**Section – 1**

1. **1-**What is the difference between controlled and uncontrolled components in React?

* **Ans -** **Controlled components**

React manages the component's state, including form data, and updates it through props. This allows developers to validate input as it's changed, and to have complete control over form input values. Controlled components are recommended for forms, and offer predictable data flow.

* **Uncontrolled components**

The component manages its state internally, directly interacting with the DOM. This means that form data is handled by the DOM itself, and validation is only performed after form submission. Uncontrolled components can be easier to integrate with non-React code, and may be slightly less code. However, they lack predictability, as form elements may change or be influenced by external sources.

**2**-What is the significance of the functional Component in React? Explain hooks methods like use Effect, use State and use Ref hooks?

**Ans-** Functional components are a type of component in React that are simple JavaScript functions that take props as input and return JSX elements. They are often used for presentational or stateless components.

React Hooks are functions that allow developers to use React features in functional components, such as state management, lifecycle methods, and side effects. Some common hooks include:

* **useState**: Allows you to have state variables in functional components. It provides two variables: the value of the state and the function used to update the state.
* **useEffect**: Helps manage tasks that aren't directly related to showing stuff on the screen, such as fetching data from the internet, retrieving data from API endpoints, and setting up timers.
* **useContext**: Allows you to access global data easily.

Use of hooks

* Only use hooks at the top level of your components and not inside loops or conditions.
* You can use custom hooks to pull out component logic into reusable functions.
* Hooks simplify component logic, make code more reusable, and improve the overall readability and maintainability of React applications.

**3**-What is Redux-toolkit and how does it work with React? What is prop drilling and how can you avoid it?

**Ans -** Redux Toolkit is a JavaScript library that helps with writing Redux logic for React:

* **What it is**

Redux Toolkit is a toolset that includes utility functions and conventions for writing Redux logic. It combines several Redux packages, such as Redux Thunk, Redux DevTools, and Redux-Immutable.

* **How it works**

Redux Toolkit helps with common Redux use cases, such as defining reducers, store setup, and immutable update logic. It also includes widely used Redux add-ons, like Redux Thunk and Reselect.

* **How it helps with React**

Redux can help avoid prop drilling, which is when data is passed down through multiple layers of components. This can lead to code duplication, which makes code difficult to maintain and debug.

Here are some other things to know about Redux Toolkit:

* It eliminates the need to write action types or action creators by hand.
* It makes it easier to write Redux feature code in one file.
* It was created to help address common concerns about Redux, such as the need for too much boilerplate code, the complexity of configuring a Redux store, and the need to add many packages.

1. **4-** What is Box Model in CSS? How to Create Custom Scrollbars using CSS?

Ans- Scrollbars are common graphical user interface elements that allow users to navigate content beyond the visible area. By default, web browsers provide basic scrollbars with a predefined style. However, using HTML and CSS, you can customize the scrollbar to match your website’s design better.

Scrollbars consist of various elements that can be styled individually, using specific CSS pseudo-elements for WebKit-based browsers (like Chrome, Safari, and Opera) and Firefox. The key pseudo-elements for customizing scrollbars include:

* ::-webkit-scrollbar: The entire scrollbar.
* ::-webkit-scrollbar-track: The track (background) of the scrollbar.
* ::-webkit-scrollbar-thumb: The draggable part of the scrollbar.
* ::-webkit-scrollbar-button: Optional buttons on the scrollbar (not commonly used).
* ::-moz-scrollbar, ::-moz-scrollbar-track, ::-moz-scrollbar-thumb: For customizing Firefox scrollbars.

1. 5- What is “callback hell”? Is it possible to avoid callback hells and how?

### Ans- What is "Callback Hell"?

**Callback hell** (also known as "Pyramid of Doom") refers to a situation in JavaScript programming where multiple nested callbacks are used to handle asynchronous operations. This often leads to deeply nested code that is hard to read, maintain, and debug. As the complexity of the code increases, it becomes challenging to manage and can lead to a structure that resembles a pyramid.

**Is it Possible to Avoid Callback Hell?**

Yes, it is possible to avoid callback hell using several techniques:

1. **Named Functions**:
   * Instead of using anonymous functions as callbacks, use named functions. This makes the code more readable and easier to debug.

**2- Promises**:

* Use Promises to flatten the structure of the code. Promises represent the completion (or failure) of an asynchronous operation and allow for chaining.

**3- Async/Await**:

* Introduced in ES2017, async/await provides a more synchronous way to write asynchronous code. It allows you to write asynchronous code that looks and behaves like synchronous code.

**Q6-** Explain the concept of closures in JavaScript and provide a real-world example where closures would be beneficial in a coding scenario?

**Ans-** A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives a function access to its outer scope. In JavaScript, closures are created every time a function is created, at function creation time.

1. **Q7-** How would you explain the event loop in JavaScript, and why is it crucial for managing asynchronous operations in the language?

**Ans -** Event loop manages asynchronous operations and ensures non-blocking behavior. While JavaScript code is executed sequentially, asynchronous tasks, such as timers or AJAX requests, are scheduled and managed by the event loop, allowing other code to continue executing.

1. **Q8-** Write a function that makes three asynchronous calls using Promises. Ensure they complete in order, regardless of their individual completion times. How would you handle errors in such a scenario?

**Ans -** To achieve sequential execution of three asynchronous calls using Promises in JavaScript, you can utilize async/await syntax along with Promise chaining. Here’s how you can structure the function to ensure they complete in order:

**async function sequentialAsyncCalls() {**

**try {**

**// First async call**

**const result1 = await firstAsyncOperation();**

**console.log('First operation completed with result:', result1);**

**// Second async call**

**const result2 = await secondAsyncOperation();**

**console.log('Second operation completed with result:', result2);**

**// Third async call**

**const result3 = await thirdAsyncOperation();**

**console.log('Third operation completed with result:', result3);**

**// All operations completed successfully**

**return [result1, result2, result3];**

**} catch (error) {**

**// Handle any errors that occur in any of the async calls**

**console.error('Error during async operations:', error);**

**throw error; // Propagate the error further if necessary**

**}**

**}**

**// Example async functions (replace with your actual async functions)**

**function firstAsyncOperation() {**

**return new Promise((resolve) => {**

**setTimeout(() => resolve('First result'), Math.random() \* 1000); // Simulating async delay**

**});**

**}**

**function secondAsyncOperation() {**

**return new Promise((resolve) => {**

**setTimeout(() => resolve('Second result'), Math.random() \* 1000);**

**});**

**}**

**function thirdAsyncOperation() {**

**return new Promise((resolve) => {**

**setTimeout(() => resolve('Third result'), Math.random() \* 1000);**

**});**

**}**

**// Example usage**

**sequentialAsyncCalls()**

**.then(results => {**

**console.log('All async operations completed with results:', results);**

**})**

**.catch(error => {**

**console.error('Error in sequential async calls:', error);**

**});**

### Explanation:

1. **async/await Structure**: The sequentialAsyncCalls function is declared as async to allow the use of await within it, ensuring sequential execution.
2. **Promises and setTimeout**: Inside each async operation function (firstAsyncOperation, secondAsyncOperation, thirdAsyncOperation), a setTimeout is used to simulate asynchronous behavior with a random delay.
3. **Awaiting Promises**: Each async operation is awaited (await firstAsyncOperation(), await secondAsyncOperation(), await thirdAsyncOperation()), ensuring that they execute in sequence.
4. **Error Handling**: Errors are caught using a try/catch block inside sequentialAsyncCalls. If any operation throws an error (either from the async function itself or if await rejects), it will be caught and logged.
5. **Returning Results**: If all operations complete successfully, their results are returned as an array ([result1, result2, result3]).
6. **Example Usage**: After defining the function, you can call sequentialAsyncCalls() and handle the results or errors using .then() and .catch().

**Q9- Explain ES6 feature in JavaScript with example?**

Ans - ES6 introduced a new feature called modules, in which each module is represented by a separate ". js" file. We can use the "import" or "export" statement in a module to import or export variables, functions, classes or any other component from/to different files and modules.

**const numbers = [1, 2, 3, 4, 5];**

**// Using an arrow function to double the numbers in the array**

**const doubled = numbers.map(num => num \* 2);**

**console.log(doubled); // Output: [2, 4, 6, 8, 10]**