React Notes

Introduction to React

What is React?

React is a JavaScript library used for building user interfaces, particularly for single-page applications. It was developed by Facebook and allows developers to create large web applications that can update and render efficiently with changing data.

Why React?

Component-Based Architecture: React breaks down the UI into independent, reusable pieces called components.

Virtual DOM: React uses a virtual DOM to improve performance. Instead of updating the entire DOM, it only updates the part that has changed.

Declarative: React makes it easier to design interactive UIs. You simply describe what you want your UI to look like, and React handles the rendering.

What are the key features of React?

Key features of React include:

- Virtual DOM for efficient updates
- Component-based architecture
- JSX for combining HTML with JavaScript
- One-way data binding
- Reusable components
- Unidirectional data flow with Flux or Redux

JSX (JavaScript XML)

Explanation of JSX:

JSX (JavaScript XML) is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript. React utilizes JSX

to define the structure of UI components, making the code more readable and intuitive.

Example:

```
const element = <h1>Hello, world!</h1>;
```

This code snippet creates a React element representing an `h1` tag with the text "Hello, world!" inside it

How does Virtual DOM works?

Virtual DOM works by comparing the current virtual DOM tree with a new virtual DOM tree, and then applying the minimal set of changes to the actual DOM. This allows React to efficiently update the user interface without causing unnecessary re-renders or layout changes.

What are the differences between Real DOM and Virtual DOM?

Real DOM is the actual tree-like structure of a web page, which can be manipulated directly to change the layout or content of the page. Virtual DOM is a lightweight in-memory representation of the actual DOM, which is used to optimise the performance of updates to the user interface.

What are React components?

React components are the building blocks of a React application. They are reusable UI elements that can manage their own state and accept inputs (props) to customise their behaviour and appearance.

Creating and Rendering Components

To create a component we can create both as a functional component and a class component.

Example 1: Functional Component

```jsx
// Greeting.js
import React from 'react';

// Define a functional component
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Export the component to use it in other parts of the app
export default Greeting;
```

Example 2: Class Component

```jsx
// GreetingClass.js
import React, { Component } from 'react';

// Define a class component
class GreetingClass extends Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

// Export the component to use it in other parts of the app
export default GreetingClass;
```

Set up React

Here's a step-by-step guide to setting up a React project from scratch

using different methods. You can choose the method that best suits your needs.

Method 1: Using Create React App (CRA)

`Create React App` is the most common and easiest way to set up a new React project. It comes with a lot of built-in features like webpack, Babel, and a development server.

Steps:

1. I**nstall Node.js and npm**

Make sure you have Node.js installed. You can download it from Node.js official website. This will also install npm (Node Package Manager).

2. **Create a New React App:**
   Open your terminal or command prompt and run the following command:

```
npx create-react-app my-app
```

Replace my-app with your desired project name.
This will create a new directory with your project's name and set up a React project inside it.

3. Navigate to Your Project Directory:

```
cd my-app
```

4. Start the Development Server:

Run the following command to start the development server:

```
npm start
```

This will start the server and open your React app in the browser at http://localhost:3000.

Method 2: Using Vite (Alternative to CRA)

Vite is a modern frontend build tool that provides a faster development experience. It's a great alternative to CRA.

Steps:

1. **Install Node.js and npm:**

Ensure you have Node.js installed as mentioned above.

2. **Create a New React App with Vite:**

Run the following commands:

```
npm create vite@latest my-app
```

Select React(if you prefer TypeScript) when prompted.

Navigate to Your Project Directory:

```
cd my-app
```

3. **Install Dependencies:**

Run the following command to install the necessary dependencies:

```
npm install
```

4. **Start the Development Server:**

Start the development server with:
npm run dev

This will start the server and open your React app in the browser, usually at http://localhost:5173.

How to use CSS in React?

In React, CSS can be added to a component in several ways:
Using inline styles Using a CSS file and importing it into the component
Using a CSS preprocessor like Sass or Less
Using a CSS-in-JS library like styled-components

What is the difference between functional components and class components in React?

- Functional components are simpler and use JavaScript functions to define UI elements.

Example:-

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

function Greeting() {
  const [name, setName] = useState('World');


const handleChange = (event) => {
    setName(event.target.value);
  };
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <input type="text" value={name}
onChange={handleChange} />
    </div>
  );
}
export default Greeting;
```

A class component is a more traditional way of creating components in React. It requires extending `React.Component` and has a `render()`

method that returns JSX. State is managed using `this.state`, and updates are handled using `this.setState()`.

Example:-

```
import React, { Component } from 'react';
class Greeting extends Component {
  constructor(props) {
    super(props);
    // Initializing state
    this.state = {
      name: 'World',
    };
  }

  handleChange = (event) => {
    // Updating state
    this.setState({ name: event.target.value });
  };

  render() {
    return (
      <div>
        <h1>Hello, {this.state.name}!</h1>
        <input  type="text
  value={this.state.name}
   onChange={this.handleChange}
        />
      </div>
    );
  }
}
export default Greeting;
```

What are props in React?

Props (short for properties) are a way to pass data from parent components to child components in React. They are read-only and help in making components reusable and customizable.

Example:-

```jsx
// Greeting.js
import React from 'react';

// Functional component that receives props
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Export the component
export default Greeting;
```

```jsx
// App.js
import React from 'react';
import Greeting from './Greeting'; // Import the Greeting component

function App() {
  return (
    <div>
      {/* Passing different props to the Greeting component */}
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}

export default App;
```

What is state in React?

State is a built-in object in React that allows components to manage their own data. It is mutable and can be updated using the `setState` method. State is used to keep track of component-specific data and trigger re-renders.

Example:-

```jsx
// Counter.js
import React, { useState } from 'react';

function Counter() {
  // Declare a state variable named "count" with an initial value of 0
  const [count, setCount] = useState(0);

  // Function to handle incrementing the count
  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>Click me</button>
    </div>
  );
}

export default Counter;
```

What is the purpose of the `setState` method in React?

The `setState` method is used to update the state of a React component. It schedules a re-render and merges the updated state with the previous state.

What are React lifecycle methods?

React lifecycle methods are special methods that are invoked at different stages of a component's life cycle, such as mounting, updating, and unmounting. Examples include `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

What is the significance of keys in React lists?

Keys are used in React lists to help React identify which items have changed, been added, or been removed. They improve performance by reducing unnecessary re-renders and aiding in efficient updates to the DOM.

What are controlled components in React?
Controlled components in React are form elements whose values are controlled by React state. Changes to the input values are handled by React, making it easier to implement features like validation and form submission.

What is React Router and how does it work?
React Router is a popular routing library for React applications. It allows developers to define routes, map them to specific components, and handle navigation without a full page refresh. React Router uses a component-based approach for routing.

What are React hooks?
React hooks are functions that allow functional components to use state and lifecycle features previously available only in class components. Examples of hooks include `useState`, `useEffect`, `useContext`, and `useReducer`.

How do you handle forms in React?
Forms in React can be handled using controlled components, where form input values are controlled by React state. You can listen to input changes, update state accordingly, and handle form submission using event handlers.

What is the role of the `React.Fragment` component?
A fragment is a way to group a list of children without adding extra nodes to the DOM. It allows you to return multiple elements from a component's render method without wrapping them in an additional DOM node.

Fragments are represented by an empty JSX tag: <> or </>

```jsx
import React from 'react';

function ItemList() {
  return (
    <>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </>
  );
}

export default ItemList;
```

What are the differences between controlled and uncontrolled components in React?

- Controlled components: Input elements whose values are controlled by React state and are updated through state changes.

```jsx
import React, { useState } from 'react';

function ControlledForm() {
  const [name, setName] = useState('');

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Submitted name: ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Uncontrolled components: Input elements that maintain their own state, typically using refs, and do not rely on React state for updates.

For Example:-

```jsx
import React, { useRef } from 'react';

function UncontrolledForm() {
  const nameInput = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Submitted name: ${nameInput.current.value}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" ref={nameInput} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}

export default UncontrolledForm;
```

What are the differences between `props` and `state` in React?

- `props`: Immutable data passed from parent components to child components. Props are read-only and cannot be modified by the child component.

- `state`: Mutable data managed within a component. State can be updated using `setState` and is used for managing component-specific data.

Explain the concept of PureComponent in React.

`PureComponent` is a class component in React that implements a shallow comparison of props and state to determine if a re-render is necessary. It can improve performance by reducing unnecessary re-renders.

Can you explain the useState hook with examples?

The useState hook allows you to add state to a functional component. It returns an array with two values: the current state and a function to update it.

Here's an example of how to use the useState hook:

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

function Counter() {
  // Declare a state variable 'count' and a function to update it 'setCount'
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      {/* Update the state by calling setCount */}
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

Can you explain the useEffect hook?

The useEffect hook allows you to run side effects, such as fetching data or updating the DOM, in a functional component. It takes a callback function as its first argument, which is called after the component has rendered.

Here's an example of how to use the useEffect hook:

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

function MyComponent() {
  const [data, setData] = useState([]);

  useEffect(() => {
    async function fetchData() {
      const response = await fetch('https://my-api.com/data');
      const json = await response.json();
      setData(json);
    }

    fetchData();
  }, []);

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

export default MyComponent;
```

What is useRef being used for?

useRef is a hook that allows you to create a reference to a DOM node or a JavaScript object. It can be used to access a DOM node directly, or to store a value that should not cause a re-render when it changes.
Here's an example of how to use the useRef hook:

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

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

  function handleClick() {
    inputRef.current.value = "Hello World!";
  }

  return (
    <div>
      <input type="text" ref={inputRef}/>
      <button onClick={handleClick}>
        Update Input
      </button>
    </div>
  );
}

export default MyComponent;
```

Can you explain the useContext hook?

The `useContext` hook in React is used to share values between components without explicitly passing props through every level of the component tree. It simplifies state or data management when multiple components need access to the same data.

Example:-

1. Create the Context

First, create a context that will hold the theme value.

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

// Create a Context with a default value of "light"
const ThemeContext = createContext('light');

export default ThemeContext;
```

2. Provide the Context

Next, create a main `App` component where you provide the context value to all the child components.

```
import React from 'react';
import ThemeContext from './ThemeContext';
import ThemedButton from './ThemedButton';

function App() {
  return (
    // Provide the "dark" theme to the entire application
    <ThemeContext.Provider value="dark">
      <div>
        <h1>Welcome to the Themed App</h1>
        <ThemedButton />
      </div>
    </ThemeContext.Provider>
  );
}

export default App;
```

3. Consume the Context

Finally, create a `ThemedButton` component that uses the `useContext` hook to access the theme value and apply it to a button.

```jsx
import React, { useContext } from 'react';
import ThemeContext from './ThemeContext';

function ThemedButton() {
  // Consume the context value using useContext
  const theme = useContext(ThemeContext);

  // Apply different styles based on the theme
  const buttonStyle = {
    backgroundColor: theme === 'dark' ? '#333' : '#FFF',
    color: theme === 'dark' ? '#FFF' : '#000',
    padding: '10px 20px',
    border: 'none',
    borderRadius: '5px',
    cursor: 'pointer',
  };

  return <button style={buttonStyle}>I am a {theme} themed button</button>;
}

export default ThemedButton;
```

Can you explain the useReducer hook?

The `useReducer` hook in React is an alternative to `useState` for managing more complex state logic. It's particularly useful when the state logic involves multiple sub-values or when the next state depends on the previous state. It is similar to how `redux` works in managing state, but it's built into React.

Example:-

1. Define the Reducer Function

The reducer function takes the current state and an action, and returns the new state.

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    case 'reset':
      return { count: 0 };
    default:
      throw new Error();
  }
}
```

2. Use the `useReducer` Hook in a Component

Now, let's create a `Counter` component that uses this reducer function.

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

// Reducer function defined above
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    case 'reset':
      return { count: 0 };
    default:
      throw new Error();
  }
}

function Counter() {
  // Initialize the useReducer hook
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      {/* Dispatch actions on button clicks */}
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
      <button onClick={() => dispatch({ type: 'reset' })}>Reset</button>
    </div>
  );
}

export default Counter;
```

3. Use the `Counter` Component in Your App

Finally, use the `Counter` component in your app.

```jsx
import React from 'react';
import Counter from './Counter';

function App() {
  return (
    <div>
      <h1>Simple Counter with useReducer</h1>
      <Counter />
    </div>
  );
}

export default App;
```

How to create refs?

To create a ref, you can use the useRef hook, which returns a ref object. You can then assign this object to a ref attribute on a JSX element.

Can you create your custom React hooks?

Yes, you can create your custom React hooks by following the naming convention "use" and using state and other hooks inside it.

What do you need to keep in mind while creating custom React hooks?

When creating custom React hooks, it's important to keep in mind that they should only call other hooks at the top level and not inside loops or conditions. Also, it's important to make sure that the hook only performs one specific action.

What is the Context API in React?

The Context API in React allows you to share data between components without passing props through every level of the component tree. It provides a way for components to access data that is "global" to the component tree, such as a user's authentication status or a theme.

The Context API consists of a Provider component, which provides the data, and a Consumer component, which accesses the data.

What is React Router?

React Router is a library for routing in React apps. It allows you to define the different routes in your application and render the appropriate components for each route. This makes it easy to change the displayed content based on the current URL, without having to refresh the page.

What are Pure components and what is their purpose?

Pure components are components that only re-render if their props or state have changed. They are optimised for performance, and they can improve the performance of your application by reducing the number of unnecessary re-renders.

What is "key" prop and what is the benefit of using it in arrays of elements?

The "key" prop is used to give a unique identifier to each item in an array of elements. When elements are re-rendered, React uses the key to identify which elements have changed, added, or removed. This allows React to update the DOM efficiently, improving the performance of the application.

What are the different phases of component lifecycle in React?

The different phases of the component lifecycle in React are:

Mounting: When a component is being added to the DOM.
Updating: When a component's props or state change.
Unmounting: When a component is being removed from the DOM.

What are the lifecycle methods of React?

The lifecycle methods of React are methods that are called at specific points during the lifecycle of a component.

They include:
componentDidMount: Called after the component has been added to the DOM.
componentDidUpdate: Called after the component has been updated.
componentWillUnmount: Called before the component is removed from the DOM.

render: Called whenever the component needs to update the DOM.
constructor: Called before the component is added to the DOM.

Why React uses className Over class attribute?

React uses the className attribute instead of the class attribute because class is a reserved keyword in JavaScript. Using className avoids any confusion and ensures that the attribute will be interpreted as intended.

Why fragments are better than container divs?

Fragments are better than container divs because they don't add an extra node to the DOM. This can make the rendered HTML cleaner and more efficient, especially when you have a component that renders a list of items. Additionally, it helps with accessibility because it doesn't create an unnecessary wrapper element around the content.

What is the use of the react-dom package?

The react-dom package is a package that provides the renderer for React components. It provides the functions that are used to render React components on the web (DOM).

What will happen if you use setState() in constructor?

If you use setState in the constructor, it will cause the component to re- render before it is added to the DOM. This can cause unexpected behaviour and should be avoided. Instead, it is recommended to initialise the state with the constructor's props and use setState in componentDidMount.

How to pass parameter to an event handler?

To pass a parameter to an event handler, you can use an arrow function to wrap the event handler.
For example: <button onClick={() => handleClick(parameter)}>.

What are React Hooks?

React Hooks are functions that allow you to use state and other React features in functional components. They were introduced in React 16.8.

What is an Event in React?

An event in React is a way to respond to user interactions such as clicks, hover, or form submissions.

How to Handle Events in React (for both functional and class components)?

In React, events can be handled using the on keyword, followed by the event name and a callback function. For example, to handle a button click event, you would use onClick={handleClick).
This can be done in both functional and class components.

How to update state in React class components?

To update state in a React class component, you can use the setState method. This method takes an object or function as an argument, and it will merge the new state with the existing state.

How to update state in React functional component?

To update state in a React functional component, you can use the useState hook. This hook returns an array containing the current state and a function to update it.

How to Differentiate Between State and Props?

State is the internal data of a component that can change and is managed by the component itself. Props are external data passed to a component from its parent component. State can be updated by the component, whereas props cannot be updated by the component.

What is React Fiber?

React Fiber is a new reconciliation algorithm that was introduced in React 16. It is designed to improve the performance and flexibility of React by breaking down the render process into smaller, asynchronous chunks.

Lifting State Up is a common pattern in React used when multiple components need to share and synchronize a piece of state. Instead of each component managing its own local state, the state is "lifted up" to the nearest common ancestor component, allowing the shared state to be passed down as props.

Example:-

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

function ParentComponent() {
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <h1>Parent Component</h1>
      <p>Count: {count}</p>
      <ChildComponent count={count} onIncrement={incrementCount} />
    </div>
  );
}

// Child component
function ChildComponent({ count, onIncrement }) {
  return (
    <div>
      <h2>Child Component</h2>
      <p>Count from parent: {count}</p>
      <button onClick={onIncrement}>Increment</button>
    </div>
  );
```

```
}

export default ParentComponent;
```

In the example above:

1. The ParentComponent holds the state (count) and the function to update it (incrementCount).
2. This state and the update function are passed as props to the ChildComponent.
3. The ChildComponent can now display the count and trigger updates through the onIncrement prop.