Ingeniería de Software **Laboratorio #2b**
Emanuel García Rojas C23166

1. **¿Cuáles son los data types que soporta javascript ?**
    - Los data types que soporta javascript son 8 según [1]:
        - String
        - Number
        - Bigint
        - Boolean
        - Undefined
        - Null
        - Symbol
        - Object

2. **¿Cómo se puede crear un objeto en javascript? De un ejemplo**
    - Se define el nombre del objeto que se desea, siendo una variable const, se definen las propiedades de este objeto dentro de los {}, ya sean variables y/o métodos [2].
    - Una opción para crear un objeto es la siguiente:
        - Ejemplo:

```
const persona = {
        nombre:"Emanuel",
        apellido:"García",
        edad:21,
        colorCabello:"café",
        comer: function() {
                console.log(`${this.nombre} está comiendo.`);
          },
};
```

3. **¿Cuáles son los alcances (scope) de las variables en javascript?**
    - Según [3] Javascript tiene 3 niveles de scopes con sus variables:
        - Block scope
        - Function scope
        - Global scope

4. **¿Cuál es la diferencia entre undefined y null?**
    - La diferencia es que undefined es cuando se tienen variables no definidas, no se expresa un return en una función o cuando se accede a elementos que no existen

en un arreglo o objeto. Por otro lado, null indica un “no valor” o cuando intencionalmente se quiere “resetear”o “limpiar” una variable [4].

5. **¿Qué es el DOM?**
    - Según [5], el DOM (Document Object Model) es una interfaz de programación que permite que los programas y scripts accedan y actualicen dinámicamente el contenido, la estructura y el estilo de un documento. Se separa en 3 diferentes partes:
        - Core DOM, modelo estándar para documentos de todo tipo
        - XML DOM, modela estándar para documentos XML
        - HTML DOM, modelo estándar para documentos HTML

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**
    - La función “getElement” se encarga de buscar un elemento en un archivo HTML, entre las principales funciones de este tipo están: “getElementById”, “getElementByClassName” y “getElementByTagName”, cada una de ellas buscan los elementos por sus respectivos identificadores. Estas funciones suelen retornar los elementos buscados en caso de que los haya encontrado o retornan “null” si no encontraron estos elementos [6].
        - Ejemplo (para cambiar el color del texto de la lista de cuadernos a verde):

          ```
          const miElemento = document.getElementById("listaDeCuadernos");
          miElemento.style.color = "green";
          ```

    - Por otro lado, la función “querySelector” realiza las búsquedas pero que coinciden con el selector CSS especificado. Retorna el primer elemento que coincide con las especificaciones del selector CSS, o si no lo encuentra, retorna “null” [7].
        - Ejemplo (cambiar el primer cuaderno de la lista de cuadernos que encuentre):

          ```
          const primerCuaderno = document.querySelector(".listaDeCuadernos li");
          primerCuaderno.style.color = "yellow";
          ```

7. **Investigue cómo se pueden crear nuevos elementos en el DOM usando Javascript. De un ejemplo**
    - Según [8], se pueden crear nuevos elementos en el DOM usando la función “createElement”.
        - Por ejemplo (agregar un nuevo párrafo dentro de un contenedor existente):

```
const contenedor = document.getElementById("contenedor");
const nuevoParrafo = document.createElement("p");
nuevoParrafo.textContent = "Mi nombre es Emanuel García Rojas";
contenedor.appendChild(nuevoParrafo);
```

8. **¿Cuál es el propósito del operador this?**
    - Según [9] el operador this tiene como propósito el de hacer referencia al contexto en el que se presenta en ese momento específico. Tiene varios casos comunes como los siguientes:
        - Puede hacer referencia al objeto global
        - En caso de uso con objetos hace referencia al propio objeto al que pertenece un método, por ejemplo
        - En una clase o constructor hace referencia a la instancia del objeto que se crea con la clase o constructor

9. **¿Qué es un promise en Javascript?De un ejemplo**
    - Según [10], un promise en JavaScript es un objeto que representa la posible resolución o rechazo de una operación asíncrona. Permite manejar tareas que toman tiempo, como solicitudes a un servidor o temporizadores.
        - Tiene 3 posibles estados
            - Pending, la operación no ha finalizado
            - Fullfilled, la operación se completó con éxito
            - Rejected, la operación tuvo un error
        - Ejemplo (se encarga de revisar si el número es par, si lo es, resuelve, sino lo rechaza). Este ejemplo fue extraído de [10]:

```
let revisarPar = new Promise((resolver, rechazar) => {
   let numero= 4;
   if (numero % 2 === 0) resolver("El número es par");
   else reject("El número es impar");
});
revisarPar
   .then((message) => console.log(message)) // Éxito
   .catch((error) => console.error(error)); //  Fallo
```

10. **¿Qué es Fetch en Javascript? De un ejemplo**
    - Es una interfaz de API que permite al navegador web realizar solicitudes HTTP a los servidores web y poder procesar estas respuestas [11].

- Ejemplo (se busca información de un producto específico, ID 1, desde la API "Fake Store" y muestra los resultados). Este ejemplo fue extraído de [11]:

```
fetch('https://fakestoreapi.com/products/1')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

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

- Es usado para simplificar el manejo de operaciones asincrónicas mediante promesas. Al permitir que el código asincrónico parezca sincrónico, mejoran la legibilidad del código y facilitan la administración de flujos asincrónicos complejos [12].
  - Ejemplo (busca información específica sobre un "post" desde una API de prueba, JSONPlaceholder, y la imprime). Este ejemplo fue extraído de [12]:

```
async function fetchData() {
 const response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
 const data = await response.json();
 console.log(data);
}

fetchData();
```

12. ¿Qué es un Callback? De un ejemplo

- Según [13] es una función que se pasa como argumento a otra función, esto permite que una función llame a otra función. Un callback puede ejecutarse después de que su función haya finalizado.
  - Ejemplo (se saluda luego se usa el callback, el cual sería la despedida):

```
function saludar(nombre, callback) {
 console.log(`Hola, ${nombre}!`);
 callback();
}

function despedida() {
 console.log("¡Hasta luego! Que tenga un buen día.");
```

```
}

saludar("Emanuel", despedida); // se usa el callback
```

13. ¿Qué es Closure?

- Un closure es una función que permite el acceso a las variables desde su función externa y variables globales, incluso después de que la función externa haya terminado de ejecutarse. Esto permite que las funciones "recuerden" su entorno [14].

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

- Según [15], las cookies usando JavaScript se pueden crear por medio del uso de la propiedad document.cookie.
  - Por ejemplo:

```
document.cookie = "nombreDeUsuario=Emanuel García";
```

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

- La diferencia const se usa cuando el valor o el tipo no será cambiado en ningún momento. Se debe usar let, si no importa si el valor puede cambiar. Por último, var se debe utilizar principalmente para el soporte a navegadores más antiguos ya que puede prestarse a confusiones. Otro punto distintivo es que var tiene un scope de función y por otro lado const y let tienen un alcance de bloque [16].

16. Enlace al repositorio

- **Referencias**

  - [1] W3Schools. “JavaScript Data Types.” *W3schools.com*, 2020, www.w3schools.com/js/js_datatypes.asp.
  - [2] W3Schools. “JavaScript Objects.” *W3schools.com*, 2019, www.w3schools.com/js/js_objects.asp.
  - [3] W3Schools. “JavaScript Scope.” *W3schools.com*, 2019, www.w3schools.com/js/js_scope.asp.
  - [4] GeeksForGeeks, “Undefined Vs Null in JavaScript,” GeeksforGeeks, Nov. 01, 2020. https://www.geeksforgeeks.org/undefined-vs-null-in-javascript/.
  - [5] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_htmldom.asp. (Accesado en Mar. 26, 2025).
  - [6] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_htmldom_elements.asp. (Accesado en Mar. 26, 2025).

- [7] W3Schools, “W3Schools.com,” W3schools.com, 2025. http://www.w3schools.com/jsref/met_document_queryselector.asp. (Accesado en Mar. 26, 2025).
- [8] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_htmldom_document.asp. (Accesado en Mar. 26, 2025).
- [9] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_this.asp. (Accesado en Mar. 26, 2025).
- [10] GeeksForGeeks, “JavaScript Promise,” GeeksforGeeks, Jan. 02, 2019. https://www.geeksforgeeks.org/javascript-promise/.
- [11] GeeksForGeeks, “JavaScript | fetch() Method,” GeeksforGeeks, Jun. 17, 2020. https://www.geeksforgeeks.org/javascript-fetch-method/.
- [12] GeeksForGeeks, “Async/Await Function in JavaScript,” GeeksforGeeks, Aug. 29, 2019. https://www.geeksforgeeks.org/async-await-function-in-javascript/.
- [13] GeeksforGeeks, “Callbacks Vs Promises Vs Async/Await,” GeeksforGeeks, Jan. 23, 2024. https://www.geeksforgeeks.org/callbacks-vs-promises-vs-async-await/. (Accesado en Mar. 26, 2025).
- [14] GeeksforGeeks, “Closure in JavaScript,” GeeksforGeeks, Oct. 09, 2017. https://www.geeksforgeeks.org/closure-in-javascript/.
- [15] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_cookies.asp. (Accesado en Mar. 26, 2025).
- [16] W3Schools, “W3Schools.com,” W3schools.com, 2015. http://www.w3schools.com/js/js_variables.asp. (Accesado en Mar. 26, 2025).