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

Ans. Controlled Components

- **State Management**: In controlled components, the form data is handled by the React component's state. The component has complete control over the input values.
- **Value Prop**: The value of the input element is set through a `value` prop that comes from the component's state. This means that any change in the input field is reflected in the component's state.
- **Event Handlers**: You typically use event handlers (like `onChange`) to update the component's state whenever the input value changes.

Uncontrolled Components

- **State Management**: In uncontrolled components, form data is handled by the DOM itself rather than by the component's state. This means that the component does not directly control the input's value.
- **Ref Prop**: To access the value of an input field, you can use React refs (using `useRef` or `createRef`). The input's value can be read directly from the DOM when needed.
- **Less React Involvement**: Since the component doesn't manage the state, it may result in less re-rendering of components, which can be beneficial for performance in some scenarios.

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

Ans.

- **Simplicity and Readability:** Functional components are simpler and easier to read compared to class components. They are plain JavaScript functions that return JSX.
- **Stateless and Stateful:** Initially, functional components were stateless, but with hooks, they can now manage state and lifecycle events, making them just as powerful as class components.
- **Performance:** Functional components can be more performant than class components since they don't have the overhead of maintaining an instance. They also optimize re-renders with React's memoization techniques.
- **Better Composition:** Functional components promote a cleaner and more functional programming style, allowing for easier composition and reusability.
- **Hooks:** The introduction of hooks allows functional components to manage state and side effects, making them versatile for a wide range of use cases.

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 library that provides a set of tools and best practices for efficiently managing application state in a Redux application. It simplifies the process of configuring a Redux store and managing global state by offering a more concise API and helpful utilities.

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

Ans. The Box Model is a fundamental concept in CSS that describes how the dimensions and spacing of elements are calculated. Every HTML element can be considered as a rectangular box composed of the following layers:

1. **Content:** The innermost area, where text, images, and other media are displayed. Its size is determined by the `width` and `height` properties.
2. **Padding:** The space between the content and the border. Padding creates space inside the box and can be adjusted using the `padding` property. It is transparent and does not have a background color.
3. **Border:** The boundary that surrounds the padding (if any) and the content. The border can be styled with colors, widths, and types (solid, dashed, etc.) using the `border` property.
4. **Margin:** The outermost space that separates the box from adjacent boxes. Margins create space outside the element and can be adjusted using the `margin` property. Margins are also transparent.

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

Ans. Callback hell refers to the phenomenon in asynchronous programming where multiple nested callbacks lead to complex, difficult-to-read, and maintain code. This often happens when performing a series of asynchronous operations that depend on the results of previous ones. The structure resembles a pyramid or a deep nesting of functions, making it challenging to follow the flow of execution.

Yes, it is possible to avoid callback hell through several strategies:

1. **Promises**
2. **Async/Await**

3. **Error Handling**

6. 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 in JavaScript is a feature that allows a function to access variables from its outer (enclosing) scope even after that outer function has finished executing. Closures are created every time a function is defined inside another function. They enable private variables and encapsulation, which is useful for data hiding and maintaining state in a clean way.

E.g. Consider a scenario where we want to create a simple counter function. The counter should have the ability to increment, decrement, and reset its value. Using closures, we can encapsulate the counter's state and provide controlled access to it.

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

Ans. The event loop is a fundamental concept in JavaScript that allows the language to perform non-blocking operations, manage asynchronous code execution, and handle events efficiently.
The event loop is essential for managing asynchronous operations in JavaScript, allowing it to maintain a responsive user interface while executing potentially time-consuming tasks in the background.

8. 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.

```
const asyncCall = (message, delay) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {

      if (Math.random() < 0.2) {
        reject(`Error occurred in ${message}`);
      } else {
        resolve(`Completed: ${message}`);
      }
    }, delay);
  });
```

```
};


const makeAsyncCalls = () => {
  asyncCall("First Call", 1000)
    .then((result) => {
      console.log(result);
      return asyncCall("Second Call", 500);
    })
    .then((result) => {
      console.log(result);
      return asyncCall("Third Call", 300);
    })
    .then((result) => {
      console.log(result);
      console.log("All calls completed successfully.");
    })
    .catch((error) => {
      console.error(error);
    });
};


makeAsyncCalls();
```

If any of the calls fail, we can handle the error using a `.catch()` block at the end of the chain.

9. Explain ES6 feature in JavaScript with example?

Ans. ECMAScript 2015 was the second major revision to JavaScript. ECMAScript 2015 is also known as ES6 and ECMAScript 6.

1. Arrow Functions: Arrow functions provide a shorter syntax for writing function expressions
E.g. const add = (a, b) => { return a + b; };

2. Destructuring Assignment: Destructuring assignment enables unpacking values from arrays or properties from objects into distinct variables.
E.g. const numbers = [1, 2, 3]; const [a, b] = numbers;

3. Spread and Rest Operators: The spread operator (...) allows an iterable such as an array to be expanded in places where zero or more arguments are expected. The rest operator allows us to represent an indefinite number of arguments as an array.

E.g. const arr1 = [1, 2, 3];

const arr2 = [4, 5, ...arr1];