React.js Preparation Summary

# useContext in React

The `useContext` Hook allows you to access context values in functional components without needing to wrap components in a `Consumer`.

Example:  
```jsx  
const ThemeContext = React.createContext();

function App() {  
 return (  
 <ThemeContext.Provider value="dark">  
 <Toolbar />  
 </ThemeContext.Provider>  
 );  
}

function Toolbar() {  
 return <ThemedButton />;  
}

function ThemedButton() {  
 const theme = useContext(ThemeContext);  
 return <button>{theme}</button>;  
}  
```

# Conditional Rendering in React

Conditional rendering allows you to render components or elements based on a condition.

Example:  
```jsx  
function Greeting(props) {  
 const isLoggedIn = props.isLoggedIn;  
 return (  
 <div>  
 {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}  
 </div>  
 );  
}  
```

# List Rendering using map()

You can use the `map()` function to render lists in React.

Example:  
```jsx  
const numbers = [1, 2, 3, 4, 5];  
const listItems = numbers.map((number) =>  
 <li key={number.toString()}>{number}</li>  
);

function NumberList() {  
 return <ul>{listItems}</ul>;  
}  
```

# Handling Events in React

React handles events using camelCase syntax and you pass functions as event handlers.

Example:  
```jsx  
function MyButton() {  
 function handleClick() {  
 alert('Button was clicked!');  
 }

return (  
 <button onClick={handleClick}>  
 Click me  
 </button>  
 );  
}  
```

# Two-way Data Binding

Two-way data binding means syncing data between input fields and component state.

Example:  
```jsx  
function InputComponent() {  
 const [value, setValue] = useState('');

return (  
 <input value={value} onChange={(e) => setValue(e.target.value)} />  
 );  
}  
```

# Axios

Axios is a promise-based HTTP client for making API requests.

Example:  
```jsx  
import axios from 'axios';

useEffect(() => {  
 axios.get('https://api.example.com/data')  
 .then(response => {  
 console.log(response.data);  
 })  
 .catch(error => {  
 console.error('There was an error!', error);  
 });  
}, []);  
```

# Redux in React

Redux is a state management library for managing global application state.

- Store: Centralized place to hold application state.  
- Actions: Describe what you want to do.  
- Reducers: Describe how state should change.

Example usage with `useSelector` and `useDispatch`:  
```jsx  
const counter = useSelector((state) => state.counter);  
const dispatch = useDispatch();  
```

# What is Store in Redux?

A store is an object that holds the application state. It allows access to the state via `getState()`, dispatching actions using `dispatch()`, and registering listeners with `subscribe()`. It is created using `createStore()` from Redux.

# Activate existing Virtual Environment in VS Code

Use the command below in the terminal where `venv` is your virtual environment folder name:

Windows:  
```  
.\venv\Scripts\activate  
```

macOS/Linux:  
```  
source venv/bin/activate  
```

# Virtual DOM in React

Virtual DOM is a lightweight JavaScript representation of the real DOM. React uses it to optimize updates and rendering by comparing differences (diffing) and updating only what’s necessary.

# Higher-Order Components (HOCs)

HOCs are functions that take a component and return a new component with enhanced behavior.

Example:  
```jsx  
function withLogger(WrappedComponent) {  
 return function(props) {  
 console.log('Rendering', WrappedComponent.name);  
 return <WrappedComponent {...props} />;  
 };  
}  
```

# React Strict Mode

React Strict Mode is a tool for highlighting potential problems in an application.

- Warns about deprecated APIs.  
- Detects unexpected side effects.  
- Helps with async rendering readiness.

Example:  
```jsx  
<React.StrictMode>  
 <App />  
</React.StrictMode>  
```