Introducción al Desarrollo Web

Ing. Marco Aedo López

EPIS UNSA – 2025 maedol@unsa.edu.pe marcoaedo@hotmail.com

Fundamentos del desarrollo Frontend

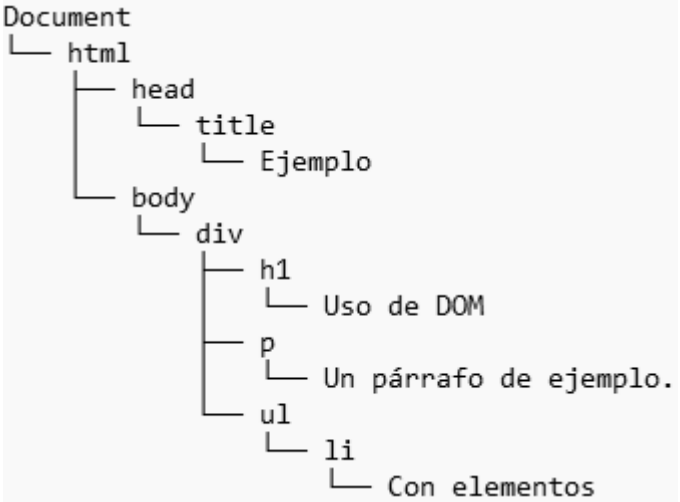
JavaScript

Tema 6

29. Manipulación del DOM

- DOM (Document Object Model) es la representación estructurada del contenido de una página web (documento HTML) en forma de árbol de nodos
- Cada elemento HTML se convierte en un nodo que JavaScript puede manipular

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <div>
      <h1>Uso de DOM</h1>
      <p>
        Un párrafo de ejemplo.
      </p>
      <ul>
          <li>Con elementos</li>
      </ul>
    </div>
  </body>
</html>
```



29. Manipulación del DOM

- Manipular el DOM significa **leer, modificar o crear elementos** dinámicamente usando JavaScript. Esto incluye cambiar texto, atributos, estilos, o incluso añadir y eliminar nodos
- El punto de entrada al DOM en JavaScript es el objeto global `document`. A través de él podemos seleccionar y manipular los elementos del HTML

```html
<h1 id="titulo">Hola Mundo</h1>
<p id="parrafo">Este es un párrafo inicial.</p>
<button id="boton">Cambiar texto</button>
<script src="dom.js"></script>
```


```js
// Crear un nuevo nodo <p>
const nuevoP = document.createElement("p");
nuevoP.textContent = "Soy un párrafo creado con JS";

// Insertar en el DOM
document.body.appendChild(nuevoP);
```

29. Manipulación del DOM

```js
// 1. Crear elemento
const card = document.createElement("div");
card.className = "tarjeta";

// 2. Añadir contenido
const titulo = document.createElement("h2");
titulo.textContent = "Producto nuevo";

const descripcion = document.createElement("p");
descripcion.textContent = "Descripción del producto";

// 3. Armar jerarquía
card.appendChild(titulo);
card.appendChild(descripcion);

// 4. Insertar en el DOM
document.body.appendChild(card);
```

```html
<!DOCTYPE html> == $0
<html lang="en">
 ▶ <head> … </head>
 ▼ <body>
    <script src="script.js"></script>
   ▼ <div class="tarjeta">
      <h2>Producto nuevo</h2>
      <p>Descripción del producto</p>
    </div>
  </body>
</html>
```

29. Manipulación del DOM

Seleccionar Elementos		
Método	**Descripción**	**Ejemplo**
getElementById	Devuelve un elemento por su id	document.getElementById(“contenedor”)
getElementsByClassName	Devuelve todos los elementos de una clase (HTMLCollection)	document.getElementsByClassName(“texto”)
getElementsByTagName	Devuelve todos los elementos de una etiqueta	document.getElementsByTagName(“p”)
querySelector	Devuelve el primer elemento que cumpla con un selector tipo CSS indicado	document.querySelector(“.texto”)
querySelectorAll	Devuelve todos los elementos que cumplan con un selector tipo CSS (NodeList)	document.querySelectorAll(“div p”)

29. Manipulación del DOM

```html
<h1 id="titulo">Hola Mundo</h1>
<p id="parrafo">Este es un párrafo inicial.</p>
<p class="parrafoEspecial">Este es un párrafo especial.</p>
<button id="boton">Cambiar texto</button>
<script src="dom.js"></script>
```

```js
const unParrafo = document.getElementById("parrafo");
const otroParrafo = document.getElementsByClassName("parrafoEspecial");
console.log(unParrafo);  // Muestra el nodo <p>
console.log(otroParrafo);  // Muestra el nodo <p>
```

```
                                                        dom.js:16
 <p id="parrafo">Este es un párrafo inicial.</p>
▼ HTMLCollection [p.parrafoEspecial] i                  dom.js:17
  ▶ 0: p.parrafoEspecial
    length: 1
  ▶ [[Prototype]]: HTMLCollection
```

Leer Contenido de Elementos

Tipo de contenido a leer	Método/Propiedad	Ejemplo	Resultado
Texto visible en la página	.textContent	document.getElementById("titulo").textContent	Devuelve todo el texto dentro del elemento (aunque esté oculto por CSS)
Texto interpretado por el navegador (incluye etiquetas HTML internas)	.innerHTML	document.getElementById("parrafo").innerHTML	Devuelve el HTML interno del elemento
Valor de un campo de formulario	.value	document.getElementById("nombre").value	Devuelve lo que el usuario escribió en un <input>, <textarea>, <select>
Atributo específico	.getAttribute("atributo")	document.getElementById("link").getAttribute("href")	Devuelve el valor de un atributo como href, src, alt, etc.
Colección de atributos	.attributes	document.getElementById("imagen").attributes	Lista de todos los atributos del elemento

29. Manipulación del DOM

```html
<h1 id="titulo">Bienvenido</h1>
<p id="parrafo">Este es un <strong>ejemplo</strong> de lectura.</p>
<input id="nombre" type="text" value="Juancito">
<a id="link" href="https://www.google.com">Visítanos</a>

<script src="domLeer.js"></script>
```

```js
// Leer contenido de un título
console.log(document.getElementById("titulo").textContent);

// Leer con innerHTML
console.log(document.getElementById("parrafo").innerHTML);

// Leer valor de input
console.log(document.getElementById("nombre").value);

// Leer atributo
console.log(document.getElementById("link").getAttribute("href"));
```

```
Bienvenido
Este es un <strong>ejemplo</strong> de lectura.
Juancito
https://www.google.com
```

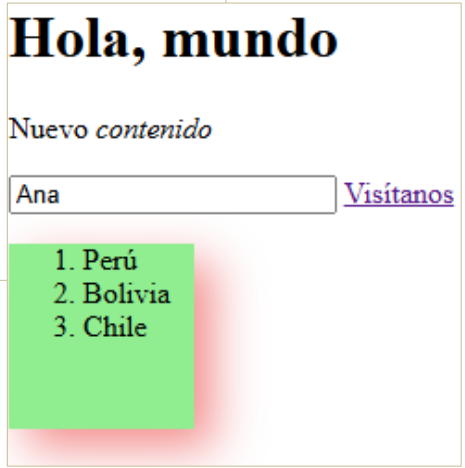
Modificar Contenido de Elementos

Tipo de contenido a modificar	Método/Propiedad	Ejemplo	Efecto en la página
Texto visible	.textContent	document.getElementById("titulo").textContent = "Nuevo título";	Cambia solo el texto plano (ignora etiquetas HTML)
HTML interno	.innerHTML	document.getElementById("parrafo").innerHTML = "Este es un nuevo párrafo.";	Permite insertar texto + etiquetas HTML
Valor de un campo de formulario	.value	document.getElementById("nombre").value = "Ana";	Modifica lo que aparece en <input>, <textarea>, <select>
Atributo específico	.setAttribute("atributo", "valor")	document.getElementById("link").setAttribute("href", "https://www.unsa.edu.pe");	Cambia el valor de un atributo como src, href, alt
Estilos en línea	.style.propiedad	document.getElementById("caja").style.backgroundColor = "lightblue";	Aplica un estilo directamente al elemento
Clases CSS	.classList.add(), .classList.remove(), .classList.toggle()	document.getElementById("caja").classList.add("resaltado");	Agrega o quita clases para cambiar estilos dinámicamente

```html
<h1 id="titulo">Bienvenido</h1>
<p id="parrafo">Este es un <strong>ejemplo</strong> de lectura.</p>
<input id="nombre" type="text" value="Juancito">
<a id="link" href="https://www.google.com">Visítanos</a>

<div id="caja"">
    <ol>
        <li>Perú</li>
        <li>Bolivia</li>
        <li>Chile</li>
    </ol>
</div>

<script src="domLeer.js"></script>
```

```css
#caja {
    width: 100px;
    height: 100px;
    background: gray;
}

.sombra {
    box-shadow: 10px 10px 25px rgba(231, 23, 23, 0.5);
}
```

```js
// Modificar con textContent
document.getElementById("titulo").textContent = "Hola, mundo";

// Modificar con HTML
document.getElementById("parrafo").innerHTML = "Nuevo <em>contenido</em>";

// Cambiar valor de input
document.getElementById("nombre").value = "Ana";

// Cambiar atributo
document.getElementById("link").setAttribute("href", "https://developer.mozilla.org");

// Cambiar estilo
document.getElementById("caja").style.backgroundColor = "lightgreen";

// Cambiar clases
document.getElementById("caja").classList.add("sombra");
```



```html
<h1 id="titulo">Métodos de selección DOM</h1>

<p class="info">Primer párrafo con clase "info".</p>
<p class="info">Segundo párrafo con clase "info".</p>

<ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
</ul>

<ol>
  <li>Elemento 4</li>
  <li>Elemento 5</li>
  <li>Elemento 6</li>
</ol>

<button id="accion">Ejecutar ejemplos</button>

<script src="script.js"></script>
```

```css
.resaltado {
    color: red;
    font-weight: bold;
}
```

Métodos de selección DOM

Modificado con getElementsByClassName

Segundo párrafo con clase "info".

- Elemento 1
- Elemento 2
- Elemento modificado por etiqueta

1. Elemento 4
2. Elemento 5
3. Elemento 6

Ejecutar ejemplos

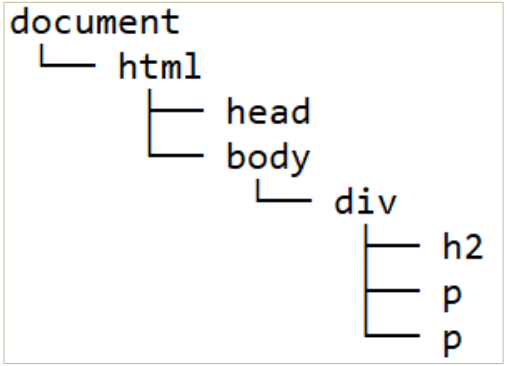
```javascript
// getElementById
const titulo = document.getElementById("titulo");
console.log("getElementById: ", titulo.textContent);
titulo.style.color = "blue";

// getElementsByClassName
const parrafos = document.getElementsByClassName("info");
console.log("getElementsByClassName: ", parrafos);
parrafos[0].textContent = "Modificado con getElementsByClassName";

// getElementsByTagName
const lista = document.getElementsByTagName("li");
console.log("getElementsByTagName: ", lista);
lista[2].textContent = "Elemento modificado por etiqueta";

// querySelector
const primerParrafo = document.querySelector(".info");
console.log("querySelector: ", primerParrafo);
primerParrafo.classList.add("resaltado");

// querySelectorAll
const todosLosLi = document.querySelectorAll("ul li");
console.log("querySelectorAll: ", todosLosLi);
todosLosLi.forEach(li => li.style.backgroundColor = "lightyellow");
```

```
getElementById:  Métodos de selección DOM
getElementsByClassName:  ▸ HTMLCollection(2) [p.info, p.info]
getElementsByTagName:
 ▸ HTMLCollection(6) [li, li, li, li, li, li]
querySelector:  ▸ p.info
querySelectorAll:  ▸ NodeList(3) [li, li, li]
```

29. Manipulación del DOM

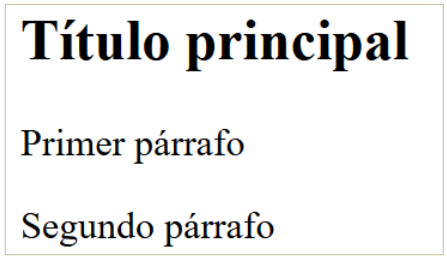
- Recorrido del DOM es el proceso de navegar por la estructura jerárquica del documento HTML desde un nodo específico
- Cada elemento del HTML se convierte en un nodo, y estos se relacionan entre sí como padres, hijos y hermanos

```
<body>
  <div id="contenedor">
    <h2>Título</h2>
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
  </div>
</body>
```



Propiedades principales para recorrer el DOM		
Propiedad	**Devuelve**	**Descripción**
parentNode	Nodo padre	Devuelve el nodo padre del elemento actual
childNodes	NodeList	Devuelve todos los nodos hijos, incluidos los de tipo texto (espacios, saltos de línea)
children	HTMLCollection	Devuelve sólo los nodos **elemento**, sin los nodos de texto
firstChild / lastChild	Nodo	Devuelve el primer o último hijo, que puede ser un nodo de texto
firstElementChild / lastElementChild	Elemento	Devuelve el primer o último hijo elemento (ignora texto o comentarios)
nextSibling / previousSibling	Nodo	Devuelve el hermano siguiente o anterior (puede ser texto o comentario)
nextElementSibling / previousElementSibling	Elemento	Devuelve sólo los hermanos tipo elemento

```html
<div id="contenedor">
    <h2>Título principal</h2>
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
</div>
```




```
PADRE DE contenedor:  BODY
HIJOS DE contenedor:  ▸ HTMLCollection(3) [h2, p, p]
 - H2 : Título principal
 - P : Primer párrafo
 - P : Segundo párrafo
PRIMER HIJO:  H2
ÚLTIMO HIJO:  P
Primer párrafo:  Primer párrafo
Siguiente hermano de primer <p>:  Segundo párrafo
Hermano anterior de segundo <p>:  Primer párrafo
RECORRIDO COMPLETO:
H2 :  Título principal
P :  Primer párrafo
P :  Segundo párrafo
```

```javascript
// Seleccionamos el contenedor principal
const contenedor = document.getElementById("contenedor");

// Mostrar el nodo padre
console.log("PADRE DE contenedor: ", contenedor.parentNode.nodeName);

// Mostrar los hijos directos
console.log("HIJOS DE contenedor: ", contenedor.children);
for (let hijo of contenedor.children) {
    console.log(" -", hijo.nodeName, ":", hijo.textContent);
}

// Primer y último hijo elemento
console.log("PRIMER HIJO: ", contenedor.firstElementChild.nodeName);
console.log("ÚLTIMO HIJO: ", contenedor.lastElementChild.nodeName);

// Recorrer entre hermanos
const primerParrafo = contenedor.firstElementChild.nextElementSibling;
console.log("Primer párrafo: ", primerParrafo.textContent);

const segundoParrafo = primerParrafo.nextElementSibling;
console.log("Siguiente hermano de primer <p>: ", segundoParrafo.textContent);

console.log("Hermano anterior de segundo <p>: ",
            segundoParrafo.previousElementSibling.textContent);

// Recorrido descendente completo
console.log("RECORRIDO COMPLETO: ");
for (let hijo of contenedor.children) {
    console.log(hijo.nodeName, ": ", hijo.textContent);
}
```

29. Manipulación del DOM

- Crear y eliminar elementos dinámicamente es clave para construir interfaces interactivas
- **Crear**: el proceso siempre sigue estos pasos:
  - Crear el nodo con `document.createElement("elemento")`
  - Configurar su contenido o atributos (`textContent`, `id`, `setAttribute()`, `classList.add()`, `innerHTML`, etc.)
  - Insertarlo en el DOM con métodos como `.appendChild()`, `.prepend()`, `before/after`, `.insertBefore()`, `.append()`
  - Además de elementos, se pueden crear nodos de texto o comentarios con `.createTextNode()`, `.createComment()`

29. Manipulación del DOM

- **Eliminar**: hay varias formas de eliminar nodos del DOM:
  - Elimina un nodo hijo desde su elemento padre `contenedor.removeChild(hijo)`
  - Elimina directamente el elemento sobre el que se invoca (más moderno y práctico) `elemento.remove()`
  - Se puede reemplazar un nodo hijo por otro `contenedor.replaceChild(nuevoHijo, antiguoHijo)`

```html
<div id="contenedor"></div>
<p id="parrafo">Este es un parrafo que será eliminado</p>

<ol id="lista">
    <li id="primer">España</li>
    <li id="segundo">Alemania</li>
    <li id="tercero">Inglaterra</li>
</ol>
```

```css
.resaltado {
  background-color: rgb(122, 122, 14);
  font-weight: bold;
  color: rgb(244, 233, 233);
  padding: 5px;
  border-radius: 8px;
}
```

Este es un parrafo que será eliminado

1. España
2. Alemania
3. Inglaterra

```html
<div id="contenedor"></div>
<p id="parrafo">Este es un parrafo que será eliminado</p>

<ol id="lista">
    <li id="primer">España</li>
    <li id="segundo">Alemania</li>
    <li id="tercero">Inglaterra</li>
</ol>

<script src="domCrearEliminar.js"></script>
```

Este es un parrafo que será eliminado

1. España
2. Alemania
3. Inglaterra

```javascript
// 1. Crear un nuevo párrafo
const nuevoParrafo = document.createElement("p");

// 2. Configurar contenido y atributos
nuevoParrafo.textContent = "Soy un párrafo creado con JavaScript";
nuevoParrafo.classList.add("resaltado");

// 3. Insertarlo dentro del contenedor
document.getElementById("contenedor").appendChild(nuevoParrafo);

// Eliminar un elemento hijo
const parrafo = document.getElementById("parrafo");
parrafo.remove();

const lista = document.getElementById("lista");
const paisEliminar = document.getElementById("tercero");
lista.removeChild(paisEliminar);
```

Soy un párrafo creado con JavaScript

1. España
2. Alemania

30. Eventos

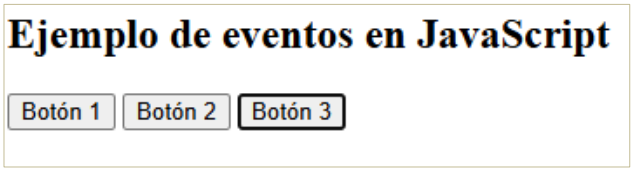
- La manipulación del DOM suele ir acompañada de eventos (clics, teclado, formularios, etc.)
- Es cualquier acción que ocurre en la página web y que el navegador puede detectar:
  - Un usuario hace clic en un botón
  - Un usuario escribe en un campo de texto
  - La página termina de cargarse
  - El puntero del mouse pasa por encima de un elemento
- JavaScript permite **escuchar** esos eventos y responder con una acción, lo que hace la web **interactiva**

30. Eventos

Evento	Se dispara cuando...
click	El usuario hace clic en un elemento
dblclick	Doble clic en un elemento
mouseover	El puntero entra en un elemento
mouseout	El puntero sale del elemento
keydown	El usuario presiona una tecla
keyup	El usuario suelta una tecla
input	Cambia el contenido de un campo de texto
submit	Se envía un formulario
load	La página o recurso termina de cargarse

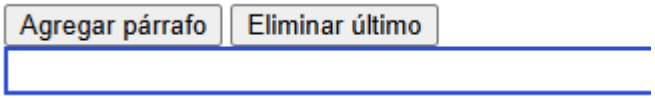
```html
<h2>Ejemplo de eventos en JavaScript</h2>

<!-- 1. Evento en línea -->
<button onclick="alert('Evento en línea: Hiciste clic en el Botón 1')">
    Botón 1
</button>

<!-- 2. Evento con propiedad del DOM -->
<button id="btn2">Botón 2</button>

<!-- 3. Evento con addEventListener -->
<button id="btn3">Botón 3</button>

<script src="eventos.js"></script>
```

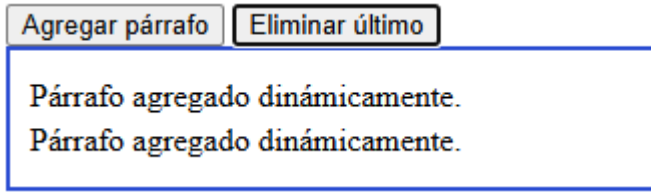
```javascript
// 2. Evento con propiedad del DOM
const boton2 = document.getElementById("btn2");
boton2.onclick = function() {
  alert("Evento con propiedad del DOM: Hiciste clic en el Botón 2");
};

// 3. Evento con addEventListener
const boton3 = document.getElementById("btn3");
boton3.addEventListener("click", () => {
  alert("Evento con addEventListener: Hiciste clic en el Botón 3");
});
```



```html
<button id="agregar">Agregar párrafo</button>
<button id="eliminar">Eliminar último</button>
<div id="contenedor"></div>
```

```css
#contenedor {
    padding: 10px;
    border: 2px solid #2b4dd2;
}
p {
    margin: 4px 0;
}
```






```javascript
const contenedor = document.getElementById("contenedor");
const btnAgregar = document.getElementById("agregar");
const btnEliminar = document.getElementById("eliminar");

btnAgregar.addEventListener("click", () => {
    const nuevo = document.createElement("p");
    nuevo.textContent = "Párrafo agregado dinámicamente.";
    contenedor.appendChild(nuevo);
});

btnEliminar.addEventListener("click", () => {
    const ultimo = contenedor.lastElementChild;
    if (ultimo)
        contenedor.removeChild(ultimo);
});
```

30. Eventos

- Objeto **event**
- Cada vez que ocurre un evento, JavaScript crea un objeto event con información útil

```
boton.addEventListener("click", (event) => {
  console.log("Tipo de evento:", event.type);
  console.log("Elemento origen:", event.target);
});
```

Propiedad	Descripción
type	Tipo de evento (ej. "click", "input")
target	Elemento que generó el evento
clientX, clientY	Posición del cursor
key, code	Tecla presionada (para eventos de teclado)
preventDefault()	Evita la acción por defecto (ej. enviar un formulario)
stopPropagation()	Detiene la propagación del evento a elementos padres

```html
<h2>Ejemplo del objeto <code>event</code></h2>

<button id="btn1">Haz clic aquí</button>
<button id="btn2">Otro botón</button>

<div id="info">Información del evento aparecerá aquí...</d>
```

```css
body {
    font-family: sans-serif;
    text-align: center;
    margin-top: 40px;
}
button {
    padding: 10px 20px;
    margin: 10px;
    font-size: 16px;
    cursor: pointer;
}
#info {
    margin-top: 20px;
    padding: 10px;
    border: 2px dashed #15c918;
    background: #ece5e5;
    width: 60%;
    margin-left: auto;
    margin-right: auto;
    text-align: left;
}
```

```javascript
const info = document.getElementById("info");

function mostrarInfo(event) {
    info.innerHTML = `
    <strong>Tipo de evento:</strong> ${event.type} <br>
    <strong>Elemento origen:</strong> ${event.target.tagName} (id="${event.target.id}") <br>
    <strong>Coordenadas:</strong> X=${event.clientX}, Y=${event.clientY} <br>
    <strong>Botón del mouse:</strong> ${event.button} <br>
    <strong>Tiempo desde carga (ms):</strong> ${event.timeStamp.toFixed(0)}`;
}

document.getElementById("btn1").addEventListener("click", mostrarInfo);
document.getElementById("btn2").addEventListener("click", mostrarInfo);
```

Ejemplo del objeto event

Haz clic aquí | Otro botón

Tipo de evento: click
Elemento origen: BUTTON (id="btn1")
Coordenadas: X=844, Y=99
Botón del mouse: 0
Tiempo desde carga (ms): 87795

31. JSON

- JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, muy utilizado en el desarrollo web por su simplicidad, legibilidad y compatibilidad con múltiples lenguajes de programación
- Tiene estructura basada en texto que representa datos mediante pares clave – valor. Su sintaxis se deriva del formato de los objetos literales de JavaScript

```
{
  "nombre": "Ana",
  "edad": 25,
  "activo": true
}
```

31. JSON

- Formato de texto plano: fácil de leer y escribir tanto para humanos como para computadoras
- Estructurado: los datos se organizan en:
  - Objetos { } → colecciones de pares clave-valor
  - Arreglos [ ] → listas ordenadas de valores
- Tipos de datos permitidos:
  - Cadenas (string)
  - Números (number)
  - Booleanos (true, false)
  - Objetos y arreglos anidados
  - Nulos (null)
- Independiente de lenguaje: aunque nació de JavaScript, se usa en Python, Java, PHP, C#, Go, etc.
- Ideal para comunicación cliente-servidor: es el formato estándar para APIs REST y respuestas AJAX

31. JSON

- Conversión entre JSON y objetos JavaScript
- De JSON a objeto:

```
const objeto = JSON.parse('{"nombre": "Ana"}');
console.log(objeto.nombre); // Ana
```

- De objeto a JSON:

```
const json = JSON.stringify({ nombre: "Ana" });
console.log(json); // {"nombre":"Ana"}
```

31. JSON

```
const textoJSON = '{"nombre": "Ana", "edad": 25, "activo": true}';
const persona = JSON.parse(textoJSON);

console.log(persona.nombre); // Ana
console.log(persona.edad);   // 25
console.log(persona.activo); // true
```

```
const persona = {
  nombre: "Ana",
  edad: 25,
  activo: true
};

const json = JSON.stringify(persona);
console.log(json); // {"nombre":"Ana","edad":25,"activo":true}
```

31. JSON - Usos

- Intercambio de datos entre cliente y servidor
  - JSON es el formato estándar para enviar y recibir datos en aplicaciones web modernas
- Permite que diferentes sistemas, por ejemplo, un frontend en React (JavaScript) y un backend en Python se comuniquen sin importar el lenguaje
- Intercambio de información entre aplicaciones
  - JSON permite conectar sistemas distintos (por ejemplo, un sistema web con una app móvil o con un servicio en la nube). Gracias a su formato simple y universal, casi cualquier lenguaje puede generar y leer JSON
- En Bases de Datos NoSQL los registros se guardan en formato JSON o similar
- Comunicación en APIs REST y servicios web usan JSON como formato principal para las solicitudes (POST, PUT, PATCH) y las respuestas (GET)
- Almacenamiento de configuración
  - Muchos entornos y herramientas utilizan archivos .json para configurar proyectos
  - Ejemplo: package.json en Node.js
- Persistencia local en el navegador
  - JSON se usa para guardar información en localStorage o sessionStorage

32. Manejo de Excepciones

- Es una parte fundamental para crear código robusto, fácil de mantener y que no rompa la ejecución del programa ante fallos
- Un error o excepción ocurre cuando el motor de JavaScript encuentra algo inesperado o incorrecto durante la ejecución

Tipo de error	Cuándo ocurre	Ejemplo
SyntaxError	Cuando el código tiene un error de sintaxis	if (true {
ReferenceError	Se usa una variable no declarada	console.log(nombre);
TypeError	Cuando se usa un tipo incorrecto	"hola".push("mundo");
RangeError	Un número está fuera del rango permitido	new Array(-5)
URIError	Error en funciones de URI (como decodeURI)	decodeURI("%");

32. Manejo de Excepciones

```
try {
  // Código que puede causar un error
  let resultado = 10 / 0;
  console.log(resultado);
} catch (error) {
  // Código que se ejecuta si ocurre un error
  console.error("Ocurrió un error:", error.message);
}
```

```
try {
  console.log("Intentando algo...");
  throw new Error("Fallo simulado");
} catch (e) {
  console.log("Error capturado:", e.message);
} finally {
  console.log("Este bloque siempre se ejecuta.");
}
```

32. Manejo de Excepciones

```
function dividir(a, b) {
  if (b === 0) {
    throw new Error("No se puede dividir entre cero");
  }
  return a / b;
}

try {
  console.log(dividir(10, 0));
} catch (e) {
  console.error(e.message);
}
```

```
try {
  // Código que puede fallar
  JSON.parse("esto no es JSON");
} catch (error) {
  if (error instanceof SyntaxError) {
    console.error("Error de sintaxis en JSON:", error.message);
  } else if (error instanceof TypeError) {
    console.error("Error de tipo:", error.message);
  } else {
    console.error("Otro tipo de error:", error.message);
  }
}
```

33. Asincronía

- Es la capacidad de JavaScript para realizar tareas sin bloquear la ejecución del programa
- Puede continuar ejecutando otras instrucciones mientras espera que una operación de mayor duración termine (como una petición a un servidor o la lectura de un archivo)
- JavaScript no se detiene mientras algo tarda en completarse; en su lugar, delegará la tarea y seguirá ejecutando el resto del código

33. Asincronía

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

setTimeout(() => {
  console.log("Tarea asincrónica completada");
}, 2000);

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

```
Inicio
Fin
Tarea asincrónica completada
```

33. Asincronía - Ventajas

- JavaScript se ejecuta en un solo hilo (single-threaded)
- Si todo fuera sincrónico, cualquier tarea lenta (como acceder a una API o cargar una imagen grande) bloquearía el resto del programa
- La asincronía permite:
  - Mantener la aplicación responsiva
  - Aprovechar mejor los recursos del sistema
  - Esperar resultados externos sin detener la ejecución principal

33. Asincronía - Formas de manejar

- Tres formas principales de manejar la asincronía, que han evolucionado con el tiempo para hacer el código más claro y fácil de mantener
  - Callbacks
  - Promesas (Promises)
  - Async / Await

33. Asincronía - Formas de manejar

1) Callbacks

- Función que se pasa como argumento a otra función y que se ejecuta cuando la operación asincrónica termina
- Fue la primera forma usada para manejar la asincronía en JavaScript

```
// CALLBACK
function obtenerDatos(callback) {
  console.log("Cargando...");
  setTimeout(() => {
    callback("Datos cargados correctamente");
  }, 2000);
}

obtenerDatos(resultado => {
  console.log(resultado);
});
```

```
Cargando...
Datos cargados correctamente
```

33. Asincronía - Formas de manejar

2) Promesas (Promises)

- Representa un valor que aún no se conoce, pero que se resolverá (éxito o error) en el futuro
- Tiene tres estados:
  - pending (pendiente)
  - fulfilled (resuelta correctamente → `resolve`)
  - rejected (con error → `reject`)

```
const promesa = new Promise((resolve, reject) => {
  // código asincrónico
});
```

33. Asincronía - Formas de manejar

2) Promesas (Promises)

```
// PROMISES
function obtenerDatos() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const exito = true;
      if (exito) resolve("Datos obtenidos");
      else reject("Error al obtener datos");
    }, 2000);
  });
}

obtenerDatos()
  .then(resultado => console.log(resultado))
  .catch(error => console.error(error));
```

```
Datos obtenidos
```

```javascript
function verificarConexion() {
  return new Promise((resolve, reject) => {
    console.log("Verificando conexión...");
    setTimeout(() => {
      const conectado = true;
      if (conectado) resolve("Conectado al servidor");
      else reject("Sin conexión");
    }, 1000);
  });
}
function cargarDatos() {
  return new Promise((resolve) => {
    console.log("Cargando datos...");
    setTimeout(() => resolve(["Usuario1", "Usuario2", "Usuario3"]), 1500);
  });
}
```

```javascript
// Uso de las promesas encadenadas
verificarConexion()
  .then(msg => {
    console.log(msg);
    return cargarDatos();
  })
  .then(datos => {
    console.log("Datos cargados:", datos);
  })
  .catch(error => console.error(error))
  .finally(() => console.log("Proceso terminado"));
```

```
Verificando conexión...
Conectado al servidor
Cargando datos...
Datos cargados: ▸ (3) ['Usuario1', 'Usuario2', 'Usuario3']
Proceso terminado
```

33. Asincronía - Formas de manejar

3) Async / Await

- Permiten escribir código asincrónico con una sintaxis similar al código síncrono, más limpia y legible que las promesas
- Código moderno y limpio

```
async function nombreFuncion() {
  try {
    const resultado = await promesa;
    // usar resultado aquí
  } catch (error) {
    // manejar error aquí
  }
}
```

33. Asincronía - Formas de manejar

3) Async / Await

```
async function obtenerDatos() {
  try {
    console.log("Cargando...");
    const respuesta = await new Promise(resolve => {
      setTimeout(() => resolve("Datos listos"), 2000);
    });
    console.log(respuesta);
  } catch (error) {
    console.error(error);
  }
}

obtenerDatos();
```

```
Cargando...
Datos listos
```

33. Asincronía - Formas de manejar

3) Async / Await

```
async function obtenerDatos() {
  try {
    const respuesta = await new Promise((resolve, reject) => {
      setTimeout(() => {
        const exito = Math.random() > 0.5;
        if (exito) resolve("Datos listos");
        else reject("Error al obtener datos");
      }, 2000);
    });

    console.log(respuesta);
  } catch (error) {
    console.error(error);
  }
}

obtenerDatos();
```

```
Datos listos
```

```js
// CALLBACK
function tareaConCallback(estado, callback) {
  console.log("Procesando (callback)...");

  setTimeout(() => {
    if (estado === "ok") {
      callback(null, "Operación exitosa");
    } else if (estado === "error") {
      callback("Ocurrió un error");
    } else {
      console.log("La tarea quedó pendiente...");
      // No se llama al callback → pendiente
    }
  }, 1000);
}

// Éxito
tareaConCallback("ok", (error, resultado) => {
  if (error) console.error(error);
  else console.log(resultado);
});
```

```js
// PROMISE
function tareaConPromesa(estado) {
  console.log("Procesando (promesa)...");

  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (estado === "ok") {
        resolve("Operación exitosa");
      } else if (estado === "error") {
        reject("Ocurrió un error");
      } else {
        console.log("La promesa quedó pendie
      }
    }, 1000);
  });
}

// Éxito
tareaConPromesa("ok")
  .then(resultado => console.log(resultado))
  .catch(error => console.error(error));
```

```js
// ASYNC/AWAIT
async function tareaAsync(estado) {
  console.log("Procesando (async/await)...");

  try {
    const resultado = await new Promise((resolve, reject) => {
      setTimeout(() => {
        if (estado === "ok") {
          resolve("Operación exitosa");
        } else if (estado === "error") {
          reject("Ocurrió un error");
        } else {
          console.log("La promesa quedó pendiente...");
        }
      }, 1000);
    });

    console.log(resultado);
  } catch (error) {
    console.error(error);
  }
}

// Éxito
tareaAsync("ok");
```

34. AJAX

- AJAX (Asynchronous JavaScript And XML) es una técnica que permite a una página web comunicarse con un servidor sin recargar toda la página
- Gracias a él, las aplicaciones web pueden enviar o recibir datos en segundo plano y actualizar solo partes específicas de la interfaz, mejorando la experiencia del usuario (UX)
- Hoy en día, aunque el nombre incluye “XML”, en la práctica se usa JSON como formato principal de intercambio de datos

34. AJAX

- En las versiones modernas de JavaScript, la forma más práctica de implementar AJAX es mediante la API `fetch()`, que reemplaza al antiguo XMLHttpRequest
- Sintaxis:

```
fetch(url, opciones)
  .then(respuesta => respuesta.json())
  .then(datos => console.log(datos))
  .catch(error => console.error('Error:', error));
```

  - url: dirección del recurso o API
  - opciones (opcional): objeto que define método HTTP, cabeceras, cuerpo, etc.
  - Devuelve una Promise, lo que facilita el manejo asíncrono con .then() o async/await

34. AJAX

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())   // Convierte la respuesta a JSON
  .then(data => console.log(data))     // Muestra los datos en consola
  .catch(error => console.error('Error:', error)); // Captura errores
```

```
▸ {userId: 1, id: 1, title: 'delectus aut autem', completed: false}
```


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
  },
  {
    "userId": 1,
    "id": 2,
    "title": "quis ut nam facilis et officia qui",
    "completed": false
  },
  {
    "userId": 1,
    "id": 3,
    "title": "fugiat veniam minus",
    "completed": false
  },
  {
    "userId": 1,
    "id": 4,
    "title": "et porro tempora",
    "completed": true
  },
  {
    "userId": 1,
    "id": 5,
    "title": "laboriosam mollitia et enim quasi adipisci quia provident illum",
    "completed": false
  },
```

34. AJAX

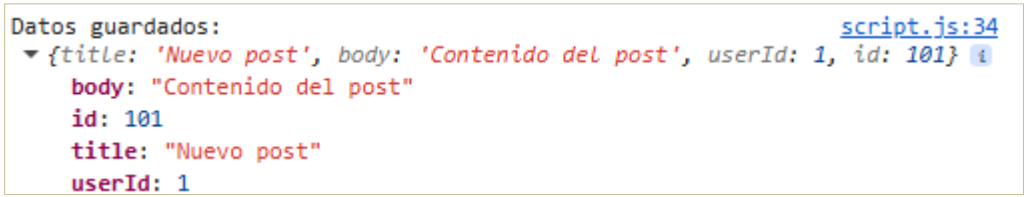
```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => {
    if (!response.ok) throw new Error('Error en la respuesta');
    return response.json(); // convierte JSON a objeto JS
  })
  .then(users => {
    console.log('Usuarios:', users);
  })
  .catch(error => console.error('Hubo un problema:', error));
```

```
Usuarios:                                                    script.js:17
▼ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▶ 0: {id: 1, name: 'Leanne Graham', username: 'Bret', email: 'Sincere@apr
  ▶ 1: {id: 2, name: 'Ervin Howell', username: 'Antonette', email: 'Shanna@
  ▶ 2: {id: 3, name: 'Clementine Bauch', username: 'Samantha', email: 'Nath
  ▶ 3: {id: 4, name: 'Patricia Lebsack', username: 'Karianne', email: 'Juli
  ▶ 4: {id: 5, name: 'Chelsey Dietrich', username: 'Kamren', email: 'Lucio_
  ▶ 5: {id: 6, name: 'Mrs. Dennis Schulist', username: 'Leopoldo_Corkery',
  ▶ 6: {id: 7, name: 'Kurtis Weissnat', username: 'Elwyn.Skiles', email: 'T
  ▶ 7: {id: 8, name: 'Nicholas Runolfsdottir V', username: 'Maxime_Nienow',
  ▶ 8: {id: 9, name: 'Glenna Reichert', username: 'Delphine', email: 'Chaim
  ▶ 9: {id: 10, name: 'Clementina DuBuque', username: 'Moriah.Stanton', ema
    length: 10
```


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
    },
    "phone": "010-692-6593 x09125",
    "website": "anastasia.net",
    "company": {
      "name": "Deckow-Crist",
      "catchPhrase": "Proactive didactic contingency",
      "bs": "synergize scalable supply-chains"
    }
  },
```

34. AJAX

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    title: 'Nuevo post',
    body: 'Contenido del post',
    userId: 1
  })
})
  .then(response => response.json())
  .then(data => console.log('Datos guardados:', data))
  .catch(error => console.error('Error al enviar:', error));
```



34. AJAX

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo AJAX</title>
</head>
<body>
    <h1>Ejemplo básico de AJAX con fetch()</h1>
    <button id="boton">Cargar datos</button>
    <p id="resultado"></p>
    <script src="script.js"></script>
</body>
</html>
```


```javascript
document.getElementById("boton").addEventListener("click", () => {
  // Petición AJAX con fetch
  fetch("https://jsonplaceholder.typicode.com/users/1")
    .then(respuesta => respuesta.json())   // Convertir a JSON
    .then(data => {
      // Mostrar el título recibido en la página
      document.getElementById("resultado").textContent = data.name;
    })
    .catch(error => console.error("Error:", error));
});
```

```
// Sintaxis con async/await

async function obtenerDatos() {
  try {
    const respuesta = await fetch(url, opciones);
    const data = await respuesta.json();
    console.log(data);
  } catch (error) {
    console.error("Error:", error);
  }
}
```

```
Usuarios:                                                    script.js:43
▼ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▶ 0: {id: 1, name: 'Leanne Graham', username: 'Bret', email: 'Sincere@apr
  ▶ 1: {id: 2, name: 'Ervin Howell', username: 'Antonette', email: 'Shanna@
  ▶ 2: {id: 3, name: 'Clementine Bauch', username: 'Samantha', email: 'Nath
  ▶ 3: {id: 4, name: 'Patricia Lebsack', username: 'Karianne', email: 'Juli
  ▶ 4: {id: 5, name: 'Chelsey Dietrich', username: 'Kamren', email: 'Lucio_
  ▶ 5: {id: 6, name: 'Mrs. Dennis Schulist', username: 'Leopoldo_Corkery',
  ▶ 6: {id: 7, name: 'Kurtis Weissnat', username: 'Elwyn.Skiles', email: 'T
  ▶ 7: {id: 8, name: 'Nicholas Runolfsdottir V', username: 'Maxime_Nienow',
  ▶ 8: {id: 9, name: 'Glenna Reichert', username: 'Delphine', email: 'Chaim
  ▶ 9: {id: 10, name: 'Clementina DuBuque', username: 'Moriah.Stanton', ema
    length: 10
```

```
async function cargarUsuarios() {
  try {
    const respuesta = await fetch('https://jsonplaceholder.typicode.com/users');
    if (!respuesta.ok) throw new Error('Error al obtener datos');
    const usuarios = await respuesta.json();
    console.log('Usuarios:', usuarios);
  } catch (error) {
    console.error('Error:', error);
  }
}

cargarUsuarios();
```

34. AJAX

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo AJAX</title>
</head>
<body>
    <h1>Ejemplo básico de AJAX con fetch()</h1>
    <button id="boton">Cargar datos</button>
    <p id="resultado"></p>
    <script src="script.js"></script>
</body>
</html>
```


```javascript
// Definimos una función asíncrona
async function cargarDatos() {
  try {
    // Espera la respuesta del servidor
    const respuesta = await fetch("https://jsonplaceholder.typicode.com/users/1");

    // Convierte la respuesta a JSON
    const data = await respuesta.json();

    // Muestra el resultado en el HTML
    document.getElementById("resultado").textContent = data.name;
  } catch (error) {
    console.error("Error:", error);
  }
}

// Ejecutar la función cuando se haga clic en el botón
document.getElementById("boton").addEventListener("click", cargarDatos);
```

```html
<h1>Buscar usuario (JSONPlaceholder)</h1>

<form id="formulario">
    <input type="number" id="userId" placeholder="ID (1-10
    <button type="submit">Buscar</button>
</form>

<p id="resultado"></p>

<script src="script.js"> </script>
```

```css
body {
    font-family: Arial, sans-serif;
    padding: 20px;
    background-color: #fafafa;
}

h2 {
    color: #333;
}

form {
    margin-bottom: 15px;
}

input {
    padding: 8px;
    margin-right: 8px;
    width: 120px;
}

button {
    padding: 8px 12px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

#resultado {
    margin-top: 20px;
    background-color: #adcc76;
    padding: 15px;
    border-radius: 5px;
    min-height: 40px;
}
```

Buscar usuario (JSONPlaceholder)

2 Buscar

Usuario encontrado:
Nombre: Ervin Howell
Usuario: Antonette
Email: Shanna@melissa.tv
Ciudad: Wisokyburgh

```javascript
const formulario = document.getElementById("formulario");
const idInput = document.getElementById("userId");
const resultado = document.getElementById("resultado");

formulario.addEventListener("submit", async (e) => {
    e.preventDefault();
    const id = idInput.value;

    resultado.textContent = "Buscando usuario...";

    try {
        const respuesta = await fetch(`https://jsonplaceholder.typicode.com/users/${id}`);

        if (!respuesta.ok) {
            throw new Error("Usuario no encontrado");
        }

        const usuario = await respuesta.json();

        // Mostrar la información del usuario
        resultado.innerHTML = `
            <strong>Usuario encontrado:</strong><br>
            <strong>Nombre:</strong> ${usuario.name}<br>
            <strong>Usuario:</strong> ${usuario.username}<br>
            <strong>Email:</strong> ${usuario.email}<br>
            <strong>Ciudad:</strong> ${usuario.address.city}
            `;
    } catch (error) {
        resultado.textContent = "No se pudo obtener el usuario. Verifica el ID.";
        console.error(error);
    }
});
```

34. AJAX - Usos

- Actualizar secciones de una página sin recargarla (por ejemplo, comentarios, notificaciones)
- Enviar formularios dinámicamente con validaciones en tiempo real
- Consumir APIs REST de servicios externos
- Cargar datos al iniciar la página (por ejemplo, listas, usuarios, productos)
- Interacción con servidores backend en proyectos con Node.js, Python, Java, PHP, etc.
- AJAX con fetch() es la forma moderna, limpia y estandarizada de realizar peticiones HTTP asíncronas en JavaScript
- Favorece la arquitectura SPA (Single Page Application) y el uso de APIs REST

35. Módulos

- Son una forma de organizar el código en archivos separados y reutilizables, de manera que cada módulo pueda encapsular sus variables, funciones, clases o constantes
- Evita conflictos de nombres y facilita el mantenimiento del proyecto
- Un módulo es simplemente un archivo de JavaScript que exporta partes de su código (funciones, clases, objetos, variables) y puede importar código desde otros módulos
- Esto permite dividir un programa grande en partes más pequeñas y comprensibles

35. Módulos - Ventajas

- Reutilización de código: se puede usar el mismo módulo en distintos proyectos
- Mantenimiento más fácil: el código está dividido en archivos especializados
- Encapsulamiento: evita conflictos de nombres (cada módulo tiene su propio alcance)
- Carga selectiva: sólo se importa lo necesario
- Compatibilidad con herramientas modernas como Node.js, React, Angular, Vue.js, etc.

35. Módulos

- ES Modules (ESM): es el estándar moderno
- Se usan en el navegador y en Node.js (desde versiones recientes)
- Extensión habitual: .js o .mjs
- Se usan las palabras clave export y import

saludo.js > ...

```
// Exportar una función
export function saludar(nombre) {
    return `Hola, ${nombre}!`;
}
```

```
<!-- Importante: declarar el tipo de módulo -->
<script type="module" src="./main.js"></script>
```

main.js > ...

```
// Importar la función desde otro archivo
import { saludar } from "./saludo.js";

console.log(saludar("Juancito"));
```

```html
    <h1>Ejemplo de Módulos ES6 en JavaScript</h1>
    <button id="boton">Usar importación dinámica</button>

    <!-- Importante: declarar el tipo de módulo -->
    <script type="module" src="./main.js"></script>
</body>
</html>
```

saludo.js > ...

```javascript
// Exportación nombrada: otra función opcional
export function despedir(nombre) {
    return `Adiós, ${nombre}. ¡Hasta pronto!`;
}

// Exportación por defecto: una función principal
export default function saludar(nombre) {
    return `¡Hola, ${nombre}! Bienvenido al sistema.`;
}
```

operaciones.js > ...

```javascript
// Exportaciones nombradas
export const PI = 3.1416;

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

export function multiplicar(a, b) {
    return a * b;
}

// También puedes exportar una clase
export class Calculadora {
    restar(a, b) {
        return a - b;
    }
    dividir(a, b) {
        return b !== 0 ? a / b : 'Error: división por cero';
    }
}
```

main.js > ...

```javascript
// Importar la exportación por defecto y una nombrada desde saludo.js
import saludar, { despedir } from "./saludo.js";

// Importar varias exportaciones nombradas desde operaciones.js
import { PI, sumar, multiplicar, Calculadora } from "./operaciones.js";

// Mostrar en la consola
console.log(saludar("Juancito"));
console.log(despedir("Juancito"));

console.log("Suma:", sumar(5, 3));
console.log("Multiplicación:", multiplicar(4, 2));
console.log("Valor de PI:", PI);

const calc = new Calculadora();
console.log("Resta:", calc.restar(10, 4));
console.log("División:", calc.dividir(10, 2));

// Ejemplo con importación dinámica
document.querySelector("#boton").addEventListener("click", async () => {
    const mod = await import("./operaciones.js");
    alert(`Importación dinámica: 2 + 3 = ${mod.sumar(2, 3)}`);
});
```

36. Expresiones Regulares

- Son patrones utilizados para buscar, validar o reemplazar texto dentro de cadenas
- Son una herramienta muy poderosa para trabajar con texto de forma precisa y flexible
- Define un patrón de búsqueda
- Se usa en procesamiento de texto, validaciones de formularios, análisis de datos y más

Notación Literal:

```
/patrón/flags


const patron = /abc/;
const regex = /[A-Z][a-z]+/g;
```

Constructor:

```
new RegExp("patrón", "flags")


const patron = new RegExp("abc");
const regex = new RegExp("[A-Z][a-z]+", "g");
```

36. Expresiones Regulares

```
const regex = /[A-Z][a-z]+/g;
```

Parte	Significado	Explicación detallada
/ ... /	Delimitadores	Indican el inicio y fin del patrón literal de expresión regular
[A-Z]	Una letra mayúscula	Coincide con cualquier carácter entre la A y la Z. Es la primera letra obligatoriamente
[a-z]+	Una o más letras minúsculas	El + significa “una o más repeticiones”. Aquí se permiten varias letras minúsculas después de la inicial
g	Bandera global (global flag)	Permite buscar todas las coincidencias en un texto (no sólo la primera)

"Hola" □ "Perro“ □ "hola" □ "HOLA" □
"ElGato" □ "123Hola" □ "H" □ "" □

36. Expresiones Regulares – Elementos Básicos

Símbolo	Significado	Ejemplo	Coincide con
.	Cualquier carácter (excepto **salto de línea**)	/a.b/	“acb”, “a7b”
\d	Dígito (0–9)	/\d+/	“123”
\w	Carácter alfanumérico o _	/\w+/	“user_1”
\s	Espacio, tabulación o salto de línea	/\s/	“ ”
^	Inicio de cadena o línea	/^Hola/	“Hola mundo”
$	Fin de cadena o línea	/mundo$/	“Hola mundo”
[...]	Uno de los caracteres listados	/[aeiou]/	“a”, “e”, “i”, “o”, “u”
[^...]	Ninguno de los caracteres listados	/[^aeiou]/	“b”, “c”, “d”
( )	Agrupación o captura	/(ab)+/	“abab”
?	0 o 1 ocurrencia	/colou?r/	“color” o “colour”
*	0 o más ocurrencias	/go*/	“g”, “go”, “goo”
+	1 o más ocurrencias	/go+/	“go”, “goo”
{n}	Exactamente n repeticiones	/\d{3}/	“123”
{n,}	n o más repeticiones	/\d{2,}/	“12”, “123”, “1234”
{n,m}	Entre n y m repeticiones	/\d{2,4}/	“12”, “123”, “1234”

36. Expresiones Regulares - Flags

Bandera	Significado	Ejemplo
g	Global: busca todas las coincidencias (no sólo la primera)	/a/g
i	Ignore case: ignora mayúsculas/minúsculas	/a/i
m	Multilínea: ^ y $ actúen por línea, no sólo al inicio o fin de todo el texto	/^Hola/m
s	DotAll: permite que . coincida también con **saltos de línea** (\n, \r, etc.)	/a.b/s
u	Unicode: permite usar caracteres Unicode (emojis, acentos, símbolos, caracteres no latinos, etc.)	/^.$/u

36. Expresiones Regulares

```
const regex = /^[A-Z][a-z]+$/g;
```

Parte	Significado	Explicación detallada
/ ... /	Delimitadores	Indican el inicio y fin del patrón literal de expresión regular
^	Inicio de la cadena	Indica que la coincidencia debe comenzar desde el principio del texto
[A-Z]	Una letra mayúscula	Coincide con cualquier carácter entre la A y la Z. Es la primera letra obligatoriamente
[a-z]+	Una o más letras minúsculas	El + significa “una o más repeticiones”. Aquí se permiten varias letras minúsculas después de la inicial
$	Fin de la cadena	Indica que la coincidencia debe terminar justo al final del texto
g	Bandera global (global flag)	Permite buscar todas las coincidencias en un texto (no solo la primera)

36. Expresiones Regulares

```
const regex = /^[A-Z][a-z]+$/g;

console.log(regex.test("Juancito"));  // true  → Empieza con mayúscula
                                              y sigue con minúsculas
console.log(regex.test("juancito"));  // false → Empieza con minúscula
console.log(regex.test("JUANCITO"));  // false → Todo en mayúsculas
console.log(regex.test("JuancitoPerez")); //false → Hay otra mayúscula
console.log(regex.test("hola JuancitoPerez")); //false → ver ^ $

const regex = /(ab)+/;
console.log(regex.test("abab")); // true
```

36. Expresiones Regulares - Métodos

Método	Uso principal	Ejemplo
test()	Verifica si el patrón existe en una cadena (devuelve true o false)	/hola/.test("hola mundo") // true
exec()	Devuelve información detallada del primer match (como un array) o null si no hay coincidencia	/\d+/.exec("Edad: 25 años") // ["25"]
match()	(De String) Devuelve todas las coincidencias si se usa con la bandera g	"abc123".match(/\d/g) // ["1","2","3"]
replace()	Reemplaza el texto que coincide con el patrón	"Hola mundo".replace(/mundo/, "JS") // "Hola JS"
split()	Divide una cadena en partes según el patrón	"uno, dos, tres".split(/,\s*/) // ["uno", "dos", "tres"]
search()	Devuelve el índice donde empieza la primera coincidencia, ó -1	"abc123".search(/\d/) // 3

Buscar coincidencias

```
const texto = "El número es 12345";
const patron = /\d+/;
const resultado = texto.match(patron);
console.log(resultado[0]); // "12345"
```

Extraer números de un texto

```
const texto = "Tengo 2 perros y 3 gatos.";
const numeros = texto.match(/\d+/g); //["2", "3"]
```

Validar un correo electrónico

```
const regex = /^[\w.-]+@[a-zA-Z\d.-]+\.[a-zA-Z]{2,}$/;
console.log(regex.test("usuario@mail.com")); // true
```

Reemplazar texto

```
const texto = "Mi número es 987654321";
const resultado = texto.replace(/\d/g, "*");
console.log(resultado); // "Mi número es *********"
```

Reemplazar múltiples espacios por uno solo

```
const texto = "Hola   mundo   JS";
console.log(texto.replace(/\s+/g, " ")); // "Hola mundo JS"
```

Verificar formato de fecha (dd/mm/yyyy)

```
const fecha = /^(\d{2})\/(\d{2})\/(\d{4})$/;
console.log(fecha.test("08/11/2025")); // true
```

Extraer información

```
const fecha = "2025-11-08";
const patron = /(\d{4})-(\d{2})-(\d{2})/;
const [, año, mes, dia] = fecha.match(patron); // desestructura ["2025-11-08", "2025", "11", "08"]
console.log(`Año: ${año}, Mes: ${mes}, Día: ${dia}`);
```

Ejercicios

1. Crea un botón que, al hacer clic, cambie el fondo de la página a un color aleatorio

   Usa addEventListener() para manejar el evento

2. Crea un formulario con un campo de texto y un botón “Agregar tarea”

   Cada vez que el usuario escriba algo y haga clic, agrega un nuevo elemento <li> dentro de una lista <ul> con el texto ingresado

   Agrega además un botón junto a cada tarea que permita eliminarla

3. Implementa la funcionalidad anterior, pero usando delegación de eventos:

   En lugar de asignar un addEventListener a cada botón □ , coloca un único listener en la lista <ul> que detecte qué elemento fue clicado

4. Crea una tabla con datos simples (por ejemplo, nombres y edades)

   Con un botón, alterna entre ordenar las filas ascendentemente o descendentemente por edad, sin recargar la página

5. Crea tres cajas con la clase "caja". Al hacer clic en una, debe:

   Cambiar su color, agregar la clase "activa“ y quitar "activa" del resto de las cajas

6. Usa fetch() para obtener datos de la API pública: https://jsonplaceholder.typicode.com/users

   Muestra los nombres de los usuarios en la consola y en un HTML

Ejercicios

7. Con los mismos datos de usuarios del ejercicio anterior, muestra sus nombres y correos en una lista ordenada dentro de una página HTML
8. Envía un nuevo “post” a la API:
   - https://jsonplaceholder.typicode.com/posts
   - El cuerpo (body) debe incluir un objeto con title, body y userId
   - Muestra la respuesta del servidor en consola
9. Crea una función asincrónica que obtenga datos de una API
   - Si la respuesta no es correcta (!response.ok), lanza un error y muéstralo en consola con un try...catch
   - Prueba usando una URL correcta y otra inexistente
10. Crea un formulario con campos nombre y email
    - Cuando el usuario lo envíe, usa fetch() para enviar los datos a una API (puede ser https://jsonplaceholder.typicode.com/posts)
    - Muestra en pantalla un mensaje con el resultado de la operación

Ejercicios

11. Crea un archivo operaciones.js que exporte tres funciones: sumar(a, b), restar(a, b) y multiplicar(a, b)

    En otro archivo app.js, importa todas las funciones y muestra sus resultados en la consola

    Luego, importa solo sumar

12. Crea una mini aplicación con los siguientes módulos:
    - datos.js: exporta un arreglo de objetos con usuarios ({ nombre, edad })
    - filtros.js: exporta una función filtrarMayores(usuarios) que retorne solo los mayores de edad
    - main.js: importa ambos módulos y muestra en consola los usuarios mayores de 18 años

13. Generar la expresiones regulares para:

    a) Sólo letras b) Sólo números c) DNI peruano (8 dígitos) d) Celular peruano

    e) Contraseña segura: al menos 8 caracteres, al menos una letra mayúscula (A–Z), al menos una letra minúscula (a–z), al menos un número (0–9) y al menos un símbolo especial (@#$%^&*_-)