```
    // Add the last sequence
    result += count + input[input.length - 1];

    return result;
}

const input = "abbcccddddeea";
const output = encodeString(input);
console.log(output);  // Outputs: 1a2b3c4d2e1a
```

---

52 Reactjs Interview questions & Answers

1. What is React?

- React is an opensource component based JavaScript library which is used to develop interactive user interfaces.

---

2. What are the features of React ?

- Jsx
- Virtual dom
- one way data binding
- Uses reusable components to develop the views
- Supports server side rendering

---

3. What is JSX ?

- JSX means javascript xml. It allows the user to write the code similar to html in their javascript files.
- This jsx will be transpiled into javascript that interacts with the browser when the application is built.

---

4. What is DOM ?

- DOM means document object model. It is like a tree like structure that represents the elements of a webpage.

---

5. What is Virtual Dom ?

- When ever any underlying data changes or whenever user enters something in textbox then entire UI is rerendered in a virtual dom representation.
- Now this virtual dom is compared with the original dom and creates a changeset which will be applied on the real dom.
- So instead of updating the entire realdom, it will be updated with only the things that have actually been changed.

Rendering Efficiency:

- **Virtual DOM**: Updates to the Virtual DOM are typically faster because they occur in memory and are not directly reflected in the browser.
- **Actual DOM**: Manipulating the actual DOM is slower due to its complexity and the potential for reflows and repaints.

---

6. What is state in Reactjs?

- State is an object which holds the data related to a component that may change over the lifetime of a component.
- When the state changes, the component re-renders.
- Eg: for functional component and class component

```
 import React, { useState } from "react";

function User() {
  const [message, setMessage] = useState("Welcome to React

  return (
    <div>
      <h1>{message}</h1>
    </div>
  );
}
```

```
import React from 'react';
class User extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      message: "Welcome to React world",
    };
  }

  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    );
  }
}
```

---

7. What is the purpose of callback function as an argument of `setState()`?

- If we want to execute some logic once state is updated and component is rerendered then we can add it in callback function.

---

8. What are props ?

- Props are inputs to the component.
- They are used to send data from parent component to child component.
- Props are immutable, so they cannot be modified directly within the child component.
- **Example:**

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const name = "John";

  return (
    <div>
      <h1>Parent Component</h1>
      <ChildComponent name={name} />
    </div>
  );
}

export default ParentComponent;
```

```
// ChildComponent.js
import React from 'react';

function ChildComponent(props) {
  return (
    <div>
```

```
      <h2>Child Component</h2>
      <p>Hello, {props.name}!</p>
    </div>
  );
}

export default ChildComponent;
```

9. What are the differences between State and Props in react ?

- Both props and state are used to manage the data of a component.
- State is used to hold the data of a component whereas props are used to send data from one component to another component.
- State is mutable but props are immutable.
- Any change in state causes rerender of component and its children.

10. What is props drilling ?

- Props drilling is the process of sending the data from one component to the component thats needs the data from several interconnected components

11. What are the disadvantages of props drilling and How we can avoid props drilling ?

- **Code complexity:**

  Prop drilling can make code difficult to read and maintain, especially in large applications with many components. This is because props need

to be passed down through multiple levels of components, and it can be difficult to keep track of which components are using which props.

- **Reduced maintainability:**

  Prop drilling can also make code less maintainable. This is because if a prop needs to be changed, the change needs to be propagated through all of the components that use it. This can be a time-consuming and error-prone process.

- **Increased risk of errors:**

  Prop drilling can also increase the risk of errors. This is because it can be difficult to keep track of which components are using which props, and it can be easy to forget to pass a prop down to a component that needs it. This can lead to errors in the application.

- **Performance overhead:**

  Prop drilling can also have a performance overhead. This is because every time a prop is passed down to a component, the component needs to re-render. This can be a significant performance overhead in large applications with many components.

Makes application slower.

We can avoid props drilling using context api or Redux or by using any state management libraries.

---

12. What is useReducer hook ?

- It is an alternative to useState hook which is used when state of the component is complex and requires more than one state variable.

---

13. What is useMemo ?

- useMemo is useful for performance optimization in react.
- It is used to cache the result of a function between re-renders.

- **Example :**
  - In our application we have a data vizualization component where we need to display charts based on performing complex calculations on some large data sets. By using useMemo we can cache the computed result, which ensures that the component does not recalculate on every re-renders.
  - This saves computational resources and provides smoother user experience.

```
import React, { useMemo } from 'react';

const DataVisualization = ({ data }) => {
  const processedData = useMemo(() => {
    // Perform expensive computations on data
    // ...
    return processedData;
  }, [data]);

  // Render the visualization using the processed data
  // ...

  return <div>{/* Visualization component */}</div>;
};
```

In this example, the `processedData` is memoized using `useMemo` to avoid recomputing it on every render. The expensive computations are performed only when the `data` prop changes.

---

14. What is useCallback ?

- useCallback caches the function defination between the re-renders
- It takes two arguments: the callback function and an array of dependencies. The callback function is only recreated if one of the dependencies has changed.

Good Ref: https://deadsimplechat.com/blog/usecallback-guide-use-cases-and-examples/#the-difference-between-usecallback-and-declaring-a-

function-directly

---

15. What are the differences between useMemo and useCallback ?

- Both useMemo and useCallback are useful for performance optimization.
- useMemo will cache the result of the function between re-renders whereas useCallback will cache the function itself between re-renders.

---

16. Which lifecycle hooks in class component are replaced with useEffect in functional components ?

1. **componentDidMount():** equivalent to useEffect with empty array.

```
useEffect(()=>{
    console.log("Called on initial mount only once")
},[])
```

2. **componentDidUpdate():** equivalent to useEffect with array of dependencies

```
useEffect(()=>{
    console.log("Called on every dependency update")
},[props.isFeature,props.content])
```

This will be called whenever dependency value changes (here Eg: isFeature or content).

3. **componentDidUnmount():** equivalent to useEffect with return statement.

```
useEffect(()=>{
    return ()=>{
        console.log("Any cleanup activities/unsubscribing
     }
})
```

---

17. What is component life cycle of React class component ?

- React life cycle consists of 3 phases.
  - mounting
  - updating
  - unmounting
- **Mounting:**
  - In this phase the component is generally mounted into the dom.
  - It is an initialization phase where we can do some operations like getting data from api, subscribing to events etc.

  1. **Constructor:**
     - It is a place to set the initial state and other initial values.
  2. **getDerivedStateFromProps:**
     - This is called right before rendering the elements into the dom.
     - Its a natural place to set the state object based on the initial props.
     - It takes state as an argument and returns an object with changes to the state.

```
getDerivedStateFromProps(props,state){
  return { favColor: props.favColor }
```

```
}
```

3. **render():**
    - It contains all the html elements and is method that actually outputs the html to the dom.
4. **ComponentDidMount():**
    - This is called once component is mounted into the dom.
    - Eg: fetch api calls, subscribing to events etc.

- **Updating phase:**
    - This is when the component is updated. The component will be updated when ever there is change in state or props.

1. **getDerivedStateFromProps:** same as above
2. **ShouldComponentUpdate:**
    - This will return boolean value that specifies whether react should continue with the rendering or not. default is true.

```
shouldComponentUpdate(){
    return true/false
}
```

3. **Render:** same as above
4. **getSnapshotBeforeUpdate:**
    - It will have access to the props and state before update. means that even after the update you can check what are the values were before update.

```
getSnapshotBeforeUpdate(prevProps,prevState){
    console.log(prevProps,prevState)
}
```

5. **ComponentDidUpdate:**
    - Called after the component is updated in the dom.

- **Unmounting phase:**
  - In this phase the component will be removed from the dom. here we can do unsubscribe to some events or destroying the existing dialogs etc.

  1. **ComponentWillUnmount:**
     - This is called when component is about to be removed from the dom.

---

18. What are keys in React ?

- Keys are used to uniquely identify the elements in the list.
- react will use this to indentify, which elements in the list have been added, removed or updated.

```
function MyComponent() {
  const items = [
    { id: 1, name: "apple" },
    { id: 2, name: "banana" },
    { id: 3, name: "orange" }
  ];

  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}
```

---

19. What are fragments in react ?

- React fragments allows us to wrap or group multiple elements without adding extra nodes to the dom.

---

20. What are Pure components in React ?

- A component which renders the same output for the same props and state is called as pure component.
- It internally implements **shouldComponentUpdate** lifecycle method and performs a shallow comparision on the props and state of the component. If there is no difference, the component is not re-rendered.
- **Advantage:**
  - It optimizes the performance by reducing unnecessary re-renders.

Example for class component:

```
import React, { PureComponent } from 'react';

class MyPureComponent extends PureComponent {
  render() {
    return (
      <div>
        <h1>Pure Component Example</h1>
        <p>Props: {this.props.text}</p>
      </div>
    );
  }
}

export default MyPureComponent;
```

Reference: https://react.dev/reference/react/PureComponent

```
import { PureComponent, useState } from 'react';

class Greeting extends PureComponent {
```

```
  render() {
    console.log("Greeting was rendered at", new Date().toL
    return <h3>Hello{this.props.name && ', '}{this.props.n
  }
}

export default function MyApp() {
  const [name, setName] = useState('');
  const [address, setAddress] = useState('');
  return (
    <>
      <label>
        Name{': '}
        <input value={name} onChange={e => setName(e.targe
      </label>
      <label>
        Address{': '}
        <input value={address} onChange={e => setAddress(e
      </label>
      <Greeting name={name} />
    </>
  );
}
```

21. What are the differences between controlled and uncontrolled components ?

- **Controlled components:**
  - In controlled components, the form data is handled by the react component
  - We use event handlers to update the state.
  - React state is the source of truth.
- **Uncontrolled components:**
  - In uncontrolled components, the form data is handled by the dom.

- We use Ref's to update the state.
- Dom is the source of truth.

feature	uncontrolled	controlled
one-time value retrieval (e.g. on submit)	✅	✅
validating on submit	✅	✅
instant field validation	❌	✅
conditionally disabling submit button	❌	✅
enforcing input format	❌	✅
several inputs for one piece of data	❌	✅
dynamic inputs	❌	✅

Ref: https://goshacmd.com/controlled-vs-uncontrolled-inputs-react/

---

22. Whatare Ref's in React?

- ref's are the way to access the dom elements created in the render method.
- they are helpful when we want to update the component whith out using state and props and prevents triggering rerender.

Common useCases:

- Managing input focus and text selection
- Media control/Playback
- Communicating between react components that are not directly related through the component tree.

Examples:

1. Managing input focus

```
function App() {
  const inputRef = useRef();

  const focusOnInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input type='text' ref={inputRef} />
      <button onClick={focusOnInput}>Click Me</button>
    </div>
  );
}
```

2.Managing Audio playback:

```
function App() {
  const audioRef = useRef();

  const playAudio = () => {
    audioRef.current.play();
  };

  const pauseAudio = () => {
    audioRef.current.pause();
  };

  return (
    <div>
      <audio
        ref={audioRef}
        type='audio/mp3'
        src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/
      ></audio>
      <button onClick={playAudio}>Play Audio</button>
      <button onClick={pauseAudio}>Pause Audio</button>
    </div>
```

```
    );
}
```

Reference: https://www.memberstack.com/blog/react-refs

---

23. What is meant by forward ref ?

- In React, forwardref is a technique which is used to send the ref from parent component to one of its children. This is helpful when we want to access the child component dom node from the parent component.

Example:

1. **Creating the Forward Ref Component:** Define a component that forwards the ref to a child component.

```
import React, { forwardRef } from 'react';

const ChildComponent = forwardRef((props, ref) => {
  return <input ref={ref} />;
});

export default ChildComponent;
```

2. **Using the Forward Ref Component:** Use this component and pass a ref to it.

```
import React, { useRef } from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const inputRef = useRef(null);

  return (
    <div>
      <ChildComponent ref={inputRef} />
      <button onClick={() => inputRef.current.focus()}>Foc
```

```
      </div>
    );
}

export default ParentComponent;
```

In this example, `ChildComponent` is a functional component that forwards the ref it receives to the `input` element it renders. Then, in the `ParentComponent`, a ref is created using `useRef`, passed to `ChildComponent`, and used to focus the input when a button is clicked.

Reference: https://codedamn.com/news/reactjs/what-are-forward-refs-in-react-js

---

24. What are Error boundaries ?

- Error boundaries are one of the feature in react which allows the developers to catch the errors during the rendering of component tree, log those erros and handle those errors without crashing the entire application by displaying a fallback ui.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallb
    return { hasError: true };
  }

  componentDidCatch(error, info) {
    // Example "componentStack":
    //   in ComponentThatThrows (created by App)
    //   in ErrorBoundary (created by App)
    //   in div (created by App)
```

```
    //   in App
    logErrorToMyService(error, info.componentStack);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return this.props.fallback;
    }

    return this.props.children;
  }
}
Then you can wrap a part of your component tree with it:

<ErrorBoundary fallback={<p>Something went wrong</p>}>
  <Profile />
</ErrorBoundary>
```

---

25. What are Higher order components in react ?

- Higher order component is a function which takes the component as an argument and returns a new component that wraps the original component.

For example if we wanted to add a some style to multiple components in our application, Instead of creating a `style` object locally each time, we can create a HOC that adds the `style` objects to the component that we pass to it.

```
function withStyles(Component) {
  return props => {
    const style = { padding: '0.2rem', margin: '1rem' }
    return <Component style={style} {...props} />
  }
```

```
}

const Button = () = <button>Click me!</button>
const Text = () => <p>Hello World!</p>

const StyledButton = withStyles(Button)
const StyledText = withStyles(Text)
```

26. What is Lazy loading in React ?

- It is a technique used to improve the performance of a webapplication by splitting the code into smaller chunks and loading them only when its required intead of loading on initial load.

```
import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'))

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}

export default App;
```

27. What is suspense in React ?

The lazy loaded components are wrapped by **Suspense.**The Suspense component receives a fallback prop which is displayed until the lazy loaded component in rendered.

```
import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'))

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}

export default App;
```

---

28. What are custom hooks ?

- Custom hooks helps us to extract and reuse the stateful logic between components.
- Eg:
  - Fetch data
  - To find user is in online or offline.

https://react.dev/learn/reusing-logic-with-custom-hooks

```
import { useState, useEffect } from 'react';
```

```
// Custom hook to fetch data from an API
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const result = await response.json();
        setData(result);
      } catch (error) {
        setError(error);
      } finally {
        setLoading(false);
      }
    };

    fetchData();

    // Cleanup function
    return () => {
      // Cleanup logic if needed
    };
  }, [url]); // Dependency array to watch for changes in t

  return { data, loading, error };
}

// Example usage of the custom hook
function MyComponent() {
  const { data, loading, error } = useFetch('https://api.e

  if (loading) {
```

```
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>Error: {error.message}</div>;
  }

  return (
    <div>
      {data && (
        <ul>
          {data.map(item => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      )}
    </div>
  );
}

export default MyComponent;
```

29. What is context api in react ?

- Context api is a feature in react by using which we can share the data across the application with out having to pass the data manually through every level of component tree.
- It solves the problem of props drilling in react.

We need to follow 3 main steps to implement context api in our application.

- Create context
- Create Provider and wrap this provider around root level component and pass the data.
- Create consumer and utilize data using useContext.

Examples Where we can use context api:

- Shopping cart: Managing cart data in ecommerce application and share between product listings and checkout pages.
- Authentication: sharing user information across the application.
- Data fetching : Sharing fetched data across multiple components that need to display this data. (for displaying search results when user makes a search request).

---

30. Give an example of context api usage ?

Example: Once user loggedin, I wanted to share the user information between the components.

- **Step - 1:** We need to create a context using **createContext** method in userContext.js file.

```
UserContext.js

import React,{createContext} from "react";

const UserContext = createContext();
const UserProvider = UserContext.Provider;
const UserConsumer = UserContext.Consumer;

export {UserProvider, UserConsumer};
```

- **Step - 2:** We need to wrap the root level component with provider and pass the user Information.

```
App component:

import React from "react";
import {ComponentA} from "./ComponentA.js";
import {UserProvider} from "./UserContext.js";
```

```
const App = () => {
  const userinfo = {
    id:"1",
    name:"saikrishna"
  }

  return (
    <div>
      <UserProvider value={userinfo}>
        <ComponentA/>
      </UserProvider>
    </div>
  );
};
```

- **Step - 3:** In componentA, We can get the data using useContext and utilize the user Information.

```
ComponentA:

import React,{useContext} from "react";
import {UserConsumer} from "./UserContext.js";

export const ComponentA = () => {
  const {id,name} = useContext(UserConsumer);

  return <div>Hello {id}{name}</div>
}
```

31. What is Strict mode in react ?

It identifies the commonly occured bugs in the development time itself.

- Checks if there are any depricated apis usage.
- Checks for deprecated lifecycle methods.

- Checks if Duplicate keys in list.
- Warns about Possible memory leaks. etc.

```
================================================================
// Enabling strict mode for entire App.
================================================================

import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';

const root = createRoot(document.getElementById('root'));
root.render(
  <StrictMode>
    <App />
  </StrictMode>
);

================================================================
// Any part of your app
================================================================

import { StrictMode } from 'react';

function App() {
  return (
    <>
      <Header />
      <StrictMode>
        <main>
          <Sidebar />
          <Content />
        </main>
      </StrictMode>
      <Footer />
    </>
  );
}
```

Good Ref: https://dev.to/codeofrelevancy/what-is-strict-mode-in-react-3p5b

---

32. What are the different ways to pass data from child component to parent component in react ?

There are 4 common ways to send data from child component to parent component. They are.,

1. Callback Functions
2. Context API
3. React Hooks (useRef)
4. Redux

---

33. How do you optimize your react application ?

- **Code Splitting:** Break down large bundles into smaller chunks to reduce initial load times
- **Lazy Loading:** Load non-essential components\asynchronously to prioritize critical content.
- **Caching and Memoization:** Cache data locally or use memoization libraries to avoid redundant API calls and computations.
- **Memoization:** Memoize expensive computations and avoid unnecessary re-renders using tools like React.memo and useMemo.
- **Optimized Rendering:** Use shouldComponentUpdate, PureComponent, or React.memo to prevent unnecessary re-renders of components.
- **Virtualization:** Implement virtual lists and grids to render only the visible elements, improving rendering performance for large datasets.
- **Server-Side Rendering (SSR):** Pre-render content on the server to improve initial page load times and enhance SEO.

- **Bundle Analysis:** Identify and remove unused dependencies, optimize images, and minify code to reduce bundle size.
- **Performance Monitoring:** Continuously monitor app performance using tools like Lighthouse, Web Vitals, and browser DevTools.
- **Optimize rendering with keys:** Ensure each list item in a mapped array has a unique and stable key prop to optimize rendering performance. Keys help React identify which items have changed, been added, or removed, minimizing unnecessary DOM updates.
- **CDN Integration:** Serve static assets and resources from Content Delivery Networks (CDNs) to reduce latency and improve reliability.

---

34. What are Portals in react ?

- Portals are the way to render the child components outside of the parent's dom hierarchy.
- We mainly need portals when a React parent component has a hidden value of overflow property(overflow: hidden) or z-index style, and we need a child component to openly come out of the current tree hierarchy.
- It is commonly used in Modals, tooltips, loaders, notifications toasters.
- This helps in improving the performance of application.

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyPortal extends React.Component {
  render() {
    return ReactDOM.createPortal(
      this.props.children,
      document.getElementById('portal-root')
    );
  }
}
```

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>My App</h1>
        <MyPortal>
          <p>This is rendered in a different part of the D
        </MyPortal>
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById('root'));
```

35. What are synthetic events in react ?

- Synthetic events are the wrapper around native browser events.
- By using this react will remove browser inconsistencies and provides a unified api interface for event handling
- They optimise the performance by event pooling and reusing event objects.

36. What are the difference between Package.json and Package.lock.json

- **Package.json:** is a metadata file that contains information about your project, such as the name, version, description, author, and most importantly, the list of dependencies required for your project to run. This file is used by npm (Node Package Manager) to install, manage, and update dependencies for your project.

- **Package.lock.json:** is a file that npm generates after installing packages for your project. This file contains a detailed description of the dependencies installed in your project, including their versions and the dependencies of their dependencies. This file is used by npm to ensure that the same version of each package is installed every time, regardless of the platform, environment, or the order of installation.
- Package.json is used to define the list of required dependencies for your project, while package-lock.json is used to ensure that the exact same versions of those dependencies are installed every time, preventing version conflicts and guaranteeing a consistent environment for your project.

---

37. What are the differences between client side and server side rendering ?

- **Rendering location:** In csr, rendering occurs on the client side after receiving raw data from the server where as in ssr, rendering occurs on server side side and server returns the fully rendered html page to the browser.
- **Initial Load time:** csr has slow initial load time as browser needs to interpret the data and render the page. where as ssr has faster initial load times as server send pre-rendered html page to the browser.
- **Subsequent interactions:** subsequent interactions in csr involves dynamic updates with out requiring full page reload whereas, ssr requires full page reload as server generates new html for every interactions.
- **Seo:** Ssr is seo friendly when compared to csr as fully rendered html content is provided to the search engine crawlers whereas csr needs to parse javascript heavy content.

---

38. What is react-router ?

React-router is a javascript library which is used to implement routing in react applications.
It allows us to navigate between views or pages with out refreshing the page.

- **Router:** It wraps the entire application and provides the routing context for the application. It contains 2 types of routers,
  Browser router and Hash router.
- **Route:** It contains mapping between urlpath and the component. When the url matches, respective component will be rendered.
- **Link:** is used to create the navigation links which user can click to navigate to different routes.
- **switch:** is used to render the first matching route among its children. It ensures only one route is rendered.

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch, Link } fr

function Home() {
    return <h2>Home</h2>;
}

function About() {
    return <h2>About</h2>;
}

function Contact() {
    return <h2>Contact</h2>;
}

function App() {
    return (
        <Router>
            <div>
                <nav>
```

```
                    <ul>
                        <li>
                            <Link to="/">Home</Link>
                        </li>
                        <li>
                            <Link to="/about">About</Link>
                        </li>
                        <li>
                            <Link to="/contact">Contact</L
                        </li>
                    </ul>
                </nav>

                <Switch>
                    <Route exact path="/" component={Home}
                    <Route path="/about" component={About}
                    <Route path="/contact" component={Cont
                </Switch>
            </div>
        </Router>
    );
}

export default App;
```

39. What are protected routes in react ?

- By using the protected routes, we can define which users/user roles have access to specific routes or components.

40. What is the difference between useEffect and useLayoutEffect ?

- useeffect is asynchronous where as uselayouteffect is synchronous
- uselayouteffect runs after browser calculation of dom measurements and before browser repaints the screen whereas useffect runs parallely.
- uselayouteffect really needed when we need to do something based on layout of the dom, if we want to measure dom elements and do something etc.

Ref : https://www.youtube.com/watch?v=wU57kvYOxT4

https://mtg-dev.tech/blog/real-world-example-to-use-uselayouteffect

---

41. Practical question: How to send data from child to parent using callback functions ?

- Define a function in the parent component that takes data as an argument.
- Pass this function as a prop to the child component.
- When an event occurs in the child component (like a button click), call this function with the data to be passed to the parent.

Parent Component:

```
import React, { useState } from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const [dataFromChild, setDataFromChild] = useState('');

  const handleDataFromChild = (data) => {
    setDataFromChild(data);
  };

  return (
    <div>
      <ChildComponent onData={handleDataFromChild} />
      <p>Data from child: {dataFromChild}</p>
```

```
    </div>
  );
}

export default ParentComponent;
```

Child Component:

```
import React from 'react';

function ChildComponent({ onData }) {
  const sendDataToParent = () => {
    const data = 'Hello from child';
    onData(data);
  };

  return (
    <button onClick={sendDataToParent}>Send Data to Parent
  );
}

export default ChildComponent;
```

---

42. Practical question: How to send the data from child component to parent using useRef ?

- It is used to store the data without re-rendering the components.
- It will not trigger any event by itself whenever the data is updated.

Parent component:

```
import React from "react";
import TextEditor from "./TextEditor";

export default function Parent() {
  const valueRef = React.useRef("");
```

```
  return (
    <>
      <TextEditor valueRef={valueRef} />
      <button onClick={() => console.log(valueRef.current)
    </>
  );
}
```

Child component:

```
export default function TextEditor({ valueRef }) {
  return <textarea onChange={(e) => (valueRef.current = e.
}
```

---

43. Practical question: Create a increment decrement counter using useReducer hook in react ?

```
import React, { useReducer } from 'react';

// Initial state
const initialState = {
  count: 0
};

// Reducer function
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
```

```
    default:
      return state;
  }
};

// Component
const Counter = () => {
  const [state, dispatch] = useReducer(reducer, initialSta

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment'
      <button onClick={() => dispatch({ type: 'decrement'
    </div>
  );
};

export default Counter;
```

---

44. Practical question: create a custom hook for increment/decrement counter ?

- **Create a custom hook for counter:**

```
useCounter.js:

export const useCounter = () => {
  const [counter, setCounter] = useState(0);

  const increment = () => {
    setCounter(counter + 1);
  };

  const decrement = () => {
    setCounter(counter - 1);
  };
```

```
  return {
    counter,
    increment,
    decrement,
  };
};
```

- **Utilize useCounter in our component:**

```
component.js:
import { useCounter } from './useCounter.js';

const App = () => {
  const { counter, increment, decrement } = useCounter();

  return (
    <div>
      <button onClick={decrement}>Decrease</button>
      <div>Count: {counter}</div>
      <button onClick={increment}>Increase</button>
    </div>
  );
};
```

---

45. Machine coding question: Dynamic checkbox counter

Display 4 checkboxes with different names and a button named selectall

User can select each checkbox

Select all button click will check all checkboxes

Button should be disabled once all checkboxes are selected.

Display selected checkboxes count and names in ui.

```
import React, { useState } from 'react';
import { render } from 'react-dom';

const Checkbox = ({ label, checked, onChange }) => {
  return (
    <div>
      <label>
        <input type="checkbox" checked={checked} onChange=
        {label}
      </label>
    </div>
  );
};
const App = () => {
  const [checkboxes, setCheckboxes] = useState([
    { id: 1, label: 'Checkbox 1', checked: false },
    { id: 2, label: 'Checkbox 2', checked: false },
    { id: 3, label: 'Checkbox 3', checked: false },
    { id: 4, label: 'Checkbox 4', checked: false },
  ]);

  const [selectAllDisabled, setSelectAllDisabled] = useSta

  const handleCheckboxChange = (id) => {
    const updatedCheckboxes = checkboxes.map((checkbox) =>
      checkbox.id === id
        ? { ...checkbox, checked: !checkbox.checked }
        : checkbox
    );
    setCheckboxes(updatedCheckboxes);
    const allChecked = updatedCheckboxes.every((checkbox)
    setSelectAllDisabled(allChecked);
  };

  const handleSelectAll = () => {
    const updatedCheckboxes = checkboxes.map((checkbox) =>
      ...checkbox,
      checked: !selectAllDisabled,
```

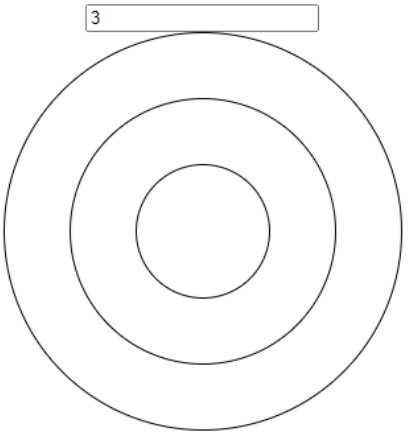
```
    }));
    setCheckboxes(updatedCheckboxes);
    setSelectAllDisabled(!selectAllDisabled);
  };

  const selectedCheckboxes = checkboxes.filter((checkbox) =
  const selectedCount = selectedCheckboxes.length;

  return (
    <div>
      {checkboxes.map((checkbox) => (
        <Checkbox
          key={checkbox.id}
          label={checkbox.label}
          checked={checkbox.checked}
          onChange={() => handleCheckboxChange(checkbox.id
        />
      ))}
      <button onClick={handleSelectAll} disabled={selectAl
        {selectAllDisabled ? 'Deselect All' : 'Select All'
      </button>
      <p>Selected: {selectedCount}</p>
      <ul>
        {selectedCheckboxes.map((checkbox) => (
          <li key={checkbox.id}>{checkbox.label}</li>
        ))}
      </ul>
    </div>
  );
};

render(<App />, document.getElementById('root'));
```

46. Create a nested circles based on user input like below image.



Solution:

Create App component and circle component. We will use recursion concept to achieve this.

App.js:

```
import React, { useState } from "react";
import { Circle } from "./Circle.js";
import "./styles.css";

export default function App() {
  const [num, setNum] = useState(0);

  const handleInput = (e) => {
    setNum(e.target.value);
  };

  return (
    <div className="nested-circles-container">
      <input
        onChange={(e) => handleInput(e)}
        placeholder="Enter number of circles"
        type="number"
      />
      <Circle numbCircles={num}></Circle>
```

```
    </div>
  );
}

App.css/Style.css:

.nested-circles-container {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}
```

```
Circle.js:

import React from "react";
import "./Circle.css";

export const Circle = ({ numbCircles }) => {
  let size = numbCircles * 100;
  return (
    <div
      className="circle-new"
      style={{
        width: `${size}px`,
        height: `${size}px`,
      }}
    >
      {numbCircles > 1 && <Circle numbCircles={numbCircles
    </div>
  );
};

Circle.css:

.circle-new {
```

```
  border-radius: 100%;
  border: 1px solid black;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

47. What are the various design patterns in react ?

- Compound pattern
- HOC Pattern
- Render props pattern
- Container/Presentational pattern

48. What is compound pattern ?

- By using this compound pattern we will create multiple components that work together to perform a single task.
- We can achieve this using context api.
- This promotes separation of concerns between parent and child components, promotes reusability of logic and maintainability.

49. What is render props pettern ?

- With the Render Props pattern, we pass components as props to other components. The components that are passed as props can in turn receive props from that component.
- Render props make it easy to reuse logic across multiple components.

https://javascriptpatterns.vercel.app/patterns/react-patterns/render-props

---

50. What is Container/Presentational pattern ?

It enforces separation of concerns by separating the view from the application logic.

- We can use the Container/Presentational pattern to separate the logic of a component from the view. To achieve this, we need to have a:
  - **Presentational** Component, that cares about **how** data is shown to the user.
  - **Container** Component, that cares about **what** data is shown to the user.

For example, if we wanted to show listings on the landing page, we could use a container component to fetch the data for the recent listings, and use a presentational component to actually render this data.

https://javascriptpatterns.vercel.app/patterns/react-patterns/conpres

---

51. What is React.memo ?

- React.memo is a Higher order component provided by react that memoizes the functional components.
- It caches the result of component's rendering and rerenders the component only when the props have changed.

```
const MyComponent = React.memo((props) => {
    console.log('Rendering MyComponent');
    return (
        <div>
            <h1>Hello, {props.name}!</h1>
            <p>{props.message}</p>
        </div>
    );
});
```

52. Differences between dependencies and devdependencies ?

- **dependencies:** These are the packages that your project needs to run in production. These are essential for the application to function correctly.
- **devDependencies:** These packages are only needed during the development phase.
  - Eg: Jest, react-developer-tools etc.