Solucion Taller de Nivelación

MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT

1. La logica de programacion se puede definir como la capacidad de abordar soluciones y generar algoritmos basados en la sintaxis de algún lenguaje de programación, por lo cual si lo llevamos al desarrollo frontend es una capacidad esencial en esta ámbito ya que no siempre se debe generar una solución desde cero sino entender y comprender lo que se debe modificar o ejecutar

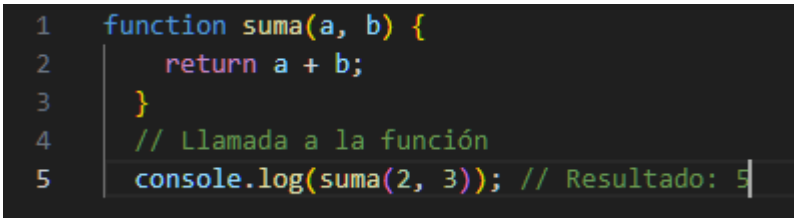
2. Un algoritmo es un conjunto de de instrucciones definido que permiten resolver un problema o una actividad esto a través de una sintaxis planteada previamente esto para que otras personas puedan comprender y seguir los pasos correctamente con tan solo conocer la sintaxis

3. Una estructura de control es un bloque de código el cual nos permite darle un orden o flujo a nuestra aplicación en distintas situaciones. por lo cual existen distintos tipos los cuales son :
    - Secuencial
    - Selección
        - If
        - If-else
        - if-elif-else"
    - Bucles
        - While
        - For
4. La declaración de variables y constantes en javaScript antes de **ES6**
   var miVariable = "Hola, mundo!";
   const PI = 3.1416;
   La declaración de variables y constantes en javaScript después de **ES6**
    - let miVariable = "Hola, mundo!";
    - const PI = 3.1416;

   Las mejoras que se obtuvo después de la implementación de **ES6** son:
    - Claridad y Mantenimiento del Código:Las declaraciones con let y const mejoran la claridad del código al proporcionar un ámbito de bloque más predecible y reducir el riesgo de errores relacionados con el hosting.
    - Prevención de Errores: La capacidad de declarar constantes con const ayuda a prevenir accidentalmente la reasignación de valores, mejorando la integridad de las variables que se espera que permanezcan constantes.
    - Mejora de la Modularidad:El ámbito de bloque facilita la creación de módulos más pequeños y autónomos, lo que mejora la modularidad y facilita el mantenimiento del código en proyectos grandes.

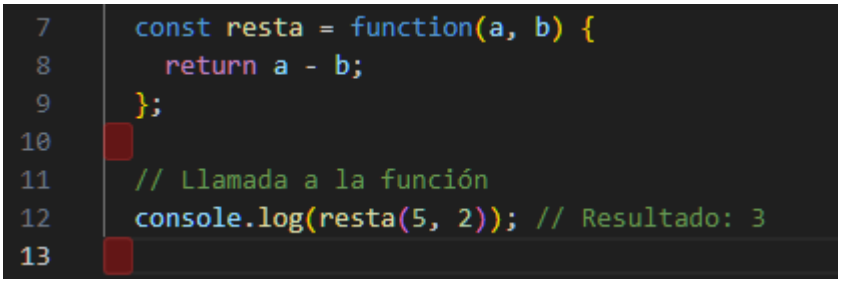
5. La declaración para las funciones, la expresiones de función y la funciones de flecha todas son distintas y tienen sus características particulares a continuación mencionaremos cada una de ellas:

   Declaración de funciones:



```
function suma(a, b) {
    return a + b;
}
// Llamada a la función
console.log(suma(2, 3)); // Resultado: 5
```

   Expresión de funciones:

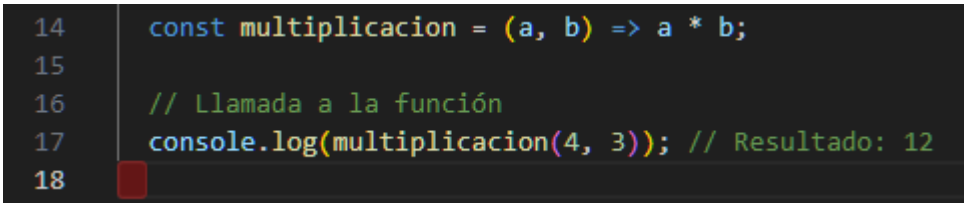


```
const resta = function(a, b) {
  return a - b;
};

// Llamada a la función
console.log(resta(5, 2)); // Resultado: 3

```

   Función de Flecha:



```
const multiplicacion = (a, b) => a * b;

// Llamada a la función
console.log(multiplicacion(4, 3)); // Resultado: 12

```

6. Las funciones son fundamentales para la programación en general por eso mencionaremos 3 razones para informar sobre su importancia:
   - **Reutilizacion de codigo**
   - **Conexión Con el backend**
   - **Modularidad y Mantenimiento**
7. **Argumentos:** Son los valores que se le pasan a una función cuando es invocada en alguna parte del código. dependiendo del orden de los argumentos se le asigna los valores a los parámetros dentro de la función

   Parámetros: Los parámetros son variables locales a la función; su ámbito está limitado a la propia función. y sus valores son dados por el orden en que envían los argumentos.
8. Es una función que se pasa como argumento a otra función y que es invocada en otro momento.Los callbacks son comúnmente utilizados para manejar operaciones asíncronas, como la lectura de archivos, solicitudes a servidores, eventos de temporizadores, entre otros. Permiten ejecutar código después de que una operación haya finalizado sin bloquear la ejecución del resto del programa.

```
function saludar(nombre) {
  alert("Hola " + nombre);
}

function procesarEntradaUsuario(callback) {
  var nombre = prompt("Por favor ingresa tu nombre.");
  callback(nombre);
}

procesarEntradaUsuario(saludar);
```

9. Los objetos en JavaScript son colecciones no ordenadas de propiedades, donde Cada propiedad tiene un nombre (clave) y un valor asociado. Estas propiedades pueden contener valores de cualquier tipo de datos, incluyendo números, cadenas, booleanos, funciones y otros objetos, lo que proporciona una gran flexibilidad.

```
const persona = {
  nombre: 'Juan',
  edad: 30,
  esEstudiante: false
};
```

10. Las propiedades de un objeto son como variables que pertenecen al objeto y se utilizan para almacenar información relacionada. Mientras que las propiedades son funciones que están asociadas a un objeto y pueden realizar acciones específicas o cálculos basados en los datos del objeto. Los métodos son propiedades especiales que contienen funciones.

11. **Notación de Punto:**

```
const persona = {
  nombre: 'Juan',
  edad: 25
};

console.log(persona.nombre);
```

Notación de Corchetes:

```
const persona = {
  'nombre completo': 'Maria García',
  'año de nacimiento': 1990
};

console.log(persona['nombre completo']);
```

Se recomienda la notación de punto para casos más simples y cuando el nombre de la propiedad es conocido, y la notación de corchetes cuando se trabaja con propiedades más dinámicas o con nombres no convencionales.

12. Si existe una forma para recorrer las propiedades de un objeto la cual consiste en

usar el bucle for … in el cual se usa para recorrer todas las propiedades de un de un objeto a continuación un ejemplo :

```
49  const persona = {
50      nombre: 'Juan',
51      edad: 30,
52      profesion: 'Ingeniero'
53    };
54
55    for (let propiedad in persona) {
56      console.log(propiedad + ': ' + persona[propiedad]);
57    }
```

Es útil recorrer las propiedades de un objeto cuando sus propiedades son generadas dinámicamente o en el caso de no conocer el nombre de todas sus propiedades

13. Los objetos en la programación web tiene una gran importancia y utilidad debido a la capacidad que tiene para organizar y estructurar datos de manera eficiente permitiéndonos representar cualquier entidad del mundo real tales como productos, usuarios,lugares etc.los tipos de datos que se pueden almacenar son: números,cadenas,booleanos,funciones ,otros objetos,arrays, nulos y undefined
14. Un array en javaScript es un tipo de dato el cual nos permite almacenar múltiples valores bajo un solo nombre, La importancia son importantes debido a su capacidad para representar colecciones de datos de una forma muy eficiente.
15. Para poder acceder a alguno de los elementos dentro de un array en javascript, se debe usar la notación de corchetes. la cual consiste en poner dos corchetes y en medio de este el índice del objeto al cual deseamos acceder.

```
1
2   // Array de nombres de personas
3   const nombres = ["Juan", "María", "Pedro", "Ana", "Luis"];
4
5   // Mostrar el nombre por consola
6   console.log(nombre[0]);
7
```

16. Funciones para los array en javascript:
    - **map:**crea un nuevo array aplicando una función a cada elemento del array original. El nuevo array contendrá los resultados de aplicar la función a cada elemento.
    - **filter:** crea un nuevo array con todos los elementos que pasan una prueba implementada por la función proporcionada.
    - **forEach**:ejecuta una función dada una vez por cada elemento del array.
        i. La utilidad de estas funciones en la programación web lo podemos ver en el momento de encontrar datos o recorrerlos, ya que generalmente obtendremos arrays con datos que no necesitaremos por lo cual debemos aplicar ciertos filtros.
17.

Filtrar Datos

```
const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Filtrar números pares
const numerosPares = numeros.filter(function(numero) {
  return numero % 2 === 0;
});
```

Transformar Datos

```
// Obtener los cuadrados de los números pares
const cuadradosDePares = numerosPares.map(function(numero) {
    return numero * numero;
  });

  console.log(cuadradosDePares); // [4, 16, 36, 64, 100]
```

MÓDULO SOBRE HTML, CSS Y RESPONSIVE DESIGN

1. HTML significa "HyperText Markup Language" (Lenguaje de Marcado de Hipertexto) en inglés. Es el lenguaje estándar utilizado para crear y diseñar páginas web. HTML proporciona una estructura básica para el contenido de una página web, permitiendo a los desarrolladores organizar y presentar información de manera semántica en la web.
2.

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi Página Web</title>
  </head>
  <body>
    <!-- Contenido de la página -->
  </body>
</html>

```

   - La etiqueta <html> envuelve todo el contenido del documento HTML.
   - La sección <head> contiene metadatos y enlaces a recursos externos.
   - La etiqueta <title> define el título de la página, que se muestra en la barra de título del navegador o en la pestaña.
   - La sección <body> contiene todo el contenido visible de la página web, como texto, imágenes, enlaces, y otros elementos HTML.
3. CSS, que significa "Cascading Style Sheets" (Hojas de Estilo en Cascada) en inglés, Es un lenguaje de estilo utilizado para describir la presentación y el formato de un documento escrito en HTML o XML. Su propósito principal en el desarrollo web es

separar la estructura del documento (definida por HTML) de su estilo y presentación, permitiendo así un mayor control sobre la apariencia visual de una página web.

4. Los selectores son patrones que se utilizan para seleccionar y aplicar estilos a elementos específicos en un documento HTML. existen distintos tipos de selectores los cuales son:
    - Selectores de Tipo
    - Selectores de Clase
    - Selectores de ID
    - Selectores de Atributo

    La especificidad es una jerarquía la cual nos indica la regla de estilo que tiene mayor prioridad cuando se presenta un conflicto.
5. **Estilos en línea:** Los estilos en línea se aplican directamente a un elemento específico en la etiqueta HTML a través del atributo **'style'**.
   Estilos Internos: Los estilos internos se definen dentro de la sección <style> en la misma página HTML.
   Estilos Externos: Los estilos externos se almacenan en archivos separados con extensión .css. Estos archivos se enlazan con la página HTML utilizando la etiqueta <link> dentro del elemento <head>.
   Para proyectos muy simples y que no se espera que crezcan en el tiempo es viable usar Estilos de línea o internos y generalmente se recomienda usar Estilos Externos para proyectos medianos o grandes. ya que estos nos facilitan el mantenimiento y la reutilización.
6. Es un modelo de diseño en CSS que proporciona un diseño más eficiente y predecible en contenedores y elementos hijos. Flexbox simplifica la alineación, distribución y reorganización de elementos en un contenedor, lo que facilita la creación de diseños responsivos y flexibles en páginas web.
7. **'Display: flex'**: con esta propiedad todos los elementos hijos de este contenedor son flexibles
   '**flex-direction**':
   '**justify-content**'
   '**align-items**'
8. CSS Grid Layout se centra en la creación de diseños bidimensionales, lo que significa que puedes trabajar tanto en columnas como en filas al mismo tiempo.
   La principal diferencia entre CSS Grid Layout con Flexbox es la orientación ya que la segunda se centra en diseños unidimensionales.
9.

```css
/* Estilo para el contenedor que utiliza Grid */
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* Tres columnas de igual tamaño */
    grid-template-rows: repeat(3, 100px); /* Tres filas de altura fija de 100px cada una */
    gap: 10px; /* Espacio entre las celdas de la cuadrícula */
}

/* Estilo para los elementos dentro de la cuadrícula */
.grid-item {
    background-color: #3498db;
    color: #fff;
    padding: 20px;
    text-align: center;
}
```

10. Se refiere a la práctica de crear sitios web de manera que se adapten y respondan de manera óptima a diversos dispositivos y tamaños de pantalla. El objetivo es proporcionar una experiencia de usuario consistente y agradable, independientemente de si el usuario accede al sitio desde un ordenador de escritorio, una tableta, un teléfono móvil u otros dispositivos con diferentes tamaños de pantalla.
11.
    - Media Queries
    - Flexbox/Grid Layout
    - Imágenes Flexibles y Etiqueta

MÓDULO SOBRE DOM E INTERACCIÓN CON EL DOM

1. En el contexto de la programación web, el DOM proporciona una interfaz que permite a los scripts o programas acceder, manipular y actualizar el contenido, estructura y estilo de un documento web.
2. El HTML es el lenguaje de marcado que define la estructura estática inicial de una página web, mientras que el DOM es una interfaz de programación que representa y permite la manipulación dinámica de esa estructura durante la interacción del usuario. Ambos son fundamentales para el desarrollo web moderno.
3. Es importante por varias razones ya que este interviene en interactividad y dinámica de una página web:
    - Interactividad del Usuario
    - Actualizaciones Dinámicas
    - Formularios y Validaciones

4. Los eventos del DOM (Modelo de Objeto de Documento) son interacciones o sucesos que ocurren en una página web, como clics del mouse, pulsaciones de teclas, cambios en formularios, entre otros. Estos eventos permiten que la página responda dinámicamente a las acciones del usuario o a cambios en el entorno, mejorando la interactividad y la experiencia del usuario. La función principal de los eventos en una página web es capturar y manejar estas interacciones.
5. **Evento "click"**

ejemplos.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo de Evento Click</title>
</head>
<body>
  <button id="boton">Haz clic</button>
<script src="ejemplo .js"> </script>
</body>
</html>
```

Welcome | ejemplos.html | JS ejemplo.js

JS ejemplo.js > ...

```javascript
var botonElemento = document.getElementById('boton');

// Asociar un evento de clic al botón
botonElemento.addEventListener('click', function() {
  alert('Botón clickeado');
});
```

Evento "submit":

ejemplos.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo de Evento Submit</title>
</head>
<body>

  <form id="miFormulario">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
    <input type="submit" value="Enviar">
  </form>
  <script src="ejemplo .js"></script>
</body>
</html>
```

```
JS ejemplo .js > ...
1   var formularioElemento = document.getElementById('miFormulario');
2
3   // Asociar un evento de envío de formulario
4   formularioElemento.addEventListener('submit', function(event) {
5       event.preventDefault(); // Evitar el envío del formulario por defecto
6       alert('Formulario enviado');
7   });
```

Evento "load" o "DOMContentLoaded":

```
ejemplos.html > ...
1   <!DOCTYPE html>
2   <html lang="es">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Ejemplo de Evento Load o DOMContentLoaded</title>
7   </head>
8   <body>
9
10    <p id="parrafo">Este párrafo se actualizará después de cargar la página.</p>
11
12    <script src="ejemplo .js"></script>
13  </body>
14  </html>
```

```
JS ejemplo .js > ...
1       // Asociar un evento de carga de contenido
2       document.addEventListener('DOMContentLoaded', function() {
3           var parrafoElemento = document.getElementById('parrafo');
4           parrafoElemento.innerHTML = 'La página se ha cargado completamente.';
5       });
```

6. Manejar eventos en la interacción usuario-web es esencial porque permite que las Las páginas web sean interactivas y respondan a las acciones del usuario de manera dinámica. Los eventos son la clave para crear experiencias de usuario fluidas y mejorar la usabilidad de una aplicación web.

7.

- Este método se utiliza para seleccionar un elemento por su identificador único (ID). Cada ID en el documento debe ser único, por lo que este método devuelve un solo elemento.
- Este método se utiliza para seleccionar elementos por su nombre de clase. Puede devolver una colección de elementos (objeto tipo NodeList) ya que múltiples elementos pueden tener la misma clase.
- Este método permite seleccionar un elemento utilizando un selector CSS. Devuelve el primer elemento que coincide con el selector.

```
var miElemento = document.getElementById('ID');
var elementosConClase = document.getElementsByClassName('Clase');
var primerParrafo = document.querySelector('p');

```

8.
    - Utilizamos el método **createElement** para crear un nuevo elemento con la etiqueta deseada.
    - Seleccionamos el elemento existente al cual deseamos agregar el nuevo elemento. Podemos utilizar métodos como getElementById, getElementsByClassName, getElementsByTagName, o cualquier otro método de selección.
    - Utilizamos el método appendChild para agregar el nuevo elemento al elemento padre seleccionado.
9. El DOM es una representación en memoria de la estructura de un documento HTML o XML, y proporciona una interfaz que permite a los desarrolladores acceder, modificar y manipular los elementos de la página web de manera dinámica mediante código JavaScript.
10. **Event Bubbling**: es un concepto en la manipulación de eventos en el DOM dónde, cuándo ocurre un evento en un elemento específico, dicho evento se propaga desde el elemento más interno hacia el elemento más externo de la jerarquía DOM. En otras palabras, el evento "burbujea" desde el objetivo del evento hasta el ancestro superior. Esto significa que si tienes elementos anidados dentro de otros en tu HTML y se produce un evento en el elemento más interno, se activarán los controladores de eventos de esos elementos en orden ascendente hacia el elemento más externo. **Event Delegation**: Es una técnica que aprovecha el burbujeo de eventos para manejar eventos en elementos secundarios a través de un único controlador de eventos en un elemento ancestro común. En lugar de asignar un controlador de eventos a cada elemento individual, puedes asignar un solo controlador al elemento contenedor y dejar que los eventos se propaguen (burbujean) desde los elementos hijos hasta el contenedor.
11. El "event bubbling" como la "event delegation" son técnicas que facilitan la gestión de eventos en páginas web con múltiples elementos interactivos, proporcionando eficiencia, flexibilidad y un código más limpio y mantenible. Estos conceptos son esenciales para construir aplicaciones web escalables y dinámicas.

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

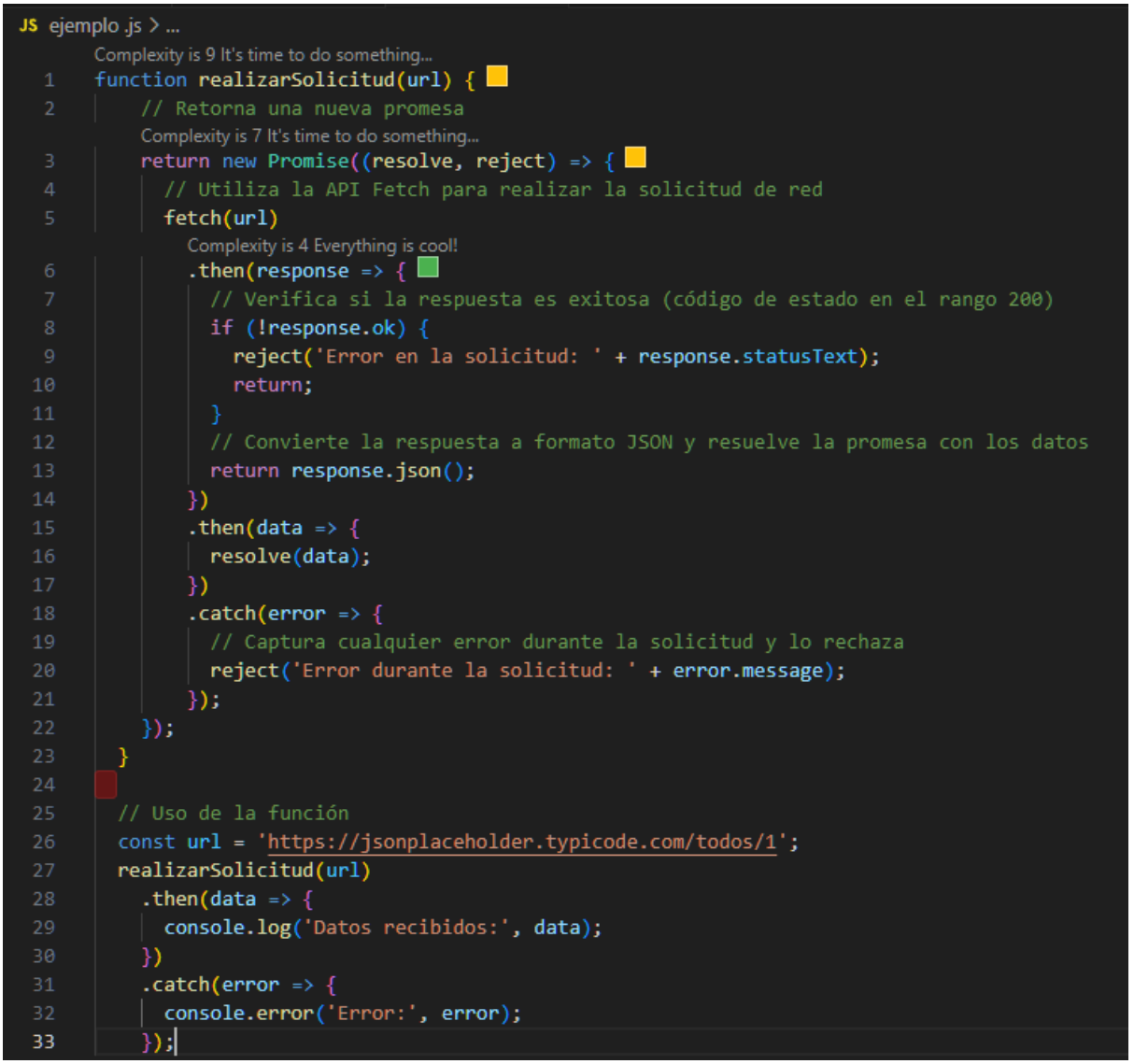
1. Son mecanismo de almacenamiento de datos proporcionados por los navegadores web los cuales permiten a los navegadores almacenar datos de manera permanente o temporal cabe resaltar que estos datos se guardan en el lado del cliente .
2.
    - Los datos almacenados en **local Storage** persisten incluso después de cerrar el navegador y se mantienen hasta que se eliminan explícitamente o el usuario borra los datos del navegador.

- Los datos almacenados en **sessionStorage** tienen una duración limitada a la duración de la sesión. Se pierden cuando se cierra la pestaña o ventana del navegador.

3. Es importante almacenar datos en el navegador para mejorar el rendimiento y reducción del tráfico de red ya que en muchas ocasiones se pueden reutilizar y al tener estados datos guardados podemos mejorar la experiencia del usuario.

   localStorage: El límite de capacidad de localStorage varía entre los navegadores, pero generalmente es de alrededor de 5 MB a 10 MB por dominio. Sin embargo, es importante tener en cuenta que este límite es una estimación y puede cambiar en futuras versiones de los navegadores.
   sessionStorage: El límite de capacidad de sessionStorage es similar al de localStorage, pero la principal diferencia es que los datos se eliminan cuando se cierra la pestaña o ventana del navegador. La capacidad también es de alrededor de 5 MB a 10 MB por dominio.

4. Las promesas en JavaScript son un objeto que representa el resultado eventual (éxito o fracaso) de una operación asíncrona. Proporcionan una forma más limpia y estructurada de trabajar con código asíncrono en comparación con los callbacks anidados.
5. La asincronía en programación se refiere a la ejecución de tareas de forma no bloqueante. En un entorno asíncrono, el programa puede continuar ejecutando otras tareas mientras espera que ciertas operaciones finalicen. Esto es fundamental en situaciones donde ciertas operaciones, como solicitudes de red, lectura de archivos o interacciones de usuario, pueden llevar tiempo y bloquear el hilo de ejecución si se realizan de manera síncrona.
   Las promesas son un patrón de diseño que permite gestionar operaciones asíncronas de manera más clara y estructurada.
6.



```
JS ejemplo.js > ...
Complexity is 9 It's time to do something...
function realizarSolicitud(url) {
    // Retorna una nueva promesa
    Complexity is 7 It's time to do something...
    return new Promise((resolve, reject) => {
        // Utiliza la API Fetch para realizar la solicitud de red
        fetch(url)
            Complexity is 4 Everything is cool!
            .then(response => {
                // Verifica si la respuesta es exitosa (código de estado en el rango 200)
                if (!response.ok) {
                    reject('Error en la solicitud: ' + response.statusText);
                    return;
                }
                // Convierte la respuesta a formato JSON y resuelve la promesa con los datos
                return response.json();
            })
            .then(data => {
                resolve(data);
            })
            .catch(error => {
                // Captura cualquier error durante la solicitud y lo rechaza
                reject('Error durante la solicitud: ' + error.message);
            });
    });
}

// Uso de la función
const url = 'https://jsonplaceholder.typicode.com/todos/1';
realizarSolicitud(url)
    .then(data => {
        console.log('Datos recibidos:', data);
    })
    .catch(error => {
        console.error('Error:', error);
    });
```

7. JSON Server es una herramienta que proporciona una API RESTful completa y funcional basada en un archivo JSON. Esta herramienta es útil durante el desarrollo web para simular una API real y permitir a los desarrolladores frontend interactuar con datos sin tener un servidor backend completamente funcional.
8. Es muy útil simular una API REST con JSON ya que este nos permite realizar un desarrollo de forma independiente sin depender del equipo backend además nos brinda una iteración mucho más rápida cualquier cambio puede probarse inmediatamente, además de permitirnos trabajar de forma local.
9. La elección entre .then().catch() y async/await a menudo se reduce a la preferencia del estilo de codificación y al contexto del proyecto. Ambos son válidos y tienen sus casos de uso. async/await es generalmente preferido por su sintaxis más clara y su capacidad para manejar errores de manera más estructurada, pero .then().catch() puede ser más apropiado en ciertos casos, especialmente si se prefiere un estilo de codificación más explícito o se trabaja con bibliotecas que devuelven Promesas.
10.

11. La elección entre Fetch y Axios depende del contexto y de las necesidades específicas del proyecto. Fetch es una opción viable para casos simples y está

disponible de forma nativa en muchos navegadores, pero Axios ofrece funcionalidades adicionales, manejo de errores simplificado y una sintaxis más intuitiva en algunos casos. La elección puede depender de factores como el tamaño del proyecto, la necesidad de características específicas y la preferencia personal del desarrollador.

12. Las peticiones HTTP son un componente central en el desarrollo de aplicaciones web modernas, permitiendo la comunicación eficiente, la obtención de recursos, la integración con servicios externos y la entrega de experiencias de usuario dinámicas y seguras. Considerar y entender el manejo de peticiones HTTP es esencial para los desarrolladores web.
13.
14. El manejo de errores al trabajar con promesas es esencial para construir aplicaciones web robustas, confiables y fáciles de mantener. Proporciona una manera estructurada de lidiar con situaciones inesperadas, mejora la experiencia del usuario y facilita el proceso de desarrollo y depuración.
15.
16. La elección entre **try/catch** y **then().catch()** depende del contexto y de si estás trabajando con operaciones síncronas o asíncronas. Para operaciones asíncronas y el manejo centralizado de errores en cadenas de promesas, **then()** y **catch()** son más apropiados. Para operaciones síncronas o para manejar errores en funciones asíncronas con **async/await**, puedes utilizar **try/catch**. En muchos casos, ambos enfoques se pueden combinar para manejar errores de manera efectiva.
17.

MÓDULO SOBRE REACT JS

1. React es una biblioteca de javascript que se utiliza para construir interfaces de usuario interactivas y eficientes. Sus principales características son los componentes,virtual DOM,JSX,estados,Reac Router.
2. Un componente es una unidad independiente y reutilizable de interfaz de usuario que puede contener su propio estado,propiedades y lógica. Estos son la base fundamental para construir aplicaciones en REACT.los componentes se dividen en **Componentes Funcionales:**Son funciones de JavaScript que aceptan propiedades (props) como argumentos y devuelven elementos React (normalmente JSX). No tienen estado interno ni métodos de ciclo de vida. Se recomienda utilizar componentes funcionales siempre que sea posible debido a su simplicidad y rendimiento.
   Componentes de Clases: Son clases de JavaScript que extienden la clase base React.Component. Pueden tener su propio estado interno, métodos de ciclo de vida y se utilizan para manejar lógica más compleja y manejar estados internos.

3. el Virtual DOM en ReactJS actúa como una capa intermedia que mejora la eficiencia

en la actualización del DOM real. Permite que React realice comparaciones eficientes y determine las actualizaciones necesarias antes de manipular el DOM real, contribuyendo a un rendimiento óptimo en aplicaciones web. Esta técnica es especialmente valiosa en aplicaciones con interfaces de usuario dinámicas y cambiantes.

4. JSX es una parte integral de ReactJS y contribuye significativamente a la facilidad de desarrollo y la legibilidad del código. Facilita la creación de interfaces de usuario, mejora la integración de lógica dinámica y ayuda a construir componentes reutilizables de manera más intuitiva. Aunque opcional, JSX se ha convertido en una práctica común y altamente recomendada al trabajar con React.
5. Los hooks en React simplifican la lógica y la gestión del estado en los componentes de función, eliminando la necesidad de convertirlos en componentes de clase para utilizar características avanzadas. Permiten un código más modular, reutilizable y fácil de entender. Su adopción ha cambiado la forma en que se estructuran y desarrollan las aplicaciones React.
6.
    - **useState:**Permite que los componentes de función tengan estado local. Este hook devuelve un array con dos elementos: el estado actual y una función para actualizar ese estado. Se utiliza para gestionar variables de estado en componentes de función.
    - **useEffect:**se utiliza para realizar efectos secundarios en componentes de función. Puede emular el ciclo de vida de los componentes de clase, como componentDidMount y componentDidUpdate. Se ejecuta después de que la renderización se ha completado y puede realizar operaciones asíncronas, suscripciones a eventos, etc.
    - **useContext:**se utiliza para suscribir un componente de función al contexto de React y acceder a los valores proporcionados por un Context.Provider superior. Permite a los componentes acceder a valores globales sin pasar propiedades a través de múltiples niveles de componentes.
7. El uso de Hooks en ReactJS está sujeto a ciertas reglas y convenciones que garantizan su correcto funcionamiento y evitan problemas comunes y las podemos definir como:
    - Solo en Componentes de Función
    - Solo en el Nivel Superior de los componentes
    - Solo llamarse desde componentes de react
8. React Router DOM es una librería que proporciona herramientas de enrutamiento para aplicaciones React de una sola página (Single Page Applications - SPAs). Permite la navegación y gestión de la URL en la aplicación, lo que facilita la creación de aplicaciones con múltiples vistas. y sus principales componentes son:
    - BrowserRouter
    - Route
    - Link
    - useParams
    - useHistory
9.
10.
11. Se ingresa a la carpeta donde queremos crear el proyecto ReactJS con Vite, abrimos el cmd de windows y colocamos el siguiente comando **npm create**

vite@latest <nombre-de-mi-proyecto> luego con las flechas del teclado seleccionamos la opción de vite y seguido de ellos la de javascript
y ya nuestro proyecto estará creado.

12. Primero debemos clonar el proyecto de forma remota en github una vez ya tengamos nuestro repositorio, accedemos a render donde nos logueamos con nuestra cuenta de gitHub y el nos permitira crear un servicio web. ahí seleccionamos el repositorio donde tenemos nuestro JSON y el lo desplegará y nos dará la ruta que podemos usar para acceder a ello.
13. Para desplegar un proyecto con github pages primero debemos subir nuestro repositorio y subir el proyecto, luego vamos a las configuraciones del repositorio y seleccionamos la opción pages donde nos permitira elegir la rama que queremos despegar una vez escojamos la de nuestro gusto volvemos a la página de nuestro repositorio y la esquina derecha seleccionamos el icono de tuerca y chuleamos la segunda opción..