Laboratorio 3B

Felipe Bianchini, C21178

1. ¿Cuáles son los data types que soporta javascript?

R/ strings, numbers (enteros o con decimales), booleans, null, undefined, NaN, symbols, arrays y objetos.

2. ¿Cómo se puede crear un objeto en javascript? De un ejemplo

R/ Los objetos pueden contener datos y funciones y se crea de la siguiente manera:

```
var car= {
     color: red,
     speed: 200,
     drive: function() { return "driving"; }
};
```

3. ¿Cuáles son los alcances (scope) de las variables en javascript?

R/ Alcance global (se accede desde cualquier parte del programa), alcance de función (se accede sólo desde la función en donde fue declarada) y alcance de bloque (se accede solo dentro del bloque donde fue declarada).

4. ¿Cuál es la diferencia entre undefined y null?

R/ null significa ausencia de algún valor o que algún dato está vacío, y es asignado por el programador, mientras que undefined indica variables no inicializadas o acceso a datos que no existen, y es asignado por el sistema.

5. ¿Qué es el DOM?

R/ El DOM es una interfaz que organiza los documentos HTML en una estructura de árbol cuyos nodos son estructuras del documento y el nodo raíz siempre es

“document”. Permite modificar páginas web de manera dinámica. Se puede manipular desde Javascript.

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

R/

Los diferentes tipos de getElement que hay buscan en el DOM de acuerdo con el parámetro especificado y retornan el primero que encuentren. Se puede buscar por id, por tag o por class. Por ejemplo:

```
const unlistedList = document.getElementById("lista");
```

Esta línea de código busca en el DOM el primer nodo que contenga el id “lista”.

querySelector busca en el DOM de acuerdo con el selector CSS especificado y retorna el primero que encuentre. Por ejemplo:

```
const elemento = document.querySelector('nav ul');
```

Esta línea de código busca en el DOM el primer nodo con el selector CSS “nav ul”.

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

R/ Se crean mediante el método createElement(), el cual recibe como parámetro el nombre de una etiqueta HTML. Lo que hace es crear un elemento del tipo especificado al cual se le puede agregar texto u otras características para, finalmente, añadirlo al DOM mediante los métodos appendChild() o insertBefore().

```
const nuevo = document.createElement('li');
nuevo.textContent = "Elemento" + (numElemento + 1);
unlistedList.appendChild(nuevo);
```

Estas líneas de código crean un list item, le ponen texto y lo unen a una unlisted list.

8. ¿Cuál es el propósito del operador this?

R/ Se usa más que todo en objetos y funciones y permite acceder a variables de ese mismo contexto, por ejemplo:

```
const Car {
     marca: Toyota;
     getMarca(): function {
          return this.marca;
     }
};
```

En este ejemplo, el uso de this permite acceder a una variable del contexto en el que se encuentra el método, que es el objeto Car.

9. ¿Qué es un promise en Javascript? De un ejemplo

R/ Es un objeto que contiene el resultado de una operación asincrónica. Tiene tres métodos: .then (maneja el resultado de la promesa o el error), .catch (maneja el error) y .finally (se ejecuta al final de la promesa, sin importar si hubo error o no). Por ejemplo:

```
const promise = new Promise(resolve, reject) => {
     if (condicion) {
          resolve("Operación exitosa");
     } else {
          reject("Operación fallida");
     }
};

promise
     .then(result => {
          console.log(result);
     })
     .catch(error => {
          console.error(error);
     })
     .finally(() => {
```

```
            console.log("Operación simulada con una promise");
      });
```

10. ¿Qué es Fetch en Javascript? De un ejemplo

R/ Es una API que permite la comunicación con servidores. Devuelve una promise que contiene la respuesta del servidor. Por ejemplo, para realizar una solicitud GET:

```
const url = "url_servidor"
fetch(url)
      .then(response => {
            if (!response.ok) {
                  throw new Error('Error de conexión');
            } else {
                  return response;
            }
      })
```

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

R/ Permiten trabajar con funciones asincrónicas. Async se usa para declarar que una función se ejecutará de forma asincrónica, y esa función debe de devolver una promise. Await se usa dentro de las funciones async para esperar que la promise de la función se ejecute, para lo cual debe de detener la ejecución de la función, ejecutar la promesa, y luego seguir con la ejecución de la función. Por ejemplo:

```
function operacion(ms) {
      return new Promise((resolver) = > {
            set Timeout(() {
                  resolve("operacion realizada");
            }, ms);
      });
}
```

```
async function operacionAsincronica() {
     console.log("Inicia");
     const op = await operacion(1000);
     console.log(op);

     console.log("Termina");
}
operacionAsincronica();
```

La función operacionAsincronica simula la operación de algo, para lo cual utiliza async para declarar que la función será asincrónica y luego utiliza await para esperar la ejecución de la función operacion, la cual devuelve la operación luego de 1 segundo.

12. ¿Qué es un Callback? De un ejemplo

R/ Es una función que recibe como argumento otra función y que permite que se ejecute la función que recibe como argumento en algún momento específico de su ejecución de manera asincrónica, por ejemplo:

```
function leer(callback) {
     console.log("Empieza a leer página");
     setTimeout(() => {
          console.log("leyendo");
          callback();
     }, 60000);
}

function pasarPagina() {
     console.log("Pasa página");
}
leer(pasarPagina());
```

La función leer simula la lectura de una página y recibe como parámetro la función pasarPagina. La idea es que se empieza a leer una página, se continua leyendo y despues de 60 segundos se llama a la función pasarPagina mediante el callback, la cual pasa de página.

13. ¿Qué es Clousure?

R/ Es una función interna que tiene acceso a las variables de la función externa de la cual es parte a pesar de que la función externa ya terminó de ejecutarse. Es útil ya que permite el encapsulamiento de datos al mantener las variables de la función externa como atributos privados que se acceden y se modifican solo desde las funciones internas y también porque permite la persistencia de datos, por ejemplo:

```
function sumar() {
     let num = 0;
     return function() {
          num++;
          return num;
     }
}
const numero = sumar();
console.log(numero()); // Ahora num es 1
console.log(numero()); // Ahora num es 2
```

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

R/ Se puede crear con el comando `document.cookie = "nombre=valor"`. También permite asignarle una fecha de expiración y una ruta.

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

R/

var: su alcance es de función o global, permite re declaraciones y también re asignaciones.

let: su alcance es de bloque, no permite re declaraciones, permite re asignaciones.
const: su alcance es de bloque, no permite re declaraciones, no permite re asignaciones.