img src="mapa.jpg" usemap="#mapaZonas" alt="Mapa de ejemplo" id="imagen" />

<map name="mapaZonas">
<area shape="rect" coords="0,0,100,100" alt="Cuadro 1" href="#" id="zona1" />
<area shape="circle" coords="200,200,50" alt="Círculo 2" href="#" id="zona2" />
</map>

<script>
document.getElementById("zona1").addEventListener("click", function(event) {
  event.preventDefault();
  alert("Has hecho clic en el Cuadro 1");
});

document.getElementById("zona2").addEventListener("click", function(event) {
  event.preventDefault();
  alert("Has hecho clic en el Círculo 2");
```

```
});
</script>
```

Este código crea una imagen con dos zonas interactivas:

- Un rectángulo que muestra un mensaje al hacer clic.

- Un círculo que también responde al clic con un mensaje.

Los `event.preventDefault()` evitan que se recargue la página.

Propiedades y Métodos del Elemento <video> en JavaScript (Actualizado a 2025)

El elemento `<video>` en HTML permite reproducir contenido audiovisual. En JavaScript se controla mediante la interfaz `HTMLVideoElement`, que hereda de `HTMLMediaElement` y ofrece propiedades y métodos específicos para controlar la reproducción, volumen, estado y más.

Propiedades del elemento <video>

src

Ruta del archivo de video.

```
video.src = "video.mp4";
```

currentTime

Tiempo actual de reproducción (en segundos).

```
console.log(video.currentTime);
```

duration

Duración total del video (en segundos).

```
console.log(video.duration);
```

paused

Devuelve true si el video está pausado.

```
console.log(video.paused);
```

ended

Devuelve true si el video ha terminado.

```
console.log(video.ended);
```

volume

Nivel de volumen (0.0 a 1.0).

```
video.volume = 0.5;
```

muted

Indica si el video está silenciado.

```
video.muted = true;
```

playbackRate

Velocidad de reproducción (1.0 es normal).

```
video.playbackRate = 1.5;
```

loop

Indica si el video se reinicia automáticamente al finalizar.

```
video.loop = true;
```

autoplay

Indica si el video debe comenzar automáticamente.

```
video.autoplay = true;
```

controls

Indica si se deben mostrar los controles del navegador.

```
video.controls = true;
```

poster

Imagen mostrada antes de reproducir.

```
video.poster = "preview.jpg";
```

preload

Sugerencia al navegador sobre qué cargar (auto, metadata, none).

```
video.preload = "metadata";
```

readyState

Indica cuánto del video está listo para reproducirse.

```
console.log(video.readyState);
```

buffered

Rangos del video que ya han sido cargados.

```
console.log(video.buffered);
```

Métodos del objeto <video>

play()

Inicia la reproducción del video. Devuelve una promesa.

```
video.play().then(() => {
  console.log("Video reproduciéndose");
}).catch(err => {
  console.error("Error al reproducir:", err);
});
```

pause()

Pausa la reproducción del video.

```
video.pause();
```

load()

Recarga el video (útil si se cambia el src).

```
video.load();
```

canPlayType()

Verifica si el navegador puede reproducir un tipo de archivo.

```
console.log(video.canPlayType("video/mp4"));
```

addEventListener()

Escucha eventos como 'play', 'pause', 'ended', etc.

```
video.addEventListener("ended", () => {
  console.log("El video ha terminado");
});
```

Propiedades y Métodos del Elemento <audio> en JavaScript (Actualizado a 2025)

El elemento `<audio>` en HTML se usa para reproducir sonido. En JavaScript se manipula a través de la interfaz `HTMLAudioElement`, que hereda de `HTMLMediaElement`. Permite controlar la reproducción, volumen, tiempo y más.

Propiedades del elemento <audio>

src

Ruta del archivo de audio.

```
audio.src = "musica.mp3";
```

currentTime

Tiempo actual de reproducción (en segundos).

```
console.log(audio.currentTime);
```

duration

Duración total del audio (en segundos).

```
console.log(audio.duration);
```

paused

Devuelve true si el audio está pausado.

```
console.log(audio.paused);
```

ended

Devuelve true si el audio ha terminado.

```
console.log(audio.ended);
```

volume

Nivel de volumen (de 0.0 a 1.0).

```
audio.volume = 0.5;
```

muted
Indica si el audio está silenciado.

```
audio.muted = true;
```

loop
Indica si el audio se reinicia automáticamente al finalizar.

```
audio.loop = true;
```

autoplay
Indica si debe empezar automáticamente.

```
audio.autoplay = true;
```

controls
Muestra los controles del navegador.

```
audio.controls = true;
```

preload
Sugerencia sobre si precargar: 'auto', 'metadata', 'none'.

```
audio.preload = "auto";
```

readyState
Indica cuánto del audio está listo para reproducirse.

```
console.log(audio.readyState);
```

buffered
Rangos de audio que ya se han cargado.

```
console.log(audio.buffered);
```

Métodos del objeto <audio>

play()
Reproduce el audio. Devuelve una promesa.

```
audio.play().then(() => {
  console.log("Audio en reproducción");
}).catch(err => {
  console.error("No se pudo reproducir:", err);
});
```

pause()

Pausa el audio.

```
audio.pause();
```

load()

Recarga el archivo de audio.

```
audio.load();
```

canPlayType()

Verifica si el tipo MIME puede reproducirse.

```
console.log(audio.canPlayType("audio/mpeg"));
```

addEventListener()

Escucha eventos como 'play', 'pause', 'ended'.

```
audio.addEventListener("ended", () => {
  console.log("Audio finalizado");
});
```

Lista Completa de Eventos en JavaScript (Actualizado a 2025)

JavaScript proporciona una gran variedad de eventos que permiten responder a interacciones del usuario, acciones del navegador, cambios de estado, entre otros. A continuación se presenta una lista clasificada de eventos por tipo de elemento o interacción, junto con ejemplos y su explicación.

Eventos de ratón

click

Disparo al hacer clic.

```
element.addEventListener('click', () => console.log('Clic hecho'));
```

dblclick

Doble clic del ratón.

```
element.addEventListener('dblclick', () => console.log('Doble clic'));
```

mousedown

Pulsar botón del ratón.

```
element.addEventListener('mousedown', () => console.log('Mouse down'));
```

mouseup

Soltar botón del ratón.

```
element.addEventListener('mouseup', () => console.log('Mouse up'));
```

mousemove

Mover el ratón sobre el elemento.

```
element.addEventListener('mousemove', () => console.log('Mouse move'));
```

mouseenter

Entrar en el área del elemento.

```
element.addEventListener('mouseenter', () => console.log('Mouse enter'));
```

mouseleave

Salir del área del elemento.

element.addEventListener('mouseleave', () => console.log('Mouse leave'));

Eventos de teclado

keydown

Presionar una tecla.

```
document.addEventListener('keydown', e => console.log(`Tecla presionada: ${e.key}`));
```

keyup

Soltar una tecla.

```
document.addEventListener('keyup', e => console.log(`Tecla soltada: ${e.key}`));
```

keypress

Presionar tecla (obsoleto en muchos navegadores, usar keydown).

```
document.addEventListener('keypress', e => console.log(e.key));
```

Eventos de formulario

submit

Cuando se envía un formulario.

```
form.addEventListener('submit', e => { e.preventDefault(); alert('Formulario enviado'); });
```

change

Cambio en input/select.

```
input.addEventListener('change', () => console.log('Valor cambiado'));
```

input

Mientras se escribe.

```
input.addEventListener('input', () => console.log('Escribiendo...'));
```

focus

Cuando un campo gana el foco.

```
input.addEventListener('focus', () => console.log('Foco en input'));
```

blur

Cuando un campo pierde el foco.

```
input.addEventListener('blur', () => console.log('Se salió del input'));
```

Eventos del DOM y ventana

load

Cuando el documento ha terminado de cargarse.

```
window.addEventListener('load', () => console.log('Página cargada'));
```

DOMContentLoaded

Cuando el DOM está listo (sin esperar imágenes).

```
document.addEventListener('DOMContentLoaded', () => console.log('DOM listo'));
```

resize

Cambio de tamaño de la ventana.

```
window.addEventListener('resize', () => console.log('Redimensionado'));
```

scroll

Cuando el usuario hace scroll.

```
window.addEventListener('scroll', () => console.log('Desplazamiento'));
```

unload

Cuando la página está a punto de cerrarse (limitado por seguridad).

```
window.addEventListener('unload', () => console.log('Saliendo de la página'));
```

Eventos multimedia

play

Reproducción iniciada.

```
video.addEventListener('play', () => console.log('Reproduciendo'));
```

pause

Reproducción pausada.

```
video.addEventListener('pause', () => console.log('Pausado'));
```

ended

Reproducción finalizada.

video.addEventListener('ended', () => console.log('Finalizado'));

volumechange

Cambio en el volumen.

audio.addEventListener('volumechange', () => console.log('Volumen cambiado'));

Explicaciones detalladas de los ejemplos de eventos

Evento: click

element.addEventListener('click', () => console.log('Clic hecho'));

• `addEventListener('click', ...)`: Escucha cuando el usuario hace clic sobre el elemento.
• `() => console.log(...)`: Función flecha que se ejecuta al ocurrir el evento.
• `console.log('Clic hecho')`: Imprime en la consola el mensaje cuando se hace clic.

Evento: keydown

document.addEventListener('keydown', e => console.log(`Tecla presionada: ${e.key}`));

• Se agrega un 'escuchador' de eventos al documento para detectar cuando se presiona una tecla.
• `e.key`: contiene la tecla presionada por el usuario.
• Se imprime en consola el nombre de la tecla.

Evento: submit

form.addEventListener('submit', e => { e.preventDefault(); alert('Formulario enviado'); });

• Escucha cuando se envía un formulario.
• `e.preventDefault()`: evita que la página se recargue.
• `alert(...)`: muestra un mensaje emergente.

Evento: DOMContentLoaded

document.addEventListener('DOMContentLoaded', () => console.log('DOM listo'));

• Escucha cuando todo el DOM está cargado (sin esperar imágenes o estilos externos).
• Imprime en consola un mensaje indicando que ya se puede manipular el DOM.

Evento: play

video.addEventListener('play', () => console.log('Reproduciendo'));

• Detecta cuando el usuario comienza a reproducir un video.
• Se imprime un mensaje indicando que el video está en reproducción.

Atributos Comunes de los Elementos en JavaScript (Actualizado a 2025)

En JavaScript, **los elementos HTML tienen múltiples atributos** que pueden ser accedidos y modificados usando sus propiedades **en objetos DOM**. A continuación se presenta una lista de atributos comunes con ejemplos y explicaciones actualizadas a 2025.

id

Identificador único del elemento.

```
const elem = document.getElementById("miElemento");
console.log(elem.id); // "miElemento"
```

className

Clases CSS aplicadas al elemento.

```
elem.className = "btn btn-primary";
```

innerHTML

Contenido HTML dentro del elemento.

```
elem.innerHTML = "<strong>Hola mundo</strong>";
```

textContent

Texto plano dentro del elemento.

```
elem.textContent = "Hola mundo";
```

value

Valor de inputs, textarea, select.

```
const input = document.querySelector("input");
input.value = "Nuevo valor";
```

type

Tipo de input (text, password, etc.).

```
input.type = "password";
```

placeholder

Texto de ayuda que aparece cuando está vacío.

```
input.placeholder = "Introduce tu nombre";
```

disabled

Desactiva el elemento.

```
input.disabled = true;
```

checked

Indica si un checkbox o radio está marcado.

```
const check = document.querySelector("input[type=checkbox]");
check.checked = true;
```

selected

Selecciona una opción en un select.

```
const option = document.querySelector("option");
option.selected = true;
```

style

Modifica estilos en línea del elemento.

```
elem.style.backgroundColor = "red";
```

src

Fuente de una imagen o video.

```
const img = document.querySelector("img");
img.src = "foto.jpg";
```

href

Enlaces de un <a>.

```
const link = document.querySelector("a");
link.href = "https://openai.com";
```

alt

Texto alternativo en imágenes.

```
img.alt = "Descripción de la imagen";
```

title

Texto que aparece como tooltip al pasar el cursor.

```
elem.title = "Información adicional";
```

name

Nombre del elemento (formulario, radio, input).

```
input.name = "usuario";
```

readonly

Hace un input de solo lectura.

```
input.readOnly = true;
```

required

Obliga a rellenar un campo antes de enviar el formulario.

```
input.required = true;
```

maxlength

Número máximo de caracteres permitidos.

```
input.maxLength = 10;
```

DOM en JavaScript - Propiedades, Métodos, Atributos y Eventos (Actualizado a 2025)

El DOM (Document Object Model) en JavaScript representa la estructura del documento HTML como una jerarquía de objetos que pueden ser accedidos, modificados o eliminados. A continuación se detallan sus propiedades, métodos, atributos y eventos más relevantes, actualizados hasta 2025, con ejemplos comentados.

Propiedades principales del DOM

document.title

Devuelve o establece el título de la pestaña.

```
// Obtener el título
console.log(document.title);

// Cambiar el título
document.title = "Nuevo título";
```

document.body

Referencia al <body> del documento.

```
document.body.style.backgroundColor = "lightblue";
```

document.documentElement

Referencia al elemento <html>.

```
console.log(document.documentElement.lang);
```

element.innerHTML

Modifica el contenido HTML interno del elemento.

```
const div = document.getElementById("contenido");
div.innerHTML = "<p>Texto <strong>nuevo</strong></p>";
```

element.textContent

Modifica el texto plano del elemento.

```
div.textContent = "Texto sin etiquetas HTML";
```

Métodos esenciales del DOM

getElementById()

Obtiene un elemento por su ID.

```
const titulo = document.getElementById("titulo");
titulo.style.color = "red";
```

getElementsByClassName()

Obtiene una colección de elementos por clase.

```
const items = document.getElementsByClassName("item");
console.log(items.length);
```

getElementsByTagName()

Devuelve todos los elementos con la etiqueta dada.

```
const parrafos = document.getElementsByTagName("p");
console.log(parrafos[0].textContent);
```

querySelector()

Devuelve el primer elemento que coincide con un selector CSS.

```
const input = document.querySelector("input[type='text']");
input.placeholder = "Nuevo texto";
```

querySelectorAll()

Devuelve todos los elementos que coinciden con un selector CSS.

```
const botones = document.querySelectorAll("button");
botones.forEach(b => b.disabled = false);
```

createElement()

Crea un nuevo elemento HTML.

```
const nuevoDiv = document.createElement("div");
nuevoDiv.textContent = "Nuevo elemento creado";
```

appendChild()

Agrega un nuevo nodo como hijo.

```
document.body.appendChild(nuevoDiv);
```

removeChild()

Elimina un nodo hijo.

```
document.body.removeChild(nuevoDiv);
```

setAttribute()

Establece un atributo en un elemento.

```
input.setAttribute("type", "email");
```

getAttribute()

Obtiene el valor de un atributo.

```
console.log(input.getAttribute("type"));
```

Eventos del DOM comunes

DOMContentLoaded

Se dispara cuando el DOM ha sido cargado completamente.

```
document.addEventListener("DOMContentLoaded", () => {
  console.log("El DOM está completamente cargado");
});
```

click

Se dispara al hacer clic sobre un elemento.

```
const btn = document.querySelector("button");
btn.addEventListener("click", () => {
  alert("Botón clicado");
});
```

input

Se lanza cuando cambia el valor de un campo input.

```
const input = document.querySelector("input");
input.addEventListener("input", () => {
  console.log("Escribiendo...");
});
```

change

Se lanza cuando se cambia y se sale del campo.

```
input.addEventListener("change", () => {
  console.log("Valor cambiado");
});
```

Lectura y recogida de arrays desde el DOM

```
// Recoger valores de inputs con clase "campo"
const campos = document.querySelectorAll(".campo");
const valores = Array.from(campos).map(input => input.value);
console.log(valores); // muestra los valores de todos los inputs
```

Aplicación de estilos CSS desde JavaScript

```
// Aplicar estilo Flexbox a un contenedor
const contenedor = document.getElementById("contenedor");
contenedor.style.display = "flex";
contenedor.style.justifyContent = "center";
contenedor.style.alignItems = "center";

// Aplicar estilo Grid
contenedor.style.display = "grid";
contenedor.style.gridTemplateColumns = "repeat(3, 1fr)";
contenedor.style.gap = "10px";
```

Manipulación de tablas con JavaScript

```
// Añadir una fila a una tabla
const tabla = document.getElementById("miTabla");
const fila = tabla.insertRow();
const celda1 = fila.insertCell(0);
const celda2 = fila.insertCell(1);
celda1.textContent = "Nombre";
celda2.textContent = "Edad";
```

Lectura de ficheros locales con File API

```
// HTML: <input type="file" id="archivo" />
document.getElementById("archivo").addEventListener("change", (e) => {
  const archivo = e.target.files[0];
  const lector = new FileReader();
  lector.onload = () => {
    console.log("Contenido del archivo:", lector.result);
  };
  lector.readAsText(archivo);
});
```

Manipulación de clases CSS

```
// Agregar clase
element.classList.add("resaltado");

// Quitar clase
element.classList.remove("resaltado");

// Alternar clase
element.classList.toggle("activo");
```

Modificación de variables CSS

```
// CSS: :root { --color-principal: blue; }
document.documentElement.style.setProperty("--color-principal", "green");
```

Elementos HTML en JavaScript - Propiedades, Métodos, Atributos y Eventos (Actualizado a 2025)

Propiedades comunes de elementos HTML

id

Devuelve o establece el ID del elemento.

```
const elem = document.getElementById("demo");
console.log(elem.id); // "demo"
```

className

Devuelve o establece el nombre de clase del elemento.

```
elem.className = "activo"; // aplica la clase CSS "activo" al elemento
```

innerHTML

Devuelve o establece el HTML interno del elemento.

```
elem.innerHTML = "<strong>Hola</strong>"; // inserta HTML
```

textContent

Devuelve o establece el texto sin formato.

```
elem.textContent = "Solo texto"; // sin etiquetas HTML
```

style

Accede a los estilos CSS en línea del elemento.

```
elem.style.color = "blue"; // cambia el color del texto a azul
```

Métodos comunes de elementos HTML

setAttribute()

Establece un atributo en el elemento.

```
elem.setAttribute("title", "Tooltip"); // agrega un tooltip
```

getAttribute()

Obtiene el valor de un atributo.

```
console.log(elem.getAttribute("title")); // muestra "Tooltip"
```

removeAttribute()

Elimina un atributo del elemento.

```
elem.removeAttribute("title");
```

appendChild()

Agrega un nuevo nodo como hijo.

```
const nuevo = document.createElement("p");
nuevo.textContent = "Nuevo párrafo";
document.body.appendChild(nuevo);
```

remove()

Elimina el elemento del DOM.

```
elem.remove(); // borra el elemento
```

Eventos comunes en elementos HTML

click

Se activa cuando se hace clic en el elemento.

```
elem.addEventListener("click", () => {
  console.log("Elemento clicado");
});
```

mouseover

Se activa cuando el puntero pasa por encima del elemento.

```
elem.addEventListener("mouseover", () => {
  elem.style.backgroundColor = "yellow";
});
```

focus

Se activa cuando un input recibe foco.

```
input.addEventListener("focus", () => {
  input.style.border = "2px solid blue";
});
```

blur

Se activa cuando un input pierde el foco.

```
input.addEventListener("blur", () => {
  input.style.border = "";
});
```

Gestión de clases con classList

classList.add()

Agrega una clase.

```
elem.classList.add("nuevo-estilo");
```

classList.remove()

Quita una clase.

```
elem.classList.remove("nuevo-estilo");
```

classList.toggle()

Agrega o quita una clase según si ya existe.

```
elem.classList.toggle("activo");
```

classList.contains()

Verifica si el elemento tiene una clase.

```
if (elem.classList.contains("visible")) {
  console.log("Está visible");
}
```

Objeto Window en JavaScript - Propiedades, Métodos y Eventos (Actualizado a 2025)

Propiedades importantes del objeto window

window.innerWidth / innerHeight

Devuelven el tamaño interno de la ventana.

```
console.log(window.innerWidth);  // Ancho en píxeles
console.log(window.innerHeight); // Alto en píxeles
```

window.location

Proporciona la URL actual y permite redirigir.

```
console.log(window.location.href); // Muestra la URL actual
window.location.href = "https://openai.com"; // Redirige a otra URL
```

window.document

Referencia al objeto document.

```
console.log(window.document.title); // Muestra el título de la página
```

window.navigator

Contiene información del navegador.

```
console.log(window.navigator.userAgent); // Tipo y versión del navegador
```

window.screen

Contiene información sobre la pantalla del usuario.

```
console.log(window.screen.width);  // Ancho total de la pantalla
console.log(window.screen.height); // Alto total de la pantalla
```

Métodos del objeto window

alert()

Muestra un cuadro de alerta al usuario.

```
window.alert("¡Hola desde alert!");
```

confirm()

Muestra una ventana de confirmación (Aceptar/Cancelar).

```
if (window.confirm("¿Estás seguro?")) {
  console.log("Confirmado");
} else {
  console.log("Cancelado");
}
```

prompt()

Solicita al usuario una entrada de texto.

```
const nombre = window.prompt("¿Cuál es tu nombre?");
console.log("Hola, " + nombre);
```

setTimeout()

Ejecuta una función después de un retraso.

```
setTimeout(() => {
  console.log("Pasaron 2 segundos");
}, 2000);
```

setInterval()

Ejecuta una función repetidamente cada cierto tiempo.

```
const intervalo = setInterval(() => {
  console.log("Cada 3 segundos");
}, 3000);
```

clearInterval()

Detiene una función que se ejecuta con setInterval.

```
clearInterval(intervalo);
```

open()

Abre una nueva ventana o pestaña.

```
window.open("https://example.com", "_blank");
```

close()

Cierra la ventana actual (solo si fue abierta con window.open).

```
// ventana.close(); (requiere ser abierto por JS)
```

Eventos comunes del objeto window

load

Se dispara cuando la página ha cargado completamente.

```
window.addEventListener("load", () => {
  console.log("Página completamente cargada");
});
```

resize

Se dispara al cambiar el tamaño de la ventana.

```
window.addEventListener("resize", () => {
  console.log("Nuevo tamaño:", window.innerWidth, window.innerHeight);
});
```

scroll

Se dispara al hacer scroll.

```
window.addEventListener("scroll", () => {
  console.log("Desplazamiento detectado");
});
```

beforeunload

Se dispara antes de abandonar la página (para advertencias).

```
window.addEventListener("beforeunload", (e) => {
  e.preventDefault();
  e.returnValue = ""; // Algunos navegadores muestran una alerta
});
```

Tipos de Datos en JavaScript

JavaScript admite varios tipos de datos que pueden clasificarse en primitivos, objetos, estructuras de colección y asíncronas. La siguiente tabla describe todos los tipos de datos conocidos y estandarizados hasta ES2023.

Tipo	Categoría	Descripción
undefined	Primitivo	Valor por defecto de una variable no inicializada.
null	Primitivo	Valor intencionalmente vacío (ausencia de valor).
boolean	Primitivo	Solo puede ser true o false.
number	Primitivo	Números enteros o decimales.
bigint	Primitivo	Números enteros muy grandes, usando una 'n' al final (ej. 123n).
string	Primitivo	Cadena de texto entre comillas.
symbol	Primitivo	Valor único e inmutable para usar como identificador.
object	Estructurado	Colección de pares clave-valor (puede incluir arrays, funciones, etc.).
function	Subtipo de objeto	Bloque reutilizable de código que se puede llamar.
array	Subtipo de objeto	Lista ordenada de valores indexados numéricamente.
date	Objeto estándar	Representa una fecha y hora con métodos para manipularlas.
regexp	Objeto estándar	Expresiones regulares para buscar patrones en cadenas.
map	Estructura clave-valor	Colección ordenada con claves de cualquier tipo.
set	Colección única	Colección de valores únicos no duplicados.
weakmap	Mapa débil	Mapa donde las claves son objetos recogibles por el recolector de basura.
weakset	Set débil	Set que solo puede contener objetos y no se puede iterar.
typed array	Array tipado	Representaciones binarias de arrays de números con tipos específicos.
array buffer	Binario crudo	Contenedor genérico para datos binarios en memoria.
promise	Asíncrono	Representa un valor que puede estar disponible ahora, en el futuro o nunca.
error	Objeto estándar	Representa errores lanzados por el código o por el motor de JS.
class	Estructura	Sintaxis de v para crear objetos y manejar herencia.

Guía para Conectar HTML y JavaScript con MySQL usando Node.js (Actualizado a 2025)

Este documento explica paso a paso cómo conectar un archivo HTML con una base de datos MySQL usando JavaScript y Node.js mediante una API REST segura y moderna.

Requisitos previos

- Tener instalado Node.js desde **https://nodejs.org**
- **Tener una base de datos MySQL funcionando con una tabla llamada `clientes`, por ejemplo**.

Crear proyecto

1. Crea una carpeta para el proyecto, por ejemplo:

C:\proyectos\api-clientes

2. Abre una consola o terminal y navega a esa carpeta:

cd C:\proyectos\api-clientes

3. Inicializa un proyecto Node.js:

npm init -y

4. Instala las dependencias necesarias:

npm install express mysql cors body-parser

Código del servidor: server.js

```
const express = require('express');
const mysql = require('mysql');
const cors = require('cors');
```

```
const bodyParser = require('body-parser');

const app = express();
app.use(cors());
app.use(bodyParser.json());

const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'clientes_db'
});

db.connect(err => {
  if (err) throw err;
  console.log('Conectado a MySQL');
});

// Obtener todos los clientes
app.get('/api/clientes', (req, res) => {
  db.query('SELECT * FROM clientes', (err, result) => {
    if (err) throw err;
    res.json(result);
  });
});

// Insertar nuevo cliente
app.post('/api/clientes', (req, res) => {
  const { nombre, email } = req.body;
  db.query('INSERT INTO clientes (nombre, email) VALUES (?, ?)', [nombre, email], (err, result) => {
    if (err) throw err;
    res.json({ id: result.insertId, nombre, email });
  });
});

app.listen(3000, () => console.log('Servidor en puerto 3000'));
```

Código HTML del cliente

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Clientes</title>
</head>
<body>
<h1>Alta de Clientes</h1>
<input id="nombre" placeholder="Nombre">
<input id="email" placeholder="Email">
<button onclick="crearCliente()">Guardar</button>
<ul id="listaClientes"></ul>

<script>
  function crearCliente() {
    const nombre = document.getElementById("nombre").value;
    const email = document.getElementById("email").value;

    fetch('http://localhost:3000/api/clientes', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ nombre, email })
    })
    .then(res => res.json())
    .then(cliente => {
      alert("Cliente guardado: " + cliente.nombre);
      cargarClientes();
    });
  }

  function cargarClientes() {
    fetch('http://localhost:3000/api/clientes')
      .then(res => res.json())
      .then(clientes => {
        const lista = document.getElementById("listaClientes");
        lista.innerHTML = "";
        clientes.forEach(c => {
          const li = document.createElement("li");
```

```
        li.textContent = `${c.nombre} (${c.email})`;
        lista.appendChild(li);
      });
    });
  }

  cargarClientes();
</script>
</body>
</html>
```

Resultado esperado

- Una API REST que permite crear y listar clientes.
- Una página web que se conecta a esa API para mostrar y añadir datos.
- Seguridad garantizada al evitar conexiones directas a MySQL desde el navegador.

Uso de fetch() en JavaScript para Comunicaciones HTTP (Actualizado a 2025)

`fetch()` es una función integrada en JavaScript que permite realizar peticiones HTTP asíncronas. Es una forma moderna de interactuar con APIs REST y servidores web sin recargar la página.

Sintaxis básica

```
fetch(url, opciones)
  .then(respuesta => respuesta.json())
  .then(data => {
    // usar datos
  })
  .catch(error => {
    // manejar error
  });
```

Ejemplo: Obtener datos con GET

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(res => res.json()) // convierte respuesta a JSON
  .then(datos => {
    console.log(datos); // muestra los datos recibidos
  })
  .catch(error => {
    console.error("Error:", error); // muestra error si ocurre
  });
```

Ejemplo: Enviar datos con POST

```
fetch('https://mi-api.com/clientes', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ nombre: "Ana", edad: 30 })
```

```
})
.then(res => res.json())
.then(data => {
  console.log("Guardado:", data); // respuesta del servidor
})
.catch(error => {
  console.error("Error:", error);
});
```

Explicación de cada parte

Elemento	Descripción
fetch()	Función para iniciar la petición.
url	Dirección del recurso o API.
method	Tipo de operación: GET, POST, etc.
headers	Indica el tipo de datos que se envían o reciben.
body	Datos enviados al servidor (solo en POST/PUT).
.then()	Ejecuta una función cuando se resuelve la promesa.
.catch()	Captura errores si la petición falla.

Ventajas de usar fetch()

- Es nativo en los navegadores modernos.
- Basado en promesas: más limpio y fácil de usar que XMLHttpRequest.
- Ideal para trabajar con APIs REST y servicios web.

url

Especifica la dirección del servidor o recurso al que queremos acceder.

```
// URL del servidor donde haremos la solicitud
const url = "https://api.ejemplo.com/usuarios";
```

method

Indica el tipo de operación HTTP que se realizará: GET, POST, PUT, DELETE, etc.

```
// Método POST para enviar datos
const opciones = {
```

```
  method: "POST"
};
```

headers

Son metadatos que se envían junto a la solicitud. Por ejemplo, para indicar el tipo de contenido.

```
// Especificamos que el contenido será JSON
const opciones = {
  headers: {
    "Content-Type": "application/json"
  }
};
```

body

Contiene los datos que se enviarán al servidor. Solo se usa con métodos como POST o PUT.

```
// Convertimos un objeto JavaScript a JSON
const opciones = {
  body: JSON.stringify({
    nombre: "Carlos",
    edad: 40
  })
};
```

.then()

Se ejecuta cuando la respuesta del servidor ha sido recibida correctamente.

```
fetch(url, opciones)
  .then(respuesta => respuesta.json()) // convertimos la respuesta a JSON
  .then(data => {
    console.log("Datos recibidos:", data); // procesamos los datos
  });
```

catch()

Se ejecuta si ocurre un error durante la petición.

```
fetch(url, opciones)
  .then(res => res.json())
  .catch(error => {
    console.error("Ocurrió un error:", error); // muestra el error
  });
```