CI-0126 Ingeniería de Software
Prof. Rebeca Obando
Laboratorio #2 B

Anabelle Porras Carrillo
A03252

Link del repositorio Git: https://github.com/anabelleporras/a03252_labs_ci0126

1. ¿Cuáles son los data types que soporta javascript?
JavaScript soporta 7 tipos primitivos de datos: string, number, bigint, boolean, undefined, null, symbol. Y tipos no primitivos basados en objetos (objetos, arrays, funciones, etc.).

2. ¿Cómo se puede crear un objeto en javascript? De un ejemplo
Hay varias formas de crear objetos en JS, la más básica es usando llaves:

```
let persona = { nombre: "Ana", edad: 25 };
```

También se puede crear usando new Object:

```
let persona = new Object();
persona.nombre = "Ana";
persona.edad = 25;
```

También se puede crear una clase y después declarar un objeto de esa clase:

```
let estudiante = new Persona("Ana", 25);
```

3. ¿Cuáles son los alcances (scope) de las variables en javascript?
El scope (alcance) determina dónde es visible y accesible una variable en el código.
Existen principalmente tres tipos de alcance:
Global scope: accesible desde todo el programa.
Function scope: accesible sólo dentro de la función.
Block scope: accesible sólo dentro del bloque donde se declaró.

4. ¿Cuál es la diferencia entre undefined y null?
Undefined: el sistema lo asigna automáticamente cuando no se da un valor.
Null: el programador lo asigna para indicar ausencia intencional de valor.

5. ¿Qué es el DOM?
DOM es una interfaz de programación que permite a JavaScript interactuar con el contenido de la página. Convierte el HTML en una estructura en memoria (como un árbol jerárquico) que se puede leer, modificar, eliminar o crear dinámicamente. Cada elemento (ej. <h1>, <p>,) se representa como un objeto con propiedades y métodos.

6. Usando Javascript se puede acceder a diferentes elementos del DOM. ¿Qué hacen, que retorna y para qué funcionan las funciones getElement y querySelector? Cree un ejemplo

getElementById: Busca un elemento único con el atributo id especificado. Retorna un elemento (HTMLElement) o null si no existe. Funciona para acceder a un nodo específico que tiene un id.

getElementsByClassName: Busca todos los elementos que tengan cierta clase. Retorna un HTMLCollection (lista “en vivo”) de elementos. Funciona para seleccionar múltiples elementos con la misma clase.

getElementsByTagName: Busca todos los elementos de una etiqueta específica (ej. div, p, h1). Retorna un HTMLCollection de elementos. Funciona para manipular todos los elementos de un mismo tipo.

querySelector: Busca el primer elemento que coincida con un selector CSS. Retorna un elemento (HTMLElement) o null. Funciona para seleccionar un elemento de manera más flexible usando selectores de CSS.

querySelectorAll: Busca todos los elementos que coincidan con un selector CSS. Retorna un NodeList (estático), que se puede recorrer con forEach. Funciona para seleccionar múltiples elementos con selectores de CSS avanzados.

Ejemplo:

```
let h1 = document.getElementById("titulo");
h1.style.color = "red";

let textos = document.getElementsByClassName("texto");
textos[1].style.fontWeight = "bold";

let primerTexto = document.querySelector(".texto");
primerTexto.style.background = "yellow";

let todos = document.querySelectorAll(".texto");
todos.forEach(el => el.style.border = "1px solid blue");
```

7. Investigue cómo se pueden crear nuevos elementos en el DOM usando Javascript. De un ejemplo

Se pueden crear nuevos elementos usando el método document.createElement(), luego configurarlos (texto, atributos, clases, estilos, etc.) y finalmente se insertan en el documento con métodos como appendChild() o append().

Ejemplo:

```
// 1. Crear un nuevo elemento <li>
let nuevoItem = document.createElement("li");
```

```
// 2. Asignar contenido
nuevoItem.textContent = "Aprender DOM en JavaScript";

// 3. Agregar una clase
nuevoItem.className = "tarea";

// 4. Insertarlo dentro del <ul>
let lista = document.getElementById("lista");
lista.appendChild(nuevoItem);
```

8. ¿Cuál es el propósito del operador this?
El propósito de this es referirse al objeto actual en un contexto dado:
En métodos se refiere al objeto.
En clases se refiere a la instancia.
En global se refiere al objeto global.
En arrow functions hereda del contexto externo.
En eventos se refiere al elemento que dispara el evento.

9. ¿Qué es un promise en Javascript?De un ejemplo
Es un objeto que representa la finalización (o falla) de una operación asíncrona. Sirve para manejar tareas que tardan tiempo (como llamadas a APIs, temporizadores, operaciones de lectura/escritura) sin bloquear la ejecución del resto del programa.

Ejemplo:

```
let promesa = new Promise((resolve, reject) => {
  let exito = true;

  if (exito) {
    resolve("La operación fue exitosa");
  } else {
    reject("Hubo un error en la operación");
  }
});

// Consumir la promesa
promesa
  .then(resultado => console.log(resultado)) // si se cumple
  .catch(error => console.error(error))     // si falla
  .finally(() => console.log("Proceso terminado"));
```

10. ¿Qué es Fetch en Javascript? De un ejemplo
Es una API nativa del navegador que permite realizar peticiones HTTP (como GET, POST, PUT, DELETE) de manera asíncrona. Se utiliza para obtener o enviar datos a un servidor, con una sintaxis más simple y basada en Promises. Permite trabajar con JSON, texto, blobs, streams, etc.

Ejemplo:

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(response => response.json()) // convertir la respuesta a
JSON
  .then(data => console.log("Datos recibidos:", data))
  .catch(error => console.error("Error en la petición:", error));
```

11. ¿Qué es Asynch/Await en Javascript ? De un ejemplo

Async/Await es una forma más moderna y legible de trabajar con código asíncrono, construida sobre las Promesas Se declara una función con async, esta siempre devuelve una Promesa. Await se usa dentro de la función async y pausa la ejecución hasta que una Promesa se resuelva o se rechace.

Ejemplo:

```
async function mostrarDatos() {
  try {
    let datos = await obtenerDatos();
    console.log("Datos con async/await:", datos);
  } catch (error) {
    console.error(error);
  }
}

mostrarDatos();
```

12. ¿Qué es un Callback? De un ejemplo

Es una función que se pasa como argumento a otra función y que se ejecuta después de que esta última termina su tarea. Puede ser sincrónica (ejecutarse inmediatamente) o asíncrona (ejecutarse después de cierto tiempo o evento).
Fueron la base del manejo asíncrono en JS antes de las Promises y async/await.

Ejemplo:

```
function procesarTarea(callback) {
   console.log("Iniciando tarea...");
   setTimeout(() => {
   console.log("Tarea completada.");
   callback(); // Ejecuta la función callback después del retraso
   }, 2000); // Espera 2 segundos
}
function mostrarMensaje() {
```

```
   console.log("¡La operación ha finalizado!");
}
procesarTarea(mostrarMensaje);
```

13. ¿Qué es Clousure?

Es cuando una función recuerda el entorno en el que fue creada, permitiéndole acceder a variables externas incluso después de que ese entorno haya finalizado.
Cada vez que se crea una función en JavaScript, se guarda también el ámbito (scope) en el que fue definida.
Si esa función se retorna o se pasa a otro lugar, seguirá teniendo acceso a esas variables externas.

Ejemplo:

```
function crearContador() {
  let contador = 0; // variable privada

  return function() {
    contador++;
    return contador;
  };
}

let incrementar = crearContador();

console.log(incrementar()); // 1
console.log(incrementar()); // 2
console.log(incrementar()); // 3
```

14. ¿Cómo se puede crear un cookie usando Javascript?

Se crea asignando un valor a la propiedad document.cookie.
Una cookie es simplemente un par clave–valor, al que se le pueden añadir atributos como fecha de expiración, ruta y dominio. Pueden tener opciones (expires, path, secure).
Son útiles para guardar información del usuario (sesiones, preferencias).

15. ¿Cuál es la diferencia entre var, let y const?

var: se usa para declarar variables, su scope es la función pero no respeta el bloque {}, por eso hoy casi no se usa.
let: se usa cuando el valor de la variable puede cambiar (respeta el block scope).
const: se usa por defecto, para valores que no deberían reasignarse (también respeta el block scope).

Referencias

[1] L. Turton, “JavaScript Essentials,” Udemy, Accessed Aug. 25, 2025. [Online]. Available: https://www.udemy.com/course/javascript-essentials/

[2] “JavaScript Tutorial,” W3Schools, Accessed Aug. 25, 2025. [Online]. Available: https://www.w3schools.com/js/