

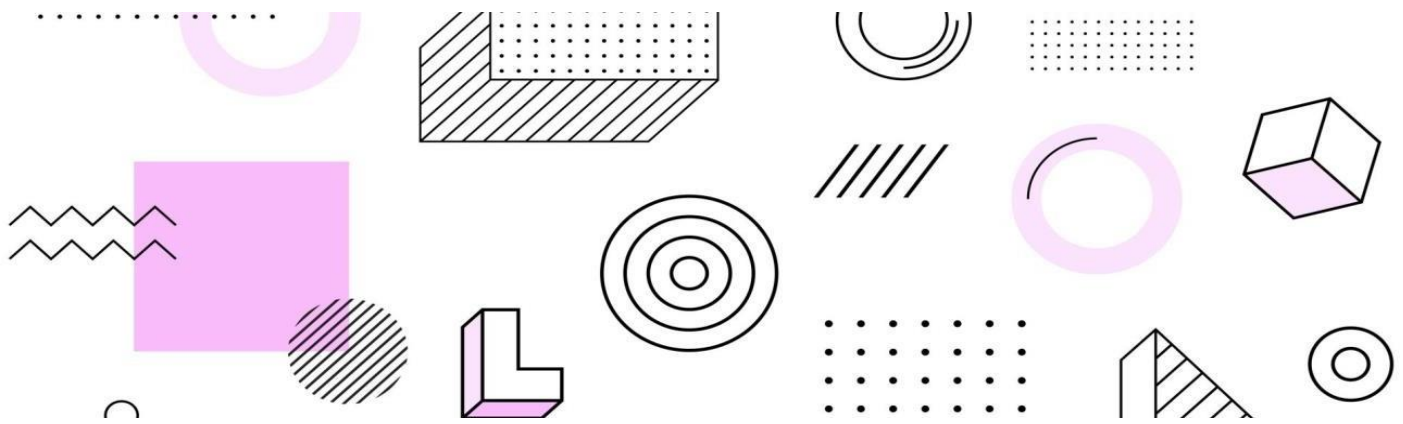
Taller de nivelación



Presentado por:
Mannuel Fernando Granoble Pino

Presentado a:
Ing. Whitney Stevenson
Ing. Ana Ramírez

Makaia
Desarrollo web Frontend
Corte 6
Popayán, 20 de Noviembre del 2023



Taller de Nivelación Curso Desarrollo Web Frontend

El presente taller tiene el propósito de evaluar la comprensión y aplicación de conceptos fundamentales del desarrollo web Frontend desde los conceptos de lógica de programación, HTML, CSS, JavaScript, React Js, los diferentes tipos de Hooks, su uso adecuado y cómo utilizar React Router DOM v6 para la navegación en una aplicación React. Se busca nivelar y asegurar la apropiación de los conceptos vistos hasta el momento en el curso de desarrollo web frontend, la implementación de ReactJS y sus tecnologías asociadas.

MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT

Preguntas teóricas

1. ¿Qué es la lógica en el contexto de la programación? Y explicar por qué es importante en el desarrollo web Frontend.

RTA:

Para mi la lógica es algo que todos tenemos, pero que en cada uno de nosotros es diferente, puesto que es una mezcla de como vemos las cosa, como las entendemos y como las interpretamos, es algo abstracto e innato, pero de suma importancia ya que en gran parte es la perspectiva que nosotros tenemos de lo que nos rodea.

En el ámbito de la programación es como percibimos un problema, lo entendemos y tratamos de generar una posible solución de manera clara, coherente, precisa y lo más optima posible, en sí, es la manera en cómo estructuramos y generamos el paso a paso de instrucciones que debe seguir la máquina para poder llevar a cabo una.

En resumen es muy importante, puesto que para un problema pueden existir n posibles soluciones pero dependiendo de la lógica de cada persona, esa solución puede ser muy complicada o muy simple, puesto que el desglose de los pasos a seguir puede ser diferente y por ende la estructuración del código varía según la perspectiva de cada uno, por eso el pulir la lógica de cada uno de nosotros es muy importante, no solo en el ámbito de la programación sino en muchos aspectos de la vida diaria.

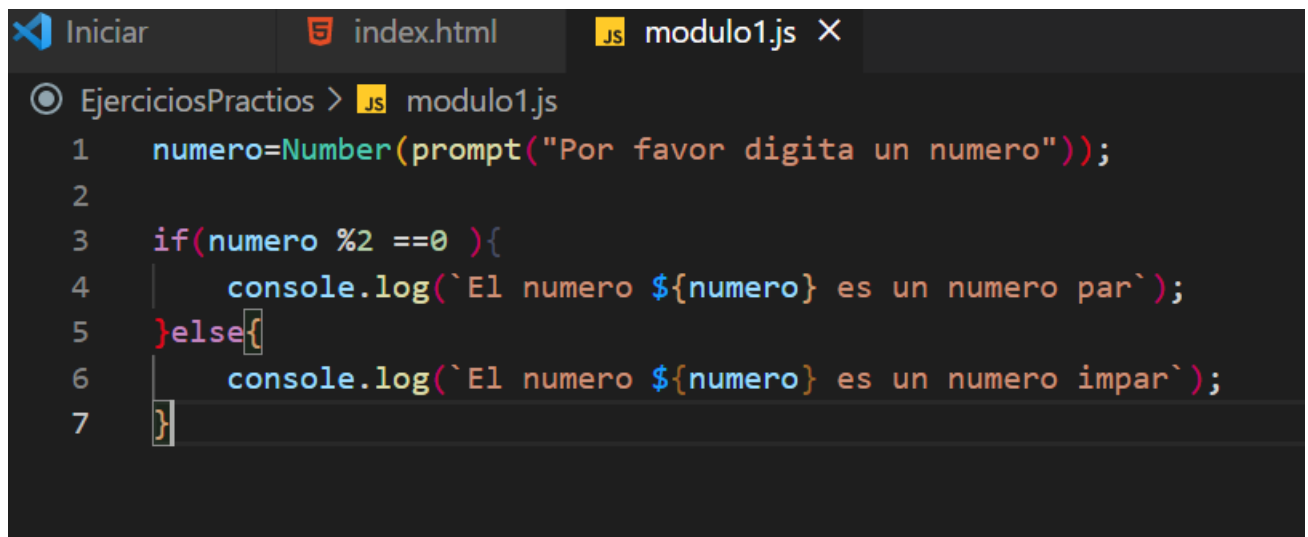
2. Definir el concepto de “algoritmo” y proporcionar un ejemplo sencillo de un algoritmo relacionado con la lógica de programación.

RTA:

Un algoritmo es una serie de pasos a realizarse para brindar solución a un problema o tarea, éste debe cumplir unas normas importantes: primero, debe ser finito, es decir, debe tener un principio y un fin, porque su meta siempre será solucionar una tarea, segundo, deben de ser lo mas óptimos posible, puesto que pueden existir n posibles soluciones a un problema, pero, aunque todas esas soluciones lleguen a un mismo punto, no significan que sean las mejores, ya que en programación los recursos de las maquinas son limitados y generar un algoritmo de gran complejidad, ilegible y que de muchas vueltas para llegar a un mismo punto, puede afectarla, aumentando el tamaño del proyecto, consumiendo mas recursos de los necesarios, etc.

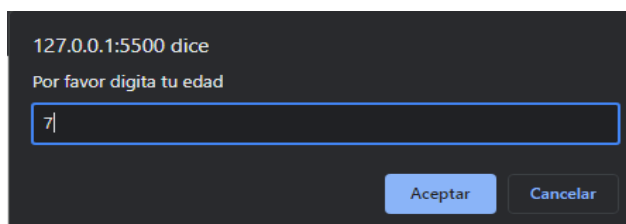
En resumen, un algoritmo es una serie de pasaos generados de manera lógica, ordenada, coherente y bien definidos, los cuales generan una solución a un problema o tarea y para ello debe de tener un inicio y un fin, deben ser bien definido, lo más óptimos posibles y tener un propósito u objetivo en específico.

Ejemplo un número es par o impar:



```
Inicio index.html JS modulo1.js X
EjerciciosPractios > JS modulo1.js
1  numero=Number(prompt("Por favor digita un numero"));
2
3  if(numero %2 ==0 ){
4      console.log(`El numero ${numero} es un numero par`);
5  }else{
6      console.log(`El numero ${numero} es un numero impar`);
7  }
```

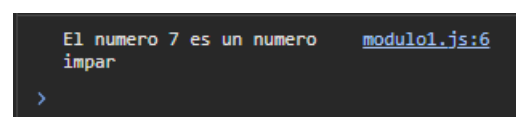
Entrada:



127.0.0.1:5500 dice
Por favor digita tu edad

Aceptar Cancelar

Salida:



```
El numero 7 es un numero impar modulo1.js:6
>
```

3. ¿Qué son estructuras de control en la programación?, ¿Cuáles son los tipos de estructuras de control y las estructuras más comunes de cada tipo?, Describir al menos dos tipos de estructura de control, explicar por qué son importantes

RTA:

Las estructuras de control son herramientas que contribuyen con la toma de decisiones o control del flujo de un programa, facilitando así el mejor funcionamiento del mismo

Existen 3 tipos de estructuras de control:

1. secuenciales, donde resaltan la declaración de variables y la asignación, estas estructuras se ejecutan una después de la otra.
 2. De selección o decisión, en ella destacan el if (si) y el else (sino), estas permiten preguntar, si una variable cumple o no x condición y dependiendo SI (IF) es verdadera la afirmación, ejecuta cierto bloque de código y SINO (ELSE) ejecuta otro bloque de código determinado, en sí estas permiten tomar decisiones en función de ciertas condiciones y realizar acciones respecto a ellas.
 3. Repetitivas o bucles, en ellas se destacan el for (para), el cual ejecuta un bloque de código un número específico de veces y el while (mientras), el cual permite ejecutar un bloque de código mientras se cumpla una condición.
4. Describir cómo se declaraban variables y constantes en JavaScript antes de la introducción de ECMAScript 6 (ES6). Explicar cómo ES6 mejoró la declaración de variables y constantes, y mencionar los problemas que esta mejora resuelve en el desarrollo web Frontend.

RTA:

Antes se declaraban las variables con la palabra reservada `var` y el nombre de la variable en minúscula y las constantes, no tenían palabra reservada, lo que se hacía era que el nombre de la constante fuera en mayúscula y se esperaba que su valor no fuera modificado, ejemplo: `var numero1` y `var NUMERO1`, la primera era una variable y la segunda una constante, el problema era que las variables solo tenían un alcance más limitado pues solo existían solo en la función del código donde fuera declarada, en cambio la constante tenía un alcance global en el código, pero no tenía inmutabilidad es decir que se podía modificar sin problema.

ECMAScript 6 introdujo las palabras `let` para variables y `const` para constantes, entonces con `let` las variables poseen un alcance de bloque, es decir solo existen y son accesibles en el espacio de código que son declaradas, esto ayuda a prevenir problemas de sobrescritura y fugas de variables y con `const` asegura que a la constante, no se le pueda modificar su valor, contribuyendo con la integridad de los datos a lo largo de todo el programa, sin contar que el código se vuelve más legible, disminuye problemas de alcance, mejora la calidad del código.

5. ¿Cómo se declaran las funciones en JavaScript y cuál es la diferencia entre una declaración de función, una expresión de función y una función de flecha (arrow function)? Proporcionar ejemplos de cada una.

RTA:

- **Declaración de función:** se debe poner la palabra clave (function) el nombre de la función, entre paréntesis '()', si recibe o no parámetros, dentro de llaves ({}), el cuerpo de la función es decir el bloque de código que ejecutara cuando sea invocada.

```
EjerciciosPracticos > modulo1.js > ...
1  function isPar ( numero)
2  {
3      if(numero %2 ==0 ){
4          console.log(`El numero ${numero} es un numero par`);
5      }else{
6          console.log(`El numero ${numero} es un numero impar`);
7      }
8  }
9
10 //Funcion para saber si un numero es par o impar
11 const resultado = isPar(8);
12
```



```
El numero 8 es un numero modulo1.js:4
par
```

- **Expresión de función:** en esta, no es necesario ponerle un nombre, puesto que es cuando defines una función como parte de una expresión y la asignas a una variable, pero si lleva la palabra reservada (function)

```
1
2 //Funcion para saber si un numero es par o impar
3 const isPar = function ( numero)
4 {
5     if(numero %2 ==0 ){
6         console.log(`El numero ${numero} es un numero par`);
7     }else{
8         console.log(`El numero ${numero} es un numero impar`);
9     }
10 }
11
12 const resultado = isPar(5);
```



```
El numero 5 es un numero modulo1.js:8
impar
```

- **Función de flecha (Arrow function):** se representan con el símbolo (=>) y no tienen su propio objeto this, estas no contienen la palabra reservada (function) y llevan su propio nombre precedidas por la palabra const

```
2 //Funcion para saber si un numero es par o impar
3 const isPar = (numero) =>
4 {
5     if(numero %2 ==0 ){
6         console.log(`El numero ${numero} es un numero par`);
7     }else{
8         console.log(`El numero ${numero} es un numero impar`);
9     }
10 }
11
12 const resultado = isPar(10);
```



```
El numero 10 es un numero modulo1.js:6
par
```

Las diferencias entre ellas son la forma en como se declaran, su alcance, puesto que la primera por declaración siempre se declara al correr el programa entonces no se tendría inconveniente a la hora de invocarlas, caso contrario a las demás, en especial la de flecha, puesto que si se invoca antes de declararla no se podría acceder a ella y que las funciones flecha se usan mas para funciones simples y anónimas, pero en si las 3 nos ayudan a programas, depende de cada persona como y cuando las utiliza.

6. ¿Por qué es necesario el uso de funciones en el desarrollo web Frontend? Enumerar al menos tres razones fundamentales y proporcionar ejemplos de situaciones en las que las funciones son esenciales. Además, mencionar la ventaja de las funciones flecha en el contexto de estas razones.

RTA:

El uso de las funciones es importante, por que ellas encapsulan un fragmento específico de código, permitiendo que esté sea más legible, reutilizable gracias a la modularidad, que el mantenimiento sea mas fácil, que la escalabilidad se pueda realizar sin problema alguno y sin afectar a los demás componentes del programa, que la optimización de código sea la mejor posible y que rendimiento del programa sea más óptimo.

Ejemplo: Reutilización de código: es importante, ya que no ahorra volver a hacer n veces algo que solo necesitamos hacerla una vez, por ejemplo, supongamos que se crea una app web para compra de ropa, entonces tienes varios tipos de ropa, niños, adultos. Hombre mujer, etc. Entonces cada vez que ingrese el cliente y escoge una prenda en específico se debería validar si esa prenda esta disponible en la talla que desea, entonces envés de hacer esa validación por las n prendas que tenga la app, lo realizan una sola vez en una función y la invocas cada vez que sea necesario, es decir cada q se seleccione una prenda.

Las funciones fechas, son buenas en cualquiera de estos ámbitos, pues estas mejoran la legibilidad del código, gracias a su sintaxis más corta y precisa.

7. ¿Cuál es la diferencia entre parámetro y argumento?

RTA:

La diferencia es que un parámetro es el nombre de la variable que se pone en la de declaración de la función, la cual representa un valor, que está espera recibir cuando es llamada.

En cambio, un argumento es un valor que se envía a la función cuando esta esta siendo llamada y dicho valor o valores se asignan automáticamente al o los nombres de los argumentos que recibe dicha función.

como sabemos, estos pueden ser opcionales, puesto que una función puede o no recibir parámetros y por ende sino recibe parámetro no hay que enviarle argumentos.

8. Definir el concepto de Callback y proporcionar un ejemplo práctico.

RTA:

Callback es una función que se pasa como argumento a otra función y que se ejecuta según sea la necesidad.

Ejemplo:

```
EjerciciosPracticos > module1.js > ...
1
2 //Funcion para saber si un numero es par o impar
3 function isPar (numero, callback)
4 {
5     callback();
6     if(numero %2 ==0 ){
7         console.log(`El numero ${numero} es un numero par`);
8     }else{
9         console.log(`El numero ${numero} es un numero impar`);
10    }
11 }
12
13 // función de callback
14 function callback() {
15     console.log("El resultado es: ");
16 }
17
18 const resultado = isPar(81, callback);
19
```



El resultado es:

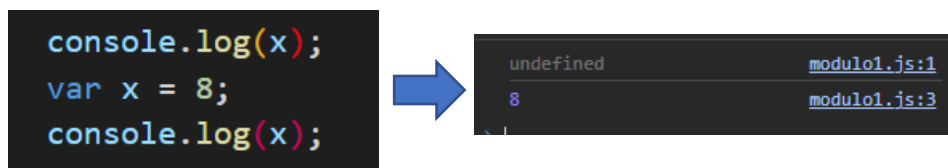
El numero 81 es un numero impar

9. ¿Qué es el hoisting en JavaScript y cómo afecta a las variables y funciones? Proporcionar ejemplos de hoisting en declaraciones de variables y funciones.

RTA:

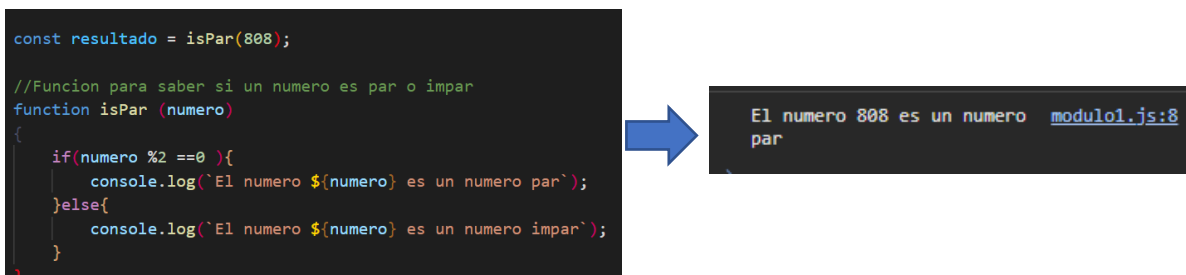
El hoisting hace que, al momento de la compilación, previo a la ejecución de un programa la declaración de variables y funciones sean llevadas al inicio del bloque de código donde estén declaradas, lo que permite que se pueda llamar dichas variables o funciones en cualquier momento del código, incluso antes de ser declaradas, pero es importante aclarar que el hoisting las afecta de tal manera que esa elevación solamente aplica para la declaración, mas no para la asignación de los valores.

Ejemplo hoisting variables:



Podemos ver como el hoisting afecta a las variables, puesto que dicha variable si existe, pero su valor no, ya que la elevación solo aplica para declaraciones, excluyendo el valor asignado en primera instancia, por ende, dicha variable asume un valor undefined.

Ejemplo hoisting funciones:



Aquí podemos ver que, aunque la función al ser llamada no está declarada, no genera un error, puesto que al ser elevada gracias al hoisting, esta ocupa una posición superior a cualquier llamado, por ende, en ese contexto si existe.

10. Definir brevemente el concepto de objeto en JavaScript y cuál es la visión general sobre este concepto. Indicar, también cómo se declaran estas estructuras de datos.


RTA:

Un objeto es una abstracción de la realidad a nivel de código, pues esta es una estructura que nos permite almacenar y organizar información, con la condición que esta información debe ser relacionada, que debe tener un nombre global que la identifique, que cada una de la información contenida en este debe tener una dualidad de clave valor, es decir un nombre que identifique a cada propiedad y un valor asignado a está. Los objetos son dinámicos es decir que se puede manipular, modificar, eliminar, está la información que estos contienen.

Existen dos maneras de declarar objetos, vacíos o con propiedades, la regla general es que ellos debes de tener un tip de dato, por lo general var o const seguidos de un nombres global, luego de un signo igual (=) y por ultimo llaves , tanto de apertura como de cierre({}), si esta llaves se dejan vacías, significa que el objeto es vacío, pero si dentro de las llaves se agrega contenido significa que es un objeto con propiedades.

```
//Objeto vacio
const carro = {};

//Objeto con propiedades
const perro = {
  raza: 'Chigugua',
  edad: 15,
  color: 'negro'
}
```



Clave: valor

11. ¿Qué son propiedades?, y ¿Cuál es la diferencia entre una propiedad y un método en un objeto?

RTA:

En resumidas cuentas, las propiedades son valores propios asociados a un objeto, donde cada una de ellas tiene una relación de clave valor, es decir el nombre de la propiedad y el valor a asociado a ella que a su vez perteneces al objeto que las contiene, en cambio un método en un objeto es una propiedad que tiene asignada una función como su valor y dicha función cumple una labor predeterminada y esta puede ser llamada a través del punto (.).

12. Explicar las dos formas de acceder a una propiedad de objetos e indicar las situaciones en que conviene usar una manera sobre la otra.

RTA:

Las dos formas para poder acceder a las propiedades de un objeto es a través de punto (.) o a través de corchetes ([])

```
//Objeto con propiedades
const perro = {
  raza: 'Chigugua',
  edad: 15,
  color: 'negro'
}

//acceder a través de punto
console.log(perro.color);

//acceder a través de corchetes
console.log(perro['color']);
```



negro	modulo1.js:23
negro	modulo1.js:26

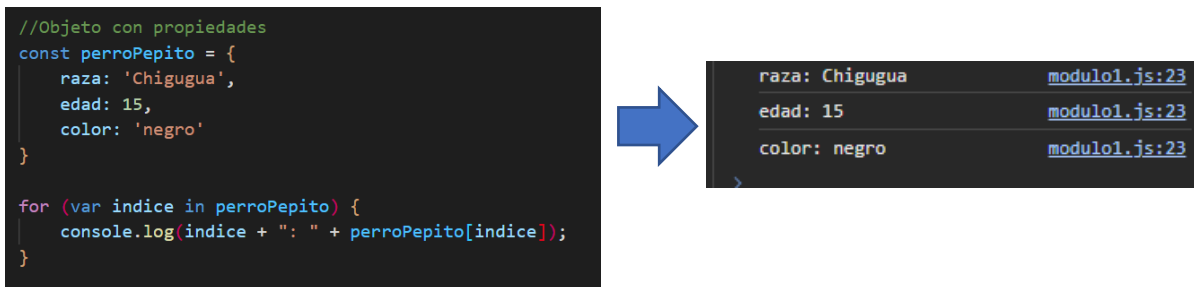
Como se puede ver de ambas maneras llegamos al mismo resultado, pero se aconseja utilizar la notación con punto (.), cuando se sabe el nombre de la propiedad y está compuesto por contenido alfanumérico es conocido, pues es más rápido y de sintaxis más simple, en cambio la notación por corchetes se recomienda usar cuando el nombre de la propiedad esta contenido es una variable o cuando se necesitas expresiones para poder acceder a ella o cuando el nombre de la propiedad esta compuesto por caracteres espaciales o espacios.

13. ¿Existe alguna forma de recorrer las propiedades de un objeto? En caso negativo, explicar por qué no es posible y en caso positivo proporcionar un ejemplo. Mencionar una situación en la cual sea muy útil recorrer las propiedades de un objeto.

RTA:

Un objeto se puede recorrer a través del bucle for in.

Ejemplo: Poder recorrer un objeto es importante, supongamos que llevo a una cita medica a mi perrito pepito, a mi veterinaria de confianza, lo examinan y demás y el veterinario desea saber información esencial para la historia clínica y análisis de pepito, entonces con poner el nombre, el software automáticamente encuentra el objeto llamado pepito con toda su información pertinente y así evitándole al dueño la tediosa tarea de que nos de toda la información de pepito, sin contar también que disminuye el riesgo de un dato erróneo sobre el perro.



14. ¿Por qué son útiles los objetos en la programación web y qué tipos de datos pueden almacenar?

RTA:

La importancia de los objetos radica en que estos son una abstracción del mundo real, también que pueden almacenar y estructurar una gran cantidad de datos, los cuales van a estar encapsulados, bajo el nombre que posea ese objeto, lo cual permite facilitar su manipulación y gestionamiento.

Por otro lado, en JavaScript se utiliza para la interacción con el DOM y en JSON, nos permite crear una falsa API, simulando una base de datos, para poder generar datos de prueba para la simulación de peticiones.

Los objetos pueden almacenar datos de tipo numéricos, cadenas, booleanos, arreglos, funciones y otros objetos.

15. ¿Qué es un array en JavaScript y por qué son esenciales?

RTA:

Un array o vector es una estructura de datos que permite almacenar y organizar múltiples elementos de manera secuencial, pues es una colección ordenada de valores, donde cada uno de estos valores está organizado en una posición o índice.

Dichos arrays son esenciales, gracias a su almacenamiento secuencial, la facilidad de acceder a dichos elementos a través de la posición donde se encuentra, también porque se pueden recorrer y conocer todos sus elementos gracias a que se pueden recorrer a través de los bucles.

Por otro lado, una de sus grandes ventajas es que son dinámicos y flexibles, puesto que pueden modificar su tamaño, también son versátiles pues pueden almacenar diferentes tipos de datos y que existen múltiples funciones las cuales permiten interactuar con los datos almacenados en ellos, es decir, los arrays permiten trabajar de una manera versátil y eficiente con los datos almacenados en ellos.

16. ¿Cómo acceder a un elemento dentro de un array? Explicar con un ejemplo.


RTA:

Para acceder a un elemento de un array se hace a través del nombre del array y entre corchetes ([]) el índice, el cual inicia en 0 y va hasta el número de elemento -1

Ejemplo:

```
const frutas = ["Pera", "Manzana", "Mango"];

//Acceder al mango posición 2
console.log("Fruta posición 2: ", frutas[2]);
```



```
Fruta posición 2: Mango  modulo1.js:29
```

Esto porque se sabe la posición del elemento al que se quiere acceder, pero si no se sabe o se quiere recorrer toda la información de array, se usa un ciclo o bucle, en este caso para el ejemplo usaremos un for, pero se puede de igual manera con los demás bucles también.

```
const frutas = ["Pera", "Manzana", "Mango"];

//Recorrer todo el vector
for (let index = 0; index < frutas.length; index++) {
  console.log("Fruta posición ", index, ": ", frutas[index]);
}
```



```
Fruta posición 0 : Pera  modulo1.js:30
Fruta posición 1 :      modulo1.js:30
Manzana
Fruta posición 2 : Mango modulo1.js:30
```


17. Mencionar al menos tres funciones de arrays y describir su utilidad en la programación web.

RTA:

1. `.indexOf()` → permite encontrar el número del índice de un elemento dentro del array, para esto hay que pasarle como parámetro el nombre del elemento y el nos retorna su índice

```
const frutas = ["Pera", "Manzana", "Mango"];

//indexOf
let position = frutas.indexOf("Pera");
console.log("indexOf, la pera esta en la posición ", position);
```



```
indexOf, la pera esta en la posición 0  modulo1.js:30
```

2. `.slice()` → permite copiar todos los elementos de un array a otro, a esta función no es necesario enviarle parámetros

```
const frutas = ["Pera", "Manzana", "Mango"];

//slice
let copiaFRutas = frutas.slice();

console.log('array original: ', frutas);
console.log('Vector copiado: ', copiaFRutas);
```



```
array original:      modulo1.js:31
  (3) ['Pera', 'Manzana', 'Mango']
Vector copiado:      modulo1.js:32
  (3) ['Pera', 'Manzana', 'Mango']
```

3. `.shift()` → elimina el primer elemento de un array, esta función no requiere parámetros

```
const frutas = ["Pera", "Manzana", "Mango"];
console.log('array original: ', frutas);
//.shift()
frutas.shift();
console.log('Vector eliminado el 1er elemento: ',frutas);
```

```
array original:      modulo1.js:27
  ▶ (3) ['Pera', 'Manzana', 'Mango']
Vector eliminado el 1er elemento:  modulo1.js:30
  ▶ (2) ['Manzana', 'Mango']
```

18. Proporcionar un ejemplo de cómo se utiliza una función de array para transformar y filtrar datos en un array.

RTA:

`.map()` es la función que nos permite transformar los elementos de un array y `.filter()` es la función que nos permite filtrar dichos elementos según no deseemos.

`.map()` → transformamos cada elemento del array multiplicándolo *2

```
const numero = [50,15,2,4,7,56,95,9,14];
//.map
console.log('array original: ', numero );
const suma = numero.map( x => x *2);
console.log('array con .map que multiplicamos cada elemento *2', suma);
```

```
array original:      modulo1.js:28
  ▶ (9) [50, 15, 2, 4, 7, 56, 95, 9, 14]
array con .map que multiplicamos cada elemento *2 modulo1.js:30
  ▶ (9) [100, 30, 4, 8, 14, 112, 190, 18, 28]
```

`.filter` → filtramos los elementos del array solo imprimiendo los mayores a 10

```
const numero = [50,15,2,4,7,56,95,9,14];
//.filter
console.log('array original: ', numero );
const suma = numero.filter( x => x > 10);
console.log('array con .filter numeros mayores a 10', suma);
```

```
array original:      modulo1.js:28
  ▶ (9) [50, 15, 2, 4, 7, 56, 95, 9, 14]
array con .filter numeros mayores a 10 modulo1.js:30
  ▶ (5) [50, 15, 56, 95, 14]
```

MÓDULO SOBRE HTML, CSS Y RESPONSIVE DESIGN

1. ¿Qué significa HTML y cuál es su función en el desarrollo web?

RTA:

HTML es una abreviación de sus siglas en inglés, las cuales significan HyperText Markup Language o en español lenguaje demarcado de hipertexto y su funcionalidad es la de brindarles el maquetado o esqueleto de una pagina web, gracias a sus etiquetas, las cuales nos permiten generar una óptima estructuración del código y por ende de la visualización en la pantalla del mismo, este es como el esqueleto, simple, pero de gran valor y utilidad para el desarrollo web

2. ¿Cuál es la estructura básica de un documento HTML? Describir las etiquetas esenciales.

RTA:

La estructura básica de un documento HTML son las etiquetas mínimas que debe contener dicho documento para poder trabajar de manera adecuada, entre ella resaltan:

- `<!DOCTYPE html>` → Esta etiqueta es la primer que encontramos y es la que nos define cual es la versión de HTML que se esta trabajando, para este ejemplo y la mas reciente es la versión 5, esta etiqueta solo es de apertura, ya q no tiene cierre.
- `<html></html>` → Todo documento HTML debe estar contenido dentro de estas etiquetas, ya que es lo que le indica al navegador que dicho contenido es código HTML y el navegador sabrá como procesarlo, cabe aclarar que uno de los atributos es la configuración del idioma.
- `<head></head>` → Dentro de esta etiqueta van a ir metadatos o información que no van a ser visibles dentro del cuerpo de la página web, pero si etiquetas que permiten la configuración de la misma y su título.
- `<body></body>` → Contiene todo el contenido visible de la página web, como texto, imágenes, párrafos, etc.

3. ¿Qué es CSS y cuál es su propósito en el desarrollo web?

RTA:

CSS es una abreviación por sus siglas en ingles que significa Cascading Style Sheets o hojas de estilo en cascada, es si es un lenguaje que nos permite aplicar estilos a una pagina web, en pocas palabras es el maquillaje que le damos al HTML para que se mire, bonito y atractivo para los clientes finales.

Su propósito es visualmente presentar de una forma mas atractiva el maquetado HTML y separar los estilos del HTML, ya que así podremos tener mejor control y flexibilidad en el manejo de nuestros proyectos web.

4. ¿Qué son selectores CSS, cuáles son los principales tipos de selectores y porqué es importante entender la especificidad en el contexto de las hojas de estilo en cascada (CSS)? Describir al menos tres tipos de selectores CSS y cómo la especificidad afecta a la aplicación de estilos en un proyecto de desarrollo web Frontend. Proporcionar ejemplos de situaciones en las que se utiliza la especificidad de selectores para resolver conflictos de estilos.

RTA:

Los selectores son los que nos permiten indicarles a los estilos (CSS) sobre que bloque de código o etiqueta se aplicaran dichos estilos, los tipos de selectores son de etiqueta o tipo, de clase o de id, estos son importantes, pues nos permiten tener una mejor organización del código y un mejor control del aplique de los estilos.

Por otro lado, la especificidad es importante pues esta es la que determina que estilo se aplica o no, en caso de que exista un conflicto, como por ejemplo si varios estilos afectan un mismo bloque de código.

- Selector de etiqueta o tipo: este selector aplica sobre una o varias etiquetas HTML, para ello en el archivo css solamente tienes que indicar cual es la etiquetas y automáticamente dicho estilo aplicara sobre la o las etiquetas que cumplan dicha condición
- Selector de Id: para poder aplicar este selector debe primero colocarle al bloque de código que quiera intervenir un id="nombreId" y en css colocas el nombre que colocaste en HTML antecedido por un numeral (#) y así se aplican los estilos sobre ese fragmento de código, cabe aclarar que el id debe ser único e irrepetible.
- Selector de clase: este es muy similar al anterior solo que en HTML en ves de id debes poner class="nombreClass" y de igual manera en css el nombre de la clase, pero antecedido por punto (.) y el o los estilos depositados hay aplicaran sobre dicho fragmento de código, cabe resaltar que las clases pueden ser repetirse en varios fragmentos de código.

La especificidad actúa en especial cuando hay anidación en el maquetado, pues entre más una etiqueta está contenida en otra, mayor deberá de ser su especificidad para poder llegar a ella.

en especificidad mayor peso tiene el id, seguid la clase y por último la etiqueta, claro está sin contar los estilos en línea pues aun que no es un selector es que iría de primera si se tuviera en consideración.

5. Explicar las diferencias entre los estilos en línea (inline), estilos internos (embedded) y estilos externos (external) en CSS. Indicar cuál de los tres estilos es el recomendado usar y por qué.

RTA:

- Los estilos en línea se ubican en el HTML directamente en la etiqueta, estos son los menos recomendados pues son los que menos permiten modificaciones, mezcla código HTML y CSS y peor enredan el código y son difíciles de mantener.
- Los estilos internos se ubican en el archivo HTML dentro de la etiqueta <style></style>, generalmente en el encabezado del mismo, no son recomendables pues su mantenimiento en proyectos extensos puede ser muy complicado, no proporciona reutilización del código y mezcla un poco el código HTML y CSS, formando un código espagueti.
- Los estilos externos, son aquellos que se encuentran ubicados en el archivo CSS y se enlazan con el documento HTML a través de la etiqueta link, estos son los más recomendados utilizar, pues desvinculan el código HTML, optimizando el código, permiten hacer reutilización de estilos y facilitan u mantenimiento en el tiempo, en especial si son códigos extensos.

6. ¿Qué es flexbox y cómo se utiliza para el diseño de páginas web?

RTA:

Flexbox es lo que nos permite distribuir los elementos de una sección de código de nuestra la página web en el espacio, dirección, alineación, etc.

Este modelo de estilos se usa gracias a ciertos atributos que dicho modelo nos brinda y nos permite crear diseños más flexibles y adaptativos, lo que genera armonía, optimización y amigabilidad de nuestra página.

Se usa aplicando la propiedad display al contenedor principal para el cual necesitamos la flexibilidad de trabajar en el espacio con sus elementos internos, cabe aclarar que la

propiedad display contiene diferentes opciones, las cuales se aplican según la necesidad del usuario, pero la más popular y la que nos brinda mayor libertad a la hora de trabajar es display: flex, obviamente flexbox nos brinda más propiedades, para alineación, medición, etc. Pero ya depende del usuario y lo que desea realizar en la página.

7. Explicar cómo se emplean las propiedades flexbox y explicar la función de las principales propiedades en la creación de diseños flexibles.

RTA:

Flexbox, nos brinda varias opciones para que la organización de los diferentes elementos en el espacio sea más fácil, dichas opciones son:

- Display: si se le aplica display: flex o display: inline-flex, convierte al contenedor flexible, lo cual nos da mayor libertad para organizar los elementos internos en el de una manera más fácil, cabe aclarar que, aunque esta propiedad tenga más opciones, las demás no nos permiten tener la libertad de organizar los elementos de la forma en que deseamos.
- Flex-direction: nos permite establecer la organización de los elementos internos en el contenedor al cual le estamos aplicando dicha propiedad, como por ejemplo si van a estar en filas, columnas, etc. esto depende de la necesidad del usuario.
- Justify-content: nos permite la alineación horizontal de los elementos internos en el contenedor al cual le aplicamos dicha propiedad
- Align-items: nos permite la alineación vertical de los elementos internos en el contenedor al cual le aplicamos dicha propiedad

NOTA: si el flex-direction es column, tanto el justify-content como el align-items cambian de sentido entre ellos, también cabe aclarar, que existen más propiedades, pero para mí estas son las más importantes y recurrentes.

8. ¿Qué es CSS Grid Layout y en qué se diferencia de flexbox?

RTA:

Al igual que flexbox Grid Layout es un método de diseño de css, que permite facilitar la creación de diseños, pero la diferencia que este tiene es que se enfoca en proporcionar un diseño en dos dimensiones no como el flexbox que es en una y además organiza los elementos en filas y columnas a la vez, como por ejemplo en una matriz o una tabla.

9. Proporcionar un ejemplo de cómo crear una cuadrícula sencilla con CSS Grid.

RTA:

Vamos a organizar una cuadrícula para mostrar por cada fila 2 imágenes, para eso usamos el modelo de Grid Layout.

```
.hijo-info{
  font-size: 80%;
  width: 100%;
  display: grid;
  grid-template-columns: repeat(2,1fr);
  padding: 0 0 0.5rem 0;
}
```



10. ¿Qué significa el diseño responsivo en el contexto del desarrollo web?

RTA:

Diseño responsivo significa, desarrollar una página web, pensando en los diferentes dispositivos desde los cuales un usuario puede acceder a ella, puesto que puede acceder desde un computador, un celular, una Tablet, etc. y cada uno de ellos tienen un tamaño diferente de pantalla, entonces al diseñar responsivamente estamos distribuyendo los elementos acorde a las diferentes pantallas, pues lo ideal es que todo tenga una adecuada visualización, dado a que si mira bien en una pantalla de un computador, no significa que en la de un celular ocurra lo mismo ya que puede verse muy pequeño y estrecho, entonces aunque el contenido sea el mismo, la distribución se maneja pensando en la comodidad del cliente y amigabilidad de la página, para hacerla más atractiva, óptima y funcional.

11. Enumerar al menos tres técnicas o estrategias utilizadas para lograr el diseño responsivo en una página web.

RTA:

1. Media queries: estas permiten cambiar los estilos CSS según los diferentes tamaños de pantalla que nosotros le especifiquemos, esto es muy útil porque podemos declarar varias medias queries y cada una de ellas es independiente, siempre y cuando las medidas de las pantallas sean diferentes, entonces estas adaptarán a los elementos internos en ella a cada una de las medidas especificadas por nosotros.
2. Flex Grids y Flexbox: estos modelos de diseños no permiten la facilidad que los elementos contenidos a ellos de adapten al espacio según el tamaño de la pantalla en donde se renderice nuestra página
3. Elementos multimedia adaptativos: estos nos permiten que los elementos multimedia se adapten al tamaño de su contenedor padre, lo bueno es que dependiendo a la postura que tenga su padre estos se adaptarán más fácilmente, se hace con tamaños de porcentajes, los cuales se aplicarán en relación al contenedor padre

MÓDULO SOBRE DOM E INTERACCIÓN CON EL DOM

1. ¿Qué es el DOM (Modelo de Objeto de Documento) en el contexto de la programación web?

RTA:

DOM es el árbol jerárquico de etiquetas y elementos que componen una página web, también nos permite acceder y manipular dinámicamente el contenido, la estructuración del contenido de dicha página.

Lo interesante de JS, es que nosotros podemos modificar una página sin ningún tipo de problema y podemos seleccionar los diferentes elementos que tenemos en la página y manipularlos como deseamos.

2. ¿Cuál es la diferencia entre el DOM y el HTML en una página web?

RTA:

La diferencia es que HTML es estático ya que una vez generado el maquetado inicial dichas etiquetas no cambian en el tiempo, en cambio DOM permite el dinamismo y manipulación de los elementos de una página según sea necesario o requerido.

3. ¿Por qué es importante entender y manipular el DOM en el desarrollo web Frontend?

RTA:

Principalmente por que nos permite a nosotros como desarrolladores, poder generar páginas web más interactiva, dinámicas y atractivas, en conclusión, podemos generar una mejor experiencia para los usuarios finales.

También, como se dijo antes el poder manipular los elementos y contenidos según nuestra necesidad facilita el desarrollo, sin dejar a un lado, que nos permite generar eventos, actualizaciones, creaciones, modificaciones, y otras acciones, según lo creamos pertinente, con el objetivo que ser lo más amigable y eficaz posible con los clientes.

4. ¿Qué son los eventos del DOM y cuál es su función en una página web?

RTA:

Los eventos son acciones que se detonan a raíz de un suceso ocurrido dentro la página web, dichas detonaciones pueden ser provocadas por los usuarios al realizar un evento en la página, como por ejemplo dar click en determinado botón o al presionar una tecla, también pueden ser detonados por el navegador al iniciar la página, o como resultado de otras interacciones con dicha página.

Como tal su función es la de permitir la interactividad del usuario con la página web y brindarle a la misma la capacidad de respuesta a determinados eventos que pueden ocurrir dentro de la misma.

5. Proporcionar ejemplos de eventos prácticos y comunes, como “click”, “submit” y “load o DOMContentLoaded”.

RTA:

- Para este primer ejemplo del evento click, simplemente va a aparecer en consola un mensaje de “le di click n veces”, al presionar un botón y ese n ira aumentando según la cantidad de veces que el usuario presioné el botón:

The first screenshot shows the HTML code in a file named `index.html`. It includes a button with the ID `myButton` and the text `ENVIAR`. The second screenshot shows the JavaScript code in a file named `modulo1.js`. It uses `document.getElementById('myButton')` to get the button and `addEventListener('click', ...)` to attach a click event listener that logs a message to the console. The third screenshot shows the browser's developer console with the output of the event listener, displaying messages like "le di click 0 veces" through "le di click 7 veces".

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Taller de nivelación</title>
8 </head>
9 <body>
10  <header>
11    <h1>MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT</h1>
12  </header>
13  <main>
14    <button id="myButton">ENVIAR</button>
15  </main>
16  <script src="modulo1.js"></script>
17 </body>
18 </html>
```

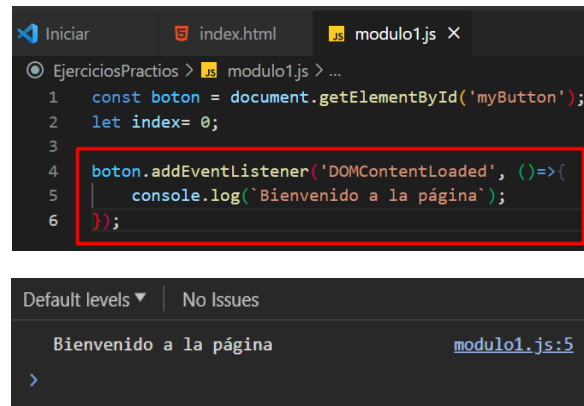
```
1 const boton = document.getElementById('myButton');
2 let index= 0;
3
4 boton.addEventListener('click', ()=>{
5   console.log(`le di click ${index++} veces`);
6 });
```

MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT

ENVIAR

le di click 0 veces
le di click 1 veces
le di click 2 veces
le di click 3 veces
le di click 4 veces
le di click 5 veces
le di click 6 veces
le di click 7 veces

- Para el evento DOMContentLoaded, se mostrará un mensaje por consola de “Bienvenido a la página” apenas inicia el programa:

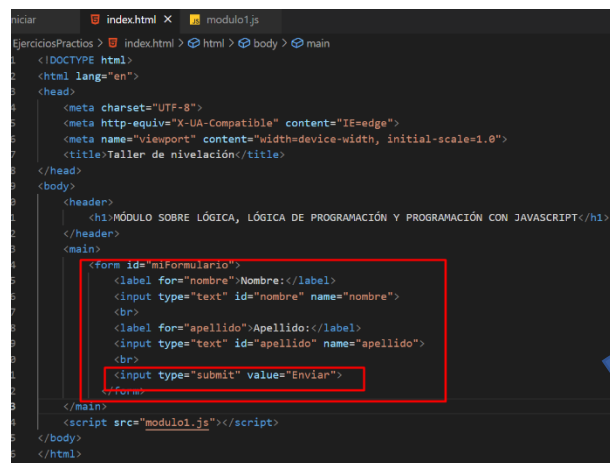


```
1 const boton = document.getElementById('myButton');
2 let index= 0;
3
4 boton.addEventListener('DOMContentLoaded', ()=>{
5   console.log('Bienvenido a la página');
6 });
```

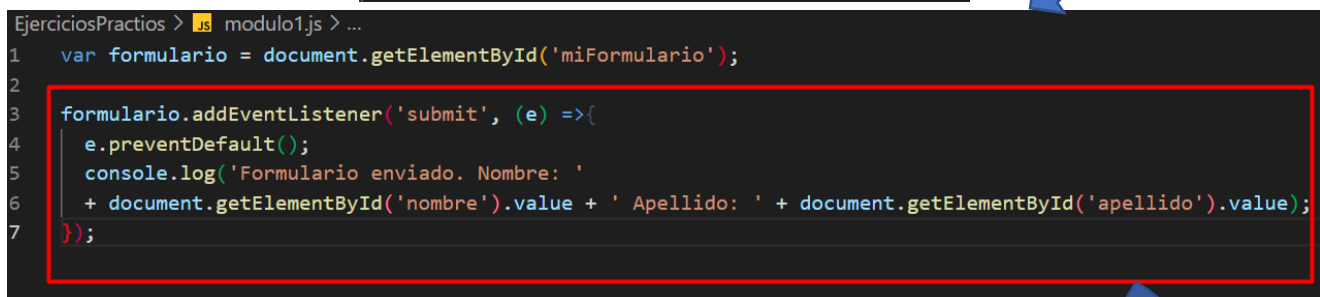
Default levels ▾ | No Issues

Bienvenido a la página [modulo1.js:5](#)

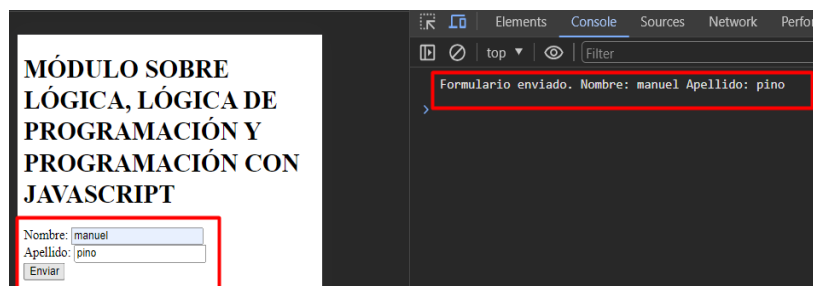
- Para el evento submit, se llenará un pequeño formulario con nombre y apellido y cuando se presione el botón de enviar se mostrará por consola la información digitada por el usuario:



```
<DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Taller de nivelación</title>
</head>
<body>
  <header>
    <h1>MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT</h1>
  </header>
  <main>
    <form id="miFormulario">
      <label for="nombre">Nombre:</label>
      <input type="text" id="nombre" name="nombre">
      <br>
      <label for="apellido">Apellido:</label>
      <input type="text" id="apellido" name="apellido">
      <br>
      <input type="submit" value="Enviar">
    </form>
  </main>
  <script src="modulo1.js"></script>
</body>
</html>
```



```
1 var formulario = document.getElementById('miFormulario');
2
3 formulario.addEventListener('submit', (e) =>{
4   e.preventDefault();
5   console.log('Formulario enviado. Nombre: '
6   + document.getElementById('nombre').value + ' Apellido: ' + document.getElementById('apellido').value);
7 });
```



MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT

Nombre:

Apellido:

Formulario enviado. Nombre: manuel Apellido: pino

6. ¿Por qué es importante manejar eventos en la interacción usuario-web y cómo se añaden controladores de eventos a los elementos del DOM?

RTA:

Es esencial porque permite generar páginas web más interactivas, amigables y con capacidad de respuesta hacia los usuarios, pues facilita la vida de los programadores y mejor la experiencia de los usuarios finales.

Por otro lado, los controladores de los eventos se añaden a través de métodos de JavaScript, como lo es el `addEventListener`, el cual permite asociar la función a eventos específicos detonados en la página web, obviamente este controlador va acompañado del tipo de evento que se desea trabajar y así se puede colocar múltiples controladores con múltiples tipos de eventos para generar una mejor interactividad de la página con el o los usuarios.

7. Describir al menos tres métodos para seleccionar elementos del DOM en JavaScript.

RTA:

1. `getElementById` → permite seleccionar los elementos del maquetado que posean un id
Sintaxis: `let variable = document.getElementById('id_elemento_del_maquetado');`
2. `getElementsByClassName` → permite seleccionar los elementos del maquetado que posean una clase
Sintaxis:
`variable = document.getElementsByClassName ('nombre_clase_elemento');`
3. `getElementsByTagName` → permite seleccionar las etiquetas del maquetado.
Sintaxis: `let variable = document.getElementsByTagName ('etiqueta');`

8. ¿Cómo se crea un nuevo elemento HTML y se agrega al DOM utilizando JavaScript?

RTA:

Se usa el método `.createElement` y especificamos que elemento deseamos crear, por ejemplo, un `div`

```
Let elementoAgregar = document.createElement('div');
```

Luego se debe agregar este nuevo elemento al DOM, pero debemos saber a qué parte del maquetado se agrega, por ejemplo, dentro de un contenedor

```
Let variableContenedor= document.getElementById('contenedorExistente');  
variableContenedor.appendChild(elementoAgregar);
```

ya el orden o demás configuraciones ya es opcionales, pero básicamente esta es la estructura mínima que se debe cumplir para poder agregar un nuevo elemento al DOM.

9. ¿Cuál es la importancia de la manipulación del DOM en la creación de aplicaciones web dinámicas e interactivas?

RTA:

A mi criterio, es importante porque le brindamos al usuario de la página una mejor experiencia y por parte del desarrollador permite poder manipular el contenido, la estructuración, el estilo o la capacidad de respuesta de la página según sea pertinente, entonces facilita generar estos desarrollos, tanto a nivel de tiempo como de esfuerzo.

10. Explicar brevemente los conceptos “event bubbling” y “event delegation” en el contexto de eventos del DOM.

RTA:

- event bubbling: es un comportamiento en el cual un evento que fue disparado por un elemento secundario, se propaga al inicio a través de la jerarquía de los elementos padres del DOM, esto afectará a cada uno de los elementos por los cuales tenga que pasar.
- event delegation: Es una técnica que utiliza al event bubbling, para poder manejar eventos de una mejor manera, especialmente cuando ocurren múltiples eventos en una misma página, puesto que, en lugar de asignar un controlador de eventos a cada elemento, se asigna un único controlador y se genera el event bubbling, pues recordemos que este afecta a todos sus padres o contenedores que estén en su camino hasta llegar al inicio detectando así eventos secundarios.

11. ¿Por qué son relevantes los conceptos “event bubbling” y “event delegation” en la gestión de eventos en páginas web con múltiples elementos interactivos?

RTA:

Son muy relevantes porque al comprenderlos y poder trabajar con ellos nos facilitan diferentes aspectos en la programación sobre todo a nivel de costos, tamaño, tiempo y esfuerzo, ya que, como se dijo anteriormente reduce la cantidad de controladores de eventos en una página, haciendo más eficientes las páginas, volviéndolas más eficientes, más simples de mantener y, por ende, mejorando el rendimiento de todo el proyecto software.

MÓDULO SOBRE COMUNICACIÓN CON EL SERVIDOR (STORAGE, PROMESAS, ASINCRONÍAS Y PETICIONES HTTPS)

Preguntas teóricas

1. Definir brevemente el concepto de localStorage y sessionStorage.

RTA:

- localStorage: este nos permite almacenar información persistente en el navegador, es muy útil pues estos datos están disponibles incluso si se cierra y se vuelve a abrir la aplicación, cabe resaltar que el guardado de esta información es de tipo clave valor, es decir que los datos que deseamos almacenar en el localStorage siempre estarán estructurados por un nombre que los identifique y su valor correspondiente, esta información no tiene fecha de caducidad y si se desea eliminar esta información se deberá hacer a nivel de código gracias a los diferentes métodos que localStorage nos permite trabajar.
- sessionStorage: es similar a localStorage, pero con la diferencia de que los datos almacenados en éste persisten solamente mientras la página web esté corriendo, en pocas palabras es un almacenamiento de datos temporal, el cual va a depender del tiempo en que se tenga la sesión en el navegador iniciada, una vez cerrado el navegador se perderán dichos datos, de igual manera trabaja bajo el concepto de clave valor del localStorage.

2. Describir las diferencias claves entre localStorage y sessionStorage.

RTA:

Para mí la principal es la persistencia de los datos, puesto que en localStorage no tienen fecha de caducidad, en cambio en el sessionStorage, los datos persisten mientras el navegador tenga una sesión activa, pero en el momento en que se cierre la pestaña del navegador los datos también dejarán de existir.

También es que en el localStorage los datos son accesibles desde cualquier pestaña del navegador, en cambio en el sessionStorage solo son accesibles desde la pestaña en que está abierto el programa.

Y una de las más decisivas es que localStorage tiene una capacidad de almacenamiento mayor a la que posee sessionStorage.

3. ¿Por qué son útiles para almacenar datos en el navegador y cuál es su límite de capacidad?

RTA:

Son útiles porque permiten la persistencia de datos en local lo cual nos ayuda a pasar información de página a página, nos permite hacer pruebas, no permite ver la ejecución en tiempo real de los programas, pues evitan muchas veces la necesidad de hacer peticiones externas como por ejemplo al servidor para evitar tener que recuperar muchas veces los mismos datos, también permite tener una especie de memoria para recordar las preferencias de los usuarios y reduce la carga al servidor lo que ayuda considerablemente a mejorar el rendimiento.

Capacidad:

localStorage: el límite típico es de al menos 5MB, pero esto puede variar según el navegador y configuración del usuario, pues en algunos navegadores puede aumentar.

sessionStorage: También el límite típico es de 5MB, pero este también puede variar según el navegador.

4. ¿Qué son las promesas en JavaScript y para qué se utilizan en el desarrollo web?

RTA:

Las promesas se utilizan para manejar las peticiones asíncronas, estas nos permiten hacer tareas a síncronas y manejar el resultado que estas nos dan a nuestro parecer y de una manera más estructurada y sobre todo legible, pues gracias a sus estados (pendiente, cumplida o rechazada), podemos realizar determinada acción según el resultado obtenido, pero al ser asíncrona no detiene el flujo del programa, lo cual facilita a la tolerancia a fallos del mismo.

5. Explica el concepto de asincronía en programación y cómo las promesas ayudan a manejar operaciones asíncronas.

RTA:

Es la manera no bloqueante de ejecutar tareas, ya que las tareas no se hacen de manera secuencial, sino que ellas se ejecutan todas y gracias al asincronismo los resultados de estas van llegando poco a poco, pero sin obstruir el flujo del programa, pues muchas de estas respuestas por x o y motivo pueden ser demoradas y si el programa se espera hasta obtener respuesta el usuario se cansa de esperar y el sistema no sería óptimo para nada.

Las promesas ayudan a manejar operaciones asíncronas, simplificando la gestión de las misma, puesto que gracias a sus tres estados (pendiente, cumplida o rechazada), se puede manipular de x o y manera las respuestas que cada una de dichas tareas obtiene del servidor, esto sin contar que permiten también encadenar funciones de manera clara, como por ejemplo con `.then()` y manejar errores de una manera más optima y efectiva con `.catch()`, mejorando sin duda alguna la legibilidad del código, la estructuración del mismo y su optimización, en especial de las tareas que no son inmediatas.

6. Proporciona un ejemplo de cómo se utiliza una promesa para realizar una operación asíncronica, como una solicitud de red.

RTA:

```
useEffect(() => {  
  getMyBookingTravels().then((response) => {  
    const getTravel = getMytravel(response);  
    setMyTravel(getTravel);  
  });  
}, []);
```

```
export const getMyBookingTravels = async() =>{  
  try {  
    const {data} = await axios.get(endPoints.travels);  
    return data;  
  } catch (error) {  
    console.log(error);  
    return null;  
  }  
}
```

```
const URL_BASE= 'https://flightapp-miniback.onrender.com/';  
  
const endPoints={  
  travels:`${URL_BASE}travels`,  
  passagers:`${URL_BASE}passagers`,  
}  
  
export default endPoints;
```

Esta es una promesa que realiza x acción una vez la función `mygetBookingTravels` obtiene la respuesta del servidor, en este caso es una petición `get` de la información de los vuelos, almacenada en una API REST.

7. ¿Qué es JSON Server y cómo se utiliza en el desarrollo web?

RTA:

JSON Server es una herramienta que nos permite crear una falsa API o falsa base de datos, es muy útil para el desarrollo y prueba de la aplicación puesto que como no contamos con un servidor backend, estos nos ayudan a la simulación y comprobación para saber si la aplicación web se integra bien o no con las peticiones asíncronas y su recuperación de datos.

Se utiliza instalando el paquete de JSON Server, creando un archivo `.json` y agregando los diferentes endpoints y sus respectivos datos, los cuales son necesarios para que nuestra aplicación trabaje de la mejor manera.

8. ¿Por qué es útil simular una API REST falsa con JSON Server en el desarrollo frontend?

RTA:

Es útil por varias razones, en primer lugar, por que permite al desarrollador frontend hacer una programación independiente del backend y no tiene que esperar a que el backend le brinde x o y información para poder continuar con su trabajo, lo que facilita que el desarrollo sea más rápido, las pruebas del misma más autónomas pero eficientes, en pocas palabras nos permite hacer desarrollos mas flexibles, eficientes e independientes, pero veraces y fiables.

9. ¿Cuáles son las diferencias claves entre los métodos del objeto promesa `then()`?`catch()` y las palabras claves `async/await`?

RTA:

La diferencia entre `then()` y `catch()` es que `then()` se usa para manejar las promesas cuando la respuesta fue exitosa, pues este ejecuta x acciones después de que la promesa es resuelta con éxito por parte del servidor, en cambio `catch()` se usa para manejar las promesas cuando la respuesta es negativa o de fallo, sirve como manejo de errores en las respuestas de las promesas.

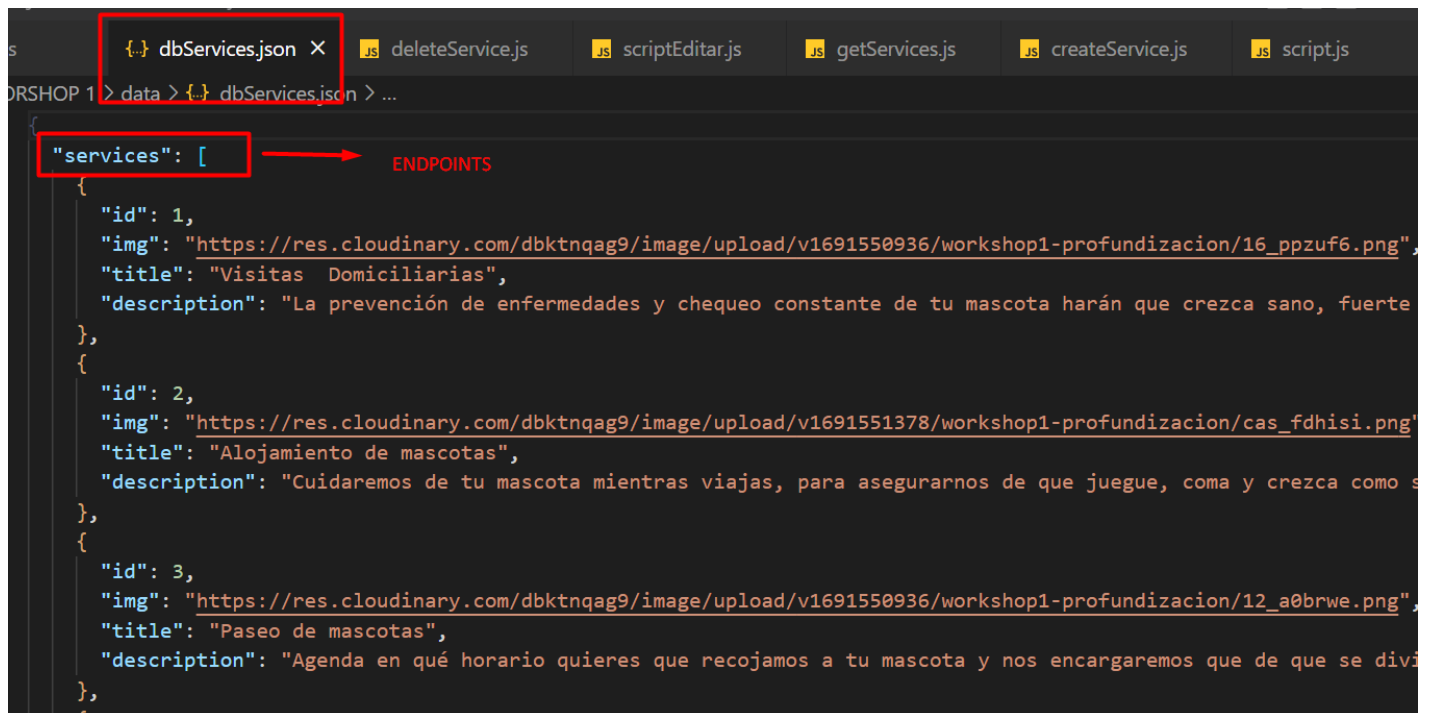
La diferencia entre `async` y `await` es que `async` se utiliza antes de la declaración de una función para indicarle al programa que esta es una función que devuelve una promesa y por ende permitirá el uso de `await` dentro de dicha función, en otras palabras, permite q la función contenga operaciones asíncronas y que dicha función de comporte como una promesa, en cambio para poder utilizar `await` se debe por obligación colocar dentro de una función `async` y hace que la ejecución de ese tramo de código donde este, espere atento la respuesta a dicha petición ya sea positiva o negativa.

10. Proporciona un ejemplo de cómo configurar una API falsa con JSON Server y realizar solicitudes (GET, POST, PUT y DELETE) a través de ella.

RTA:

Primero se debe instalar el paquete de JSON Server, con el comando:
`npm install -g json-server`

luego crear una estructura de carpetas (opcional) y un archivo `.json` el cual contendrá los endpoints necesarios por la aplicación y poner a escuchar cualquier cambio que tenga nuestra app con el comando `json-server --watch nombreDb.json`



```
{
  "services": [
    {
      "id": 1,
      "img": "https://res.cloudinary.com/dbktnqag9/image/upload/v1691550936/workshop1-profundizacion/16_ppzuf6.png",
      "title": "Visitas Domiciliarias",
      "description": "La prevención de enfermedades y chequeo constante de tu mascota harán que crezca sano, fuerte"
    },
    {
      "id": 2,
      "img": "https://res.cloudinary.com/dbktnqag9/image/upload/v1691551378/workshop1-profundizacion/cas_fdhisi.png",
      "title": "Alojamiento de mascotas",
      "description": "Cuidaremos de tu mascota mientras viajas, para asegurarnos de que juegue, coma y crezca como s"
    },
    {
      "id": 3,
      "img": "https://res.cloudinary.com/dbktnqag9/image/upload/v1691550936/workshop1-profundizacion/12_a0brwe.png",
      "title": "Paseo de mascotas",
      "description": "Agenda en qué horario quieres que recojamos a tu mascota y nos encargaremos que de que se divi"
    }
  ]
}
```

SOLICITUDES:

Get → Obtener información

```
export const getServices = async (URL) => {
  try {
    const RESPONSE = await axios.get(URL);
    return RESPONSE.data;
  } catch (error) {
    console.log('Error fetching services: ', error);
    return [];
  }
}
```

Post → Crear o agregar

```
export const createServices = async (URL, service) => {
  try {
    const RESPONSE = await axios.post(URL, service);
    return RESPONSE;
  } catch (error) {
    console.log('Error creating service: ', error);
    alert('Hubo un error al crear el nuevo servicio, por favor, inténtalo de nuevo.');
```

Patch → Actualizar

```
import axios from 'https://cdn.jsdelivr.net/npm/axios@1.3.5/+esm';

export async function deleteMessage(id, conversacion) {
  try {
    const response = await axios.patch('https://whatsapp-dz29.onrender.com/mensajes/${id}', {conversacion});
    console.log('response delete', response.data);
  } catch (error) {
    console.error('Error deleting message: ', error);
    alert('Hubo un error al eliminar el usuario. Por favor, inténtalo de nuevo.');
```

Delete → Eliminar

```
export const deleteService = async (URL, serviceID) => {
  try {
    await axios.delete(`${URL}${serviceID}`);
    return true;
  } catch (e) {
    console.log('Error, deleting service: ', e);
    alert('Hubo un error al eliminar el servicio, inténtalo de nuevo.');
```

Put → Actualizar

```
export const updateService = async (service) => {
  try {
    const RESPONSE = await axios.put('http://localhost:3000/services/${service.id}', {
      ...service
    });
    return RESPONSE;
  } catch (error) {
    console.error('Error, updating service: ', error);
    alert('Hubo un error al actualizar el producto. Por favor, inténtalo de nuevo.');
```

11. Describe las diferencias entre Fetch y Axios como métodos para realizar solicitudes HTTP en JavaScript.

RTA:

Fetch no hay que instalar en cambio axios si, también la sintaxis en fetch es mucho más extensa que en axios, el manejo de errores en ambos es diferente, en fetch es con ok y en axios es con catch, también axios tiene una gama más amplia de compatibilidad con navegadores incluso si son antiguos.

12. ¿Por qué es importante considerar las peticiones HTTP en aplicaciones web modernas?

RTA:

Primero porque las peticiones HTTP nos permite generar la comunicación cliente-servidor, la cual nos facilita la transferencia de datos entre el usuario y la aplicación.

También facilita el desarrollo de aplicaciones asíncronas y dinámicas, pues se pueden realizar actualizaciones dinámicas, directamente desde la interfaz del usuario, mejorando la experiencia del mismo.

Otra razón es por que nos permite la recuperación de datos de APIS externas, de una manera más fácil, en especial en el tipo de página SPA, pues los datos los recuperara solamente cuando es estrictamente necesario para el funcionamiento de la página.

Y por último mejora la escalabilidad de la aplicación, ya que a los múltiples usuarios querer obtener información de la página, el poder hacer peticiones asíncronas facilita mucho la interacción de todos ellos y sin dejar a un lado que permite la realización del CRUD, donde podemos gestionar de una mejor manera los datos del aplicativo.

13. Proporciona ejemplos de cómo se utilizan Fetch y Axios para realizar solicitudes GET y POST.

RTA:

GET

Fetch:

```
export const listProducts = async () =>{
  try{
    const response = await fetch('http://localhost:3000/productos');
    const productsData = await response.json();
    return productsData;
  }catch(e){
    console.log("Error",e)
  }
}
```

Axios:

```
/**Listar productos, metodo get */
export async function fetchProducts(){
  try{
    const response = await axios.get('http://localhost:3000/products/');
    return response.data;
  }catch(error){
    console.log('Error fetching products: ',error);
    return[];
  }
}
```

POST

Fetch:

```
export const createProduct = async (product) =>{
  try{
    const response = await fetch('http://localhost:3000/compras',{
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body:JSON.stringify(product)
    });
    const productos = await response.json();
    return productos
  }catch(e){
    console.log("Error",e)
  }
}
```


Axios:

```
export async function createProduct(product){
  try{
    const response = await axios.post('http://localhost:3000/products/', product);
    console.log('response create', response.data);
    return response
  }catch (error){
    console.log('Error creating product: ',error);
    alert('Hubo un error al crear el producto, Por favor, inténtalo de nuevo.')
  }
}
```

14. Explica la importancia del manejo de errores al trabajar con promesas en el desarrollo web.

RTA:

El manejo de errores al trabajar con promesas es muy importante, ya que estos nos permiten tener cierto control sobre la aplicación y la información allí depositada, puesto que, si no se manejan por x o y motivos podrían existir errores que no concebimos y si no se tratan, provocaría la presentación de información errónea, defectuosa o conllevaría a que la aplicación tome un rumbo no deseado.

En otras palabras, el manejar errores, nos permite brindar al usuario un índice mayor de fiabilidad e integridad en nuestra aplicación

15. Describe cómo se manejan los errores en las promesas, incluyendo el uso de catch.

RTA:

El manejo de los errores en las promesas se hace principalmente a través de 2 métodos, `.then()` y `.catch()`, puesto que el `.then()` nos sirve para expresarle al programa que hacer en caso de que la promesa tenga una respuesta positiva y el `.catch()` para expresarle al sistema que debe hacer en caso que no se pueda resolver dicha promesa, este se coloca al final de la cadena del `.then()` para capturar cualquier error que genere, se podría decir que `.then()` y `.catch()` son una especie de un si y un sino

16. ¿Cuáles son las diferencias clave entre los métodos del objeto promesa `.then().catch()` y la estructura `try/catch`?

RTA:

La principal diferencia es que, aunque todos son métodos para trabajar con errores, `.then()` y `.catch()` son para trabajar específicamente errores en operaciones o promesas asíncronas y `try/catch` son para trabajar errores en bloques de código síncronos.

17. Proporciona un ejemplo de cómo se puede manejar un error en una promesa al realizar una solicitud de red.

```
const ContainerRight = ({ id }) => {
  const [myTravel, setMyTravel] = useState([]);

  useEffect(() => {
    getMyBookingTravels().then(response => {
      const getTravel = getMyTravel(response);
      setMyTravel(getTravel);
    }).catch(error => {
      console.error('Error en la solicitud:', error.message);
    });
  }, []);
};
```