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

24. Estructuras de Datos - Map

- Estructura de datos incorporada (ES6, 2015) que permite almacenar pares **clave–valor**
- Las claves pueden ser de cualquier tipo: primitivas, funciones, objetos, etc.
- Mantiene el orden de inserción de los elementos
- Tiene métodos y propiedades específicos para manipular los pares clave–valor

```
const nombre = new Map();
```

```
const nombre = new Map([
    [clave1, valor1],
    [clave2, valor2],
    ...
])
```

24. Estructuras de Datos - Map

```
const miMapa = new Map();

console.log(miMapa); //usa {}
```

```
const miMapa = new Map([
  ["nombre", "Juancito"],
  ["edad", 30],
  ["activo", true]
]);

console.log(miMapa);
```

```
maps.js:25
▸ Map(3) {'nombre' => 'Juancito', 'edad' => 30, 'activo' =
  > true}
```

Método / Propiedad	Descripción	Ejemplo

set(clave, valor)	Agrega un nuevo par clave–valor o actualiza el valor si la clave ya existe	const miMapa = new Map();
		miMapa.set("nombre", "Juan");
		miMapa.set("edad", 30);

get(clave)	Obtiene el valor asociado a una clave	miMapa.get("nombre");
has(clave)	Verifica si existe una clave en el Map	miMapa.has("edad");
delete(clave)	Elimina un elemento por su clave	miMapa.delete("edad");
clear()	Elimina todos los elementos del Map	miMapa.clear();
size	Devuelve la cantidad de pares clave–valor en el Map	const mapa = new Map([["a", 1], ["b", 2]]);

24. Estructuras de Datos - Map

```
const usuarios = new Map([
  [1, "Carla"],
  [2, "Valeria"],
  [3, "Jennifer"]
]);
// for...of con desestructuración
for (const [id, nombre] of usuarios) {
  console.log(id, nombre);
}

// forEach
usuarios.forEach((nombre, id) => {
  console.log(id, nombre);
});
```

```js
let capitales = new Map([
  ["Perú", "Lima"],
  ["Chile", "Santiago"],
  ["Argentina", "Buenos Aires"]
]);

console.log(capitales);

for (let [pais, capital] of capitales) {
  console.log(pais, "→", capital);
}

for (let pais of capitales.keys()) {
  console.log(pais);
}

for (let capital of capitales.values()) {
  console.log(capital);
}

capitales.forEach((valor, clave) => {
  console.log(clave + " → " + valor);
});
```

```
▸ Map(3) {'Perú' => 'Lima', 'Chile' => 'Santiago', 'Argentin
  ires'}
Perú → Lima
Chile → Santiago
Argentina → Buenos Aires
Perú
Chile
Argentina
Lima
Santiago
Buenos Aires
Perú → Lima
Chile → Santiago
Argentina → Buenos Aires                    script.js:365
```

25. Objetos

- Es una colección de pares **clave–valor**, donde cada clave (o propiedad) se asocia a un valor
- La clave siempre es un string
- El valor puede ser cualquier tipo de dato: primitivo (número, string, boolean, null, undefined), otro objeto, un array o incluso una función

```
const nombreObjeto = {
  clave1: valor1,
  clave2: valor2,
  clave3: valor3
};
```

```
const persona = {
  nombre: "Juancito",
  edad: 30,
  casado: false
};
```

25. Objetos

```
const persona = {
  nombre: "Juancito",
  edad: 30,
  casado: false
};

console.log(persona); //{nombre: 'Juancito', edad: 30, casado: false}
console.log(persona.nombre); // "Juancito"
console.log(persona["edad"]); // 30

persona.edad = 31;                // Modificar
persona.profesion = "Ingeniero"; // Agregar nueva

//{nombre: 'Juancito', edad: 31, casado: false, profesion: 'Ingeniero'}
console.log(persona);
```

```js
const elAuto = {
  marca: "Toyota",
  modelo: "Corolla",
  arrancar() {
    console.log("El auto está arrancando...");
  }
};

//{marca: 'Toyota', modelo: 'Corolla', arrancar: ƒ}
console.log(elAuto);

elAuto.arrancar();
```

```js
const tuUsuario = {};
tuUsuario.nombre = "Ana";
tuUsuario.edad = 25;

//{nombre: 'Ana', edad: 25}
console.log(tuUsuario);

//{nombre: 'Ana'}
delete tuUsuario.edad; // elimina propiedad

console.log(tuUsuario);
```

```js
const persona = {
  nombre: "Juancito",
  edad: 30,
  casado: false
};
```

```js
//para recorrer
for (let clave in persona) {
  console.log(clave, persona[clave]);
}

// Obtener las claves (propiedades) del objeto
const claves = Object.keys(persona);
console.log(claves);

//para recorrer
Object.keys(persona).forEach(clave => {
  console.log(clave, ":", persona[clave]);
});

// Obtener los valores del objeto
const valores = Object.values(persona);
console.log(valores);

//para recorrer
Object.values(persona).forEach(valor => {
  console.log(valor);
});

// Obtener pares clave-valor
const entradas = Object.entries(persona);
console.log(entradas);

//para recorrer
Object.entries(persona).forEach(([clave, valor]) => {
  console.log(clave, ":", valor);
});
```

```
// Objeto principal que representa una universidad
const universidad = {
  nombre: "Universidad Nacional de San Agustín",
  ubicacion: {
    ciudad: "Arequipa",
    pais: "Perú"
  },

  // Set para almacenar las carreras sin duplicados
  carreras: new Set(["Ingeniería de Sistemas", "Industrial", "Administración"]),

  // Mapa que relaciona cada alumno con sus datos
  alumnos: new Map(),

  // Método para agregar un alumno
  agregarAlumno(nombre, edad, carrera) {
    // Crear el objeto alumno
    const alumno = {
      nombre,
      edad,
      carrera,
      cursos: [],
      agregarCurso(curso) {
        this.cursos.push(curso);
      },
```

```
      obtenerResumen() {
        return `${this.nombre}, ${this.edad} años – ${this.carrera}`;
      }
    };

    // Registrar en el Map usando el nombre como clave
    this.alumnos.set(nombre, alumno);
  },

  // Método para mostrar información general
  mostrarResumen() {
    console.log(`${this.nombre} (${this.ubicacion.ciudad}, ${this.ubicacion.pais})`);
    console.log("Carreras disponibles:", [...this.carreras].join(", "));
    console.log("Total de alumnos registrados:", this.alumnos.size);
  },

  // Método para mostrar los alumnos registrados
  listarAlumnos() {
    for (let [nombre, alumno] of this.alumnos.entries()) {
      console.log("–", alumno.obtenerResumen());
    }
  }
};
```

```
// --- Uso del objeto complejo ---

universidad.agregarAlumno("Marco", 19, "Ingeniería de Sistemas");
universidad.agregarAlumno("Carla", 20, "Industrial");

// Acceso a un alumno y modificación
const carla = universidad.alumnos.get("Carla");
carla.agregarCurso("Desarrollo Web");
carla.agregarCurso("Bases de Datos");

universidad.mostrarResumen();
universidad.listarAlumnos();

console.log("Cursos de Carla:", carla.cursos);
```

25. Objetos: Clonación y Desestructuración

- **Clonar** significa crear una copia de un objeto o arreglo existente
- JavaScript distingue entre copia superficial (shallow copy) y copia profunda (deep copy)

```
const persona = {
  nombre: "Juancito",
  direccion: { ciudad: "Arequipa" }
};

// Copia superficial con el spread operator
const copia = { ...persona };

copia.nombre = "Luis"; // solo cambia en copia, es primitivo
copia.direccion.ciudad = "Lima"; // ¡también cambia en persona!

console.log(persona.direccion.ciudad); // "Lima"
console.log(persona.nombre); // "Juancito"
console.log(copia.direccion.ciudad); // "Lima"
console.log(copia.nombre); // "Luis"
```

```
const persona = {
  nombre: "Juancito",
  direccion: { ciudad: "Arequipa" }
};

// Copia profunda
const copia = structuredClone(persona);

copia.nombre = "Luis";
copia.direccion.ciudad = "Lima";

console.log(persona.direccion.ciudad); // "Arequipa"
console.log(persona.nombre); // "Juancito"
console.log(copia.direccion.ciudad); // "Lima"
console.log(copia.nombre); // "Luis"
```

25. Objetos: Clonación y Desestructuración

- **Desestructuración** permite extraer valores de arrays u objetos de manera más clara y compacta
- Es una forma rápida de asignar múltiples variables a partir de una estructura de datos

```
// Básico
const usuario = { nombre: "Ana", edad: 25, ciudad: "Lima" };

const { nombre, edad } = usuario;

console.log(nombre); // "Ana"
console.log(edad);   // 25
```

```
// Con alias y valores por defecto
const { nombre: n, pais = "Perú" } = usuario;

console.log(n);    // "Ana"
console.log(pais); // "Perú"
```

25. Objetos: Clonación y Desestructuración

```
// Anidada
const persona = {
  nombre: "Carlos",
  direccion: {
    ciudad: "Arequipa",
    pais: "Perú"
  }
};

const { direccion: { ciudad } } = persona;
console.log(ciudad); // "Arequipa"
```

```
// Con arrays
const numeros = [10, 20, 30];

const [a, b] = numeros;
console.log(a, b); // 10 20

// Salteando
const [primero, , tercero] = numeros;
console.log(tercero); // 30
```

26. Clases

- Sintaxis especial introducida con ECMAScript 2015 (ES6) que permite definir objetos y manejar la programación orientada a objetos (POO) de una forma más clara

```
class NombreClase {
  // Constructor: inicializa la instancia
  constructor(param1, param2) {
    this.propiedad1 = param1;
    this.propiedad2 = param2;
  }

  // Métodos de instancia: disponibles para cada objeto
  metodo1() {
    console.log(`Propiedad1: ${this.propiedad1}`);
  }

  metodo2() {
    console.log(`Propiedad2: ${this.propiedad2}`);
  }

  // Método estático: pertenece a la clase, no a la instancia
  static metodoEstatico() {
    console.log("Soy un método de la clase");
  }
}
```

```
const obj1 = new NombreClase("Valor1", "Valor2");
obj1.metodo1(); // Propiedad1: Valor1
obj1.metodo2(); // Propiedad2: Valor2

// Llamar a método estático
NombreClase.metodoEstatico(); // Soy un método de la clase
```

```
class Persona {
  // Constructor: inicializa atributos
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  // Método de instancia
  saludar() {
    console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años.`);
  }
  toString(){
    return this.nombre+", "+this.edad;
  }
}

const miPersona = new Persona("Juan", 23);
miPersona.saludar();
console.log(miPersona.toString());
```

26. Clases

Propiedades privadas:

```
class Cuenta {
  #saldo = 0; // privada

  depositar(monto) {
    this.#saldo += monto;
  }

  verSaldo() {
    return this.#saldo;
  }
}

const cuenta = new Cuenta();
cuenta.depositar(100);
console.log(cuenta.verSaldo()); // 100
// cuenta.#saldo -> Error: privada
```

Setters y Getters:

```
class Persona {
  // Propiedades privadas (usando #)
  #nombre;
  #edad;

  constructor(nombre, edad) {
    this.#nombre = nombre;
    this.#edad = edad;
  }

  // Setter y Getter para nombre
  set nombre(nuevoNombre) {
    if (nuevoNombre.trim() === "") {
      console.log("El nombre no puede estar vacío");
      return;
    }
    this.#nombre = nuevoNombre;
  }
  get nombre() {
    return this.#nombre;
  }
  // Setter y Getter para edad
  get edad() {
    return this.#edad;
  }
  set edad(valor) {
    if (valor < 0 || valor > 120) {
      console.log("Edad no válida");
      return;
    }
    this.#edad = valor;
  // Método público
  saludar() {
    console.log(`Hola, soy ${this.#nombre} y tengo ${this.#edad} años.`);
  }
}

// Uso
let p1 = new Persona("Ana", 25);
p1.saludar(); // Hola, soy Ana y tengo 25 años.
// Acceder a través de getters y setters
console.log(p1.nombre); // Ana
p1.nombre = "Carlos";   // Cambia el nombre
p1.saludar();           // Hola, soy Carlos y tengo 25 años.
// Intentar acceder directamente (❌ error)
console.log(p1.#nombre); // SyntaxError: campo privado
```

toString():

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  // Sobrescribimos el método toString
  toString() {
    return `Persona: ${this.nombre}, ${this.edad} años`;
  }
}

const p = new Persona("Ana", 25);

console.log(p);          // Muestra el objeto completo
console.log(String(p)); // Llama a toString() automáticamente
console.log("Los datos son "+ p);
console.log(`${p}`);     // También llama a toString()
```

```
▸ Persona {nombre: 'Ana', edad: 25}
Persona: Ana, 25 años
Los datos son Persona: Ana, 25 años
Persona: Ana, 25 años
```

26. Clases

Herencia:

```
class NombreClase {
  // Constructor: inicializa la instancia
  constructor(param1, param2) {
    this.propiedad1 = param1;
    this.propiedad2 = param2;
  }

  // Métodos de instancia: disponibles para cada objeto
  metodo1() {
    console.log(`Propiedad1: ${this.propiedad1}`);
  }

  metodo2() {
    console.log(`Propiedad2: ${this.propiedad2}`);
  }

  // Método estático: pertenece a la clase, no a la instancia
  static metodoEstatico() {
    console.log("Soy un método de la clase");
  }
}
```

```
class SubClase extends NombreClase {
  constructor(param1, param2, param3) {
    super(param1, param2); // Llama al constructor de la clase padre
    this.propiedad3 = param3;
  }

  metodo3() {
    console.log(`Propiedad3: ${this.propiedad3}`);
  }
}

const obj2 = new SubClase("A", "B", "C");
obj2.metodo1(); // Heredado de NombreClase
obj2.metodo3(); // Propio de SubClase
```

26. Clases

Prototipos:

```
// CLASES
class Persona {
  constructor(nombre) {
    this.nombre = nombre;
  }

  saludar() {
    return `Hola, soy ${this.nombre}`;
  }
}

let p1 = new Persona("Ana");
console.log(p1.saludar()); // "Hola, soy Ana"
```

```
// Prototipos, antiguo
// Función constructora
function Persona(nombre) {
  this.nombre = nombre;
}

// Método asociado al prototipo
Persona.prototype.saludar = function() {
  return `Hola, soy ${this.nombre}`;
};

let p1 = new Persona("Ana");
console.log(p1.saludar()); // "Hola, soy Ana"
```

```javascript
// CLASES
class Persona {
  constructor(nombre) {
    this.nombre = nombre;
  }

  saludar() {
    return `Hola, soy ${this.nombre}`;
  }
}

let p1 = new Persona("Ana");
console.log(p1.saludar()); // "Hola, soy Ana"
```

```javascript
// Prototipos, antiguo
// Función constructora
function Persona(nombre) {
  this.nombre = nombre;
}

// Método asociado al prototipo
Persona.prototype.saludar = function() {
  return `Hola, soy ${this.nombre}`;
};

let p1 = new Persona("Ana");
console.log(p1.saludar()); // "Hola, soy Ana"
```

```javascript
// Objeto prototipo base
let persona = {
  saludar() {
    console.log(`Hola, soy ${this.nombre}`);
  }
};

// Crear un nuevo objeto que herede de persona
let p1 = Object.create(persona);
p1.nombre = "Ana";
p1.saludar(); // Hola, soy Ana
```

26. Clases

Polimorfismo:

```
class Animal {
  hablar() {
    return "El animal hace un sonido";
  }
}

class Perro extends Animal {
  hablar() {
    return "Guau Guau";
  }
}

class Gato extends Animal {
  hablar() {
    return "Miau Miau";
  }
}

const animales = [new Animal(), new Perro(), new Gato()];

for (const a of animales) {
  console.log(a.hablar()); // Cada uno se comporta distinto
}
```

```
const perro = {
  hablar: () => "Guau Guau"
};

const gato = {
  hablar: () => "Miau Miau"
};

function hacerHablar(obj) {
  console.log(obj.hablar());
}

hacerHablar(perro);
hacerHablar(gato);
```

Polimorfismo:

```
class Figura {
  area() {
    return 0;
  }
}

class Cuadrado extends Figura {
  constructor(lado) {
    super();
    this.lado = lado;
  }
  area() {
    return this.lado * this.lado;
  }
}

class Círculo extends Figura {
  constructor(radio) {
    super();
    this.radio = radio;
  }
  area() {
    return Math.PI * this.radio ** 2;
  }
}

const figuras = [new Figura, new Cuadrado(4), new Círculo(3)];
for (const f of figuras) {
  console.log(f.area()); // Polimorfismo
}
```

27. Math

- Math es un objeto global estático que proporciona funciones y constantes matemáticas
- No se instancia con new Math(), sino que se usa directamente como Math.función()
- Sirve para realizar operaciones comunes como redondeos, potencias, raíces, trigonometría, logaritmos y generación de números aleatorios

```
let x = 4.7;
console.log(Math.round(x));  // 5
console.log(Math.sqrt(16));  // 4
console.log(Math.random());  // número aleatorio entre 0 y 1
```

Categoría	Método / Constante	Descripción	Ejemplo
Constantes	Math.PI	Valor de π (3.14159...)	Math.PI
	Math.E	Base del logaritmo natural (≈2.718)	Math.E
	Math.SQRT2	√2 (≈1.414)	Math.SQRT2
	Math.LN10	ln(10)	Math.LN10
	Math.LOG10E	$\log_{10}(e)$	Math.LOG10E
Redondeo y signo	Math.round(x)	Redondea al entero más cercano	Math.round(4.6) → 5
	Math.floor(x)	Redondea hacia abajo	Math.floor(4.9) → 4
	Math.ceil(x)	Redondea hacia arriba	Math.ceil(4.1) → 5
	Math.trunc(x)	Elimina los decimales (trunca)	Math.trunc(4.9) → 4
	Math.sign(x)	Devuelve -1, 0 o 1 según el signo	Math.sign(-8) → -1
Máximo, mínimo, valor absoluto	Math.max(a,b,...)	Mayor de una lista	Math.max(2, 8, 5) → 8
	Math.min(a,b,...)	Menor de una lista	Math.min(2, 8, 5) → 2
	Math.abs(x)	Valor absoluto	Math.abs(-7) → 7
Potencias y raíces	Math.pow(a,b)	a elevado a b	Math.pow(2, 3) → 8
	Math.sqrt(x)	Raíz cuadrada	Math.sqrt(25) → 5
	Math.cbrt(x)	Raíz cúbica	Math.cbrt(27) → 3
	Math.hypot(a,b,...)	$\sqrt{(a^2 + b^2 + \ldots)}$	Math.hypot(3,4) → 5
Trigonometría	Math.sin(x)	Seno (x en radianes)	Math.sin(Math.PI/2) → 1
	Math.cos(x)	Coseno	Math.cos(0) → 1
	Math.tan(x)	Tangente	Math.tan(Math.PI/4) → 1
Logaritmos y exponenciales	Math.log(x)	Logaritmo natural (base e)	Math.log(Math.E) → 1
	Math.log10(x)	Logaritmo base 10	Math.log10(100) → 2
	Math.log2(x)	Logaritmo base 2	Math.log2(8) → 3
	Math.exp(x)	e^x	Math.exp(1) → ≈2.718
Aleatorios	Math.random()	Número aleatorio en [0, 1>	Math.random()

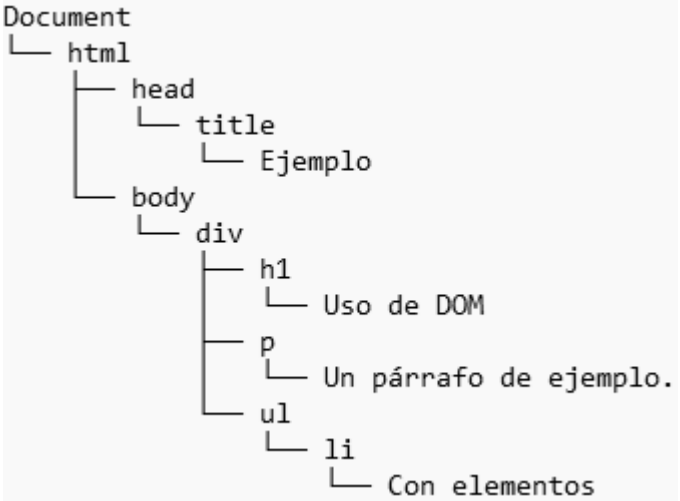
28. Clases y Objetos en el entorno web

Contexto	Cómo se usan los objetos/clases
DOM	Crear clases que manipulen elementos del HTML
Eventos	Métodos que responden a clicks, teclas, etc.
APIs REST	Manejar datos JSON como objetos JS
Librerías y Frameworks (React, Angular, Vue)	Componentes se basan en objetos y clases
Node.js (backend)	Clases para controladores, modelos, servicios, etc.

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]
```

Ejercicios

1. Crea una función que reciba dos arreglos (claves y valores) y devuelva un Map
2. Crea una función que use un Map para contar cuántas veces aparece cada número en un arreglo
3. Crea una función que reciba un Map y devuelva otro con claves y valores intercambiados
4. Crea una función que reciba un objeto persona con al menos 3 propiedades y cambie el valor de la propiedad ciudadResidencia
5. Crea una función que reciba un objeto, que recorra todas sus propiedades mostrándolas en consola y que devuelva cuántas propiedades tiene
6. Crea una clase EquipoFutbol con propiedades nombre y ciudad, constructor y un método saludar()
7. Crea una clase CuentaBancaria con titular y saldo, y métodos para depositar y retirar
8. Crear la clase base Persona con 3 propiedades y constructor. Crear las clases hijas Estudiante y Profesor que heredan de Persona y aumentar al menos 2 propiedades y 2 métodos diferentes a cada una. Crear un objeto de cada clase y probarlos. Crear un array con un objeto de cada una y una función para mostrar cada elemento usando toString

Ejercicios

9. Crear un documento HTML con tres párrafos. Usar JavaScript para seleccionar:
   - El primer párrafo por su id
   - Todos los párrafos por su etiqueta
   - Todos los elementos con una clase común
   - Muestra cada selección en la consola
10. Selecciona un elemento <h1> y cambia su texto a "DOM modificado con JavaScript"
    Luego, cambia su color y tamaño de fuente usando propiedades tipo CSS desde JS
11. Mediante JavaScript, crea una lista <ul> con tres elementos
    Luego agrega un cuarto elemento dinámicamente y elimina el segundo elemento
12. Selecciona un contenedor y muestra en consola:
    Su elemento padre, su primer hijo, su último hijo, Y la lista de todos sus nodos hijos