REACT JS TUTORIAL

Basic Concepts:

1. **Introduction to React:**
    - What is React?
    - Why use React?
    - React's component-based architecture.
2. **Setting Up Your Environment:**
    - Installing Node.js and npm.
    - Creating a new React app using Create React App.
3. **JSX (JavaScript XML):**
    - Introduction to JSX syntax.
    - Embedding expressions in JSX.
    - Using JSX in components.
4. **Components:**
    - Creating functional components.
    - Creating class components.
    - Rendering components.
    - Props and prop drilling.
    - Using defaultProps and propTypes.

5. **State and Lifecycle:**
    - Understanding component state.
    - Updating state using setState.
    - Component lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount).
6. **Handling Events:**
    - Adding event handlers to components.
    - Binding event handlers.
    - Passing data through event handlers.
7. **Conditional Rendering:**
    - Using conditional statements to render components.
    - Ternary operators for conditional rendering.
    - Rendering lists using map.

Intermediate Concepts:

8. **Forms and Controlled Components:**
    - Building controlled forms.
    - Handling form input changes.
    - Submitting and validating forms.
9. **Component Composition:**
    - Composing components.
    - Container and presentational component pattern.

- Children props.

10. **State Management:**
    - Introduction to state management libraries (Redux, MobX).
    - Setting up Redux in a React app.
    - Actions, reducers, and the store.
    - Connecting components to the store.
11. **Routing:**
    - Introduction to React Router.
    - Setting up routes.
    - Navigating between routes.
    - Route parameters and query strings.
12. **Styling in React:**
    - CSS modules.
    - Styled-components library.
    - Inline styling in React.

Advanced Concepts:

13. **Hooks:**
    - Introduction to React Hooks.
    - useState for managing state.
    - useEffect for handling side effects.

- Custom hooks.

14. Context API:

- Managing state with Context API.
- Creating and using context providers and consumers.
- Context API vs. Redux.

15. Performance Optimization:

- Using React.memo for optimizing rendering.
- Avoiding unnecessary re-renders.
- Profiling and optimizing performance using React DevTools.

16. Server-side Rendering (SSR) and Static Site Generation (SSG):

- Introduction to SSR and SSG.
- Using Next.js for SSR and SSG.

17. Testing:

- Writing unit tests using Jest and React Testing Library.
- Testing component behavior and UI.

18. Advanced Patterns:

- Higher-order components (HOCs).
- Render props.
- Compound components.

19. Real-time Data with WebSockets:

- Integrating WebSocket communication.
- Building real-time features in React apps.

20. Authentication and Authorization:

- Implementing user authentication.
- Protecting routes based on user roles.

21. Deployment:

- Optimizing production builds.
- Deploying React apps to various platforms (Heroku, Netlify, Vercel, etc.).

REACT JS INTERVIEW QUESTION WITH ANSWER

Basic React Js Interview Question

1) What is React?

⇨ React is an open-source front-end JavaScript library. That is used for building user interface, especially for single-page-applications. It is also used for web or mobile apps based on components in declarative approach.

⇨ **The important features of React are:**

- It supports server-side rendering.

- It will make use of the virtual DOM rather than real DOM (Data Object Model) as RealDOM manipulations are expensive.
- It follows unidirectional data binding or data flow.
- It uses reusable or composable UI components for developing the view.

2) What are the Advantage of using React?

⇨ MVC is generally abbreviated as Model View Controller.

- **Use of Virtual DOM to improve efficiency:** React uses virtual DOM to render the view. As the name suggests, virtual DOM is a virtual representation of the real DOM. Each time the data changes in a react app, a new virtual DOM gets created. Creating a virtual DOM is much faster than rendering the UI inside the browser. Therefore, with the use of virtual DOM, the efficiency of the app improves.

- **Easy to Learn:** React has a gentle learning curve when compared to frameworks like Angular. Anyone with little knowledge of JavaScript can start building web applications using React.

- **SEO friendly:** React allows developers to develop engaging user interfaces that can be easily navigated in various search engines. It also allows server-side rendering, which boosts the SEO of an app.

- **Reusable components:** React uses component-based architecture for developing applications. Components are independent and reusable bits of code. These components can

be shared across various applications having similar functionality. The re-use of components increases the pace of development.

- **Huge ecosystem of libraries to choose from:** React provides you with the freedom to choose the tools, libraries, and architecture for developing an application based on your requirement.

3) What are the Limitations of React?

⇨ React is not full-stack framework as it is only a library.
⇨ It might be difficult for beginner programmers to understand React.
⇨ Coding might become complex as it will make use of inline templating and JSX.

4) What is useState(), useEffect(), useContext(), useReducer() and useRef() in React?

⇨ **useState()**

- The useState() is a built-in React Hook. It Is work Like variable. It should be used when the DOM has something that is dynamically manipulating/controlling.

- Example :

```
const [count, setCount] = useState(0);

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

return (
```

```
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
```

⇨ **useEffect()**

- **useEffect** is a React Hook that perform side effects in functional components, such as data fetching, updating the DOM, setting up event listeners.
- In class components, side effects were typically managed in lifecycle methods like **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount**.

- The useEffect React Hook will accept 2 arguments:
  useEffect(callback,[dependencies]);

  - Where the **first argument callback** represents the function having the **logic of side-effect** and it will be **immediately executed** after changes were being **pushed to DOM**. The **second argument dependencies** represent an **optional array of dependencies**. The useEffect() will execute the callback only if there is a **change in dependencies in between renderings**.

- Example :

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

function DataFetcher() {
  const [data, setData] = useState([]);
 useEffect(() => {
   fetch('https://api.example.com/data')
    .then((response) => response.json())
```

```
      .then((data) => setData(data));
  }, []);

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

⇨ **useContext()**

- You should use useContext in React when passing data from a parent component to a deep-level child component without passing it down through all intermediate components.
- useContext is a React Hook that allows functional components to consume context (global state) created by React.createContext.

- Example :

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

// Create a context

const ThemeContext = createContext();

// Create a component that uses the context

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return (
```

```
    <button style={{ background: theme.background, color:
theme.foreground }}>
      Themed Button
    </button>
  );
}

// Provide the context value higher in the component tree

function App() {
  const theme = { background: 'blue', foreground: 'white' };

  return (
    <ThemeContext.Provider value={theme}>
      <ThemedButton />
    </ThemeContext.Provider>
  );
}
```

- Difference between **useContext** and **Redux :**

useContext	Redux
useContext is a hook.	Redux is a state management library.
It is used to share data.	It is used to manage data and state.
Changes are made with the Context value.	Changes are made with pure functions i.e. reducers.
We can change the state in it.	The state is read-only. We

	cannot change them directly.
It re-renders all components whenever there is any update in the provider's value prop.	It only re-render the updated components.
It is better to use with small applications.	It is perfect for larger applications.
It is easy to understand and requires less code.	It is quite complex to understand.

⇨ **u**

seReducer()

- useReducer is a React Hook that returns the current state paired with a dispatch method for state updates, similar to Redux reducers.

- Example :

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

// 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;
  }
};
```

```
// APP.JS
function Counter() {

  // Initial state and reducer
  const initialState = { count: 0 };
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT'
})}>Increment</button>
      <button onClick={() => dispatch({ type: 'DECREMENT'
})}>Decrement</button>
    </div>
  );
}
```

⇨ **useRef()**

- useRef is a React Hook that returns a mutable ref object that persists for the lifetime of the component. It can be used to access DOM elements or persist values across renders.

- Example :

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

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

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

  return (
```

```
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={handleButtonClick}>Focus
Input</button>
    </div>
  );
}
```

5) What are Keys in React?

⇨ A key is help in providing the identify to the elements in the Lists (identify which item have changed, added or removed).

⇨ Example :

```
const  users = [
        { id: 1, name: 'Alice' },
        { id: 2, name: 'Bob' },
        { id: 3, name: 'Charlie' },
];

const  userList = users.map((user) => {
  return (
    <li key={user.id}>
      {user.name}
    </li>
  );
});
```

⇨ **Importance of Keys :-**

- Keys help react identify which elements were added, changed or removed.
- Without keys, React does not understand the order or uniqueness of each element.

- Keys should be given to array elements for providing a unique identity for each element.
- With keys, React has an idea of which particular element was deleted, edited, and added.
- Keys are generally used for displaying a list of data coming from an API.

6) What is JSX and JSX in React?

⇨ **JSX** stands for JavaScript XML. it is a **syntax extension to JavaScript**.it allow to write **HTML** Type Tag inside **JavaScript** and place them in **DOM** without using functions like **appendChild()** or **createElement()**.

Let's understand how JSX works:

Without using JSX, we would have to create an element by the following process :

```
const text = React.createElement('p', {}, 'This is a text');
const container = React.createElement('div','{}',text );
ReactDOM.render(container,rootElement);
```

Using JSX, the above code can be simplified:

```
const container = (
<div>
  <p>This is a text</p>
</div>
);
ReactDOM.render(container,rootElement);
```

7) What are the React Components? Explain Different Type of Components in React.

⇨ Components are the building blocks of a user interface. They are reusable, self-contained pieces of code that encapsulate UI logic and can be composed together to create complex user interfaces.

Type of Components :

A) Functional Components (Stateless Components):

Functional components are simple JavaScript functions that accept props (properties) as arguments and return JSX (React elements).

Example :

```
const Greeting = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};
```

B) Class Components (Stateful Components):

Class components are ES6 classes that extend the React.Component class and have their own internal state.

React Class components have a built-in state object.

They can handle complex logic, manage state, and have access to lifecycle methods like **componentDidMount**, **componentDidUpdate**, etc.

Example :

```
import React, { Component } from 'react';
class Counter extends Component {
  constructor(props) {
```

```
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState((prevState) => ({ count:
prevState.count + 1 }))}>
          Increment
        </button>
      </div>
    );
  }
}
```

C) Pure Component:

Pure components are similar to functional components, but they come with a built-in mechanism to optimize rendering.

They implement a shallow comparison of props and state, and if there are no changes, they prevent unnecessary re-rendering.

Example of pure class component:

```
import React from 'react';
```

```
class PureClassComponent extends React.PureComponent {
  render() {
    return <div>{this.props.value}</div>;
  }
}
```

Example of Pure Functional Component using React.memo():

```
import React from 'react';
const PureFunctionalComponent = React.memo(function(props) {
  return <div>{props.value}</div>;
});
```

8) What are difference between functional and class components?

⇨ Before the introduction of Hooks in React, functional components were called **stateless components** and were behind class components on a **feature basis**. After the introduction of Hooks, functional components are **equivalent to class components.**

⇨ Functional components are JavaScript functions that return JSX (JavaScript XML) to describe the UI. They are simpler and more concise than class components.

⇨ Example :

```
import React from 'react';

const FunctionalComponent = () => {
  return <div>Hello, I'm a functional component!</div>;
};
```

```
export default FunctionalComponent;
```

⇨ Class components, as the name suggests, are created using ES6 classes and extend the base **React.Component** class. They define a **render()** method to return JSX to render the UI.

⇨ Example :

```
import React from 'react';

class ClassComponent extends React.Component {
  render() {
    return <div>Hello, I'm a class component!</div>;
  }
}

export default ClassComponent;
```

⇨ **A) State and Lifecylcle Methods :**

- Functional components, prior to React Hooks, were considered "stateless" components because they couldn't handle state or lifecycle methods. functional components can now manage state and use lifecycle methods through the **useState** and other Hook functions.
- Class components, on the other hand, can manage state using the **this.state** and **this.setState** methods. They can also use lifecycle methods such as **componentDidMount**, **componentDidUpdate**, etc., for handling side effects and other tasks.

⇨ **B) Usage of React Hooks :**

- It is allow functional components to manage state, handle side effects, and access context information without the need for class components.

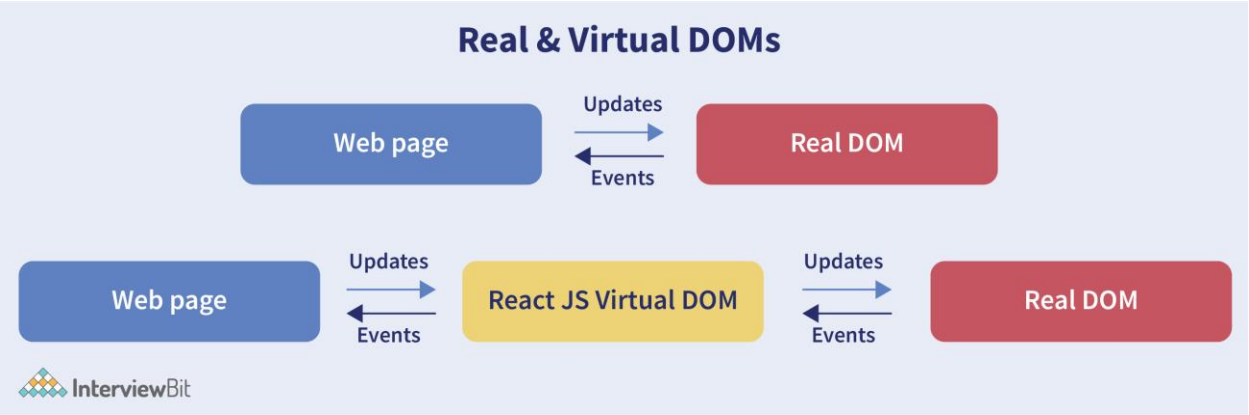
⇨ **C) Performance :**

- Functional components are generally more performant than class components. Class components create instances and require additional overhead, whereas functional components are simple JavaScript functions and have less overhead.

9) What is the Virtual DOM? How does react use the virtual DOM to render the UI?

⇨ DOM stands for Document Object Model. The DOM represents an HTML document with a logical tree structure.

⇨ React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, the virtual DOM changes only that object in the real DOM, rather than updating all the objects.



10) Can Web Browser Read JSX directly?

⇨ Web browser cannot read JSX directly. This is because they are built to only read regular JS Objects and JSX is not regular JavaScript Object.

For a Web Browser to read a JSX file, the file needs to be a transformed into a regular JavaScript Object. For this, Using ***Babel.***

11) What are the differences between controlled and uncontrolled components?

Feature	Uncontrolled	Controlled	Name attrs
One-time value retrieval (e.g. on submit)	✔	✔	✔
Validating on submit	✔	✔	✔
Field-level Validation	✘	✔	✔
Conditionally disabling submit button	✘	✔	✔
Enforcing input format	✘	✔	✔
several inputs for one piece of data	✘	✔	✔
dynamic inputs	✘	✔	

⇨ **Controlled Component :**

- A controlled component is a React component where the form element's value is controlled by React's state. This means that the value of the input element is directly tied to a state variable and is

only updated through a handler function that updates the state. The component re-renders whenever the state changes, which in turn updates the form element's value.

- Example :

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

const ControlledComponent = () => {
 const [inputValue, setInputValue] = useState('');

 const handleChange = (event) => {
  setInputValue(event.target.value);
  console.log(inputValue);
 };

 return (
  <input
   type="text"
   value={inputValue}
   onChange={handleChange}
  />
 );
};
```

⇨ **Uncontrolled Components :**

- An uncontrolled component is a form element where React doesn't directly control the value. Instead, you rely on the DOM to maintain the state of the input. You can still access the current value of the input element using a ref or other DOM methods.

- Example :

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

```
const UncontrolledComponent = () => {
  const inputRef = useRef(null);

  const handleButtonClick = () => {
    const value = inputRef.current.value;
    console.log('Input value:', value);
  };

  return (
    <>
      <input type="text" ref={inputRef} />
      <button onClick={handleButtonClick}>Get Value</button>
    </>
  );
};
```

⇨ **Difference between Controlled and Uncontrolled Components :**

- **State Management:**
  - Controlled components are managed by React.
  - Uncontrolled components manage their state internally. React doesn't have direct control over their state, and you have to use DOM references to access their values.
- **Data Flow:**
  - Controlled components have a unidirectional data flow.
  - Uncontrolled components don't follow a strict data flow pattern like controlled components. Data is accessed directly from the DOM when needed.
- **Validation and Synchronization:**
  - Controlled components allow for easy validation and synchronization of input values since React is in control of the data.

- Uncontrolled components might require extra validation checks and are generally less suitable for synchronized form validations.

12) What is Props in React?

⇨ **Props** stand for **properties**, Props are arguments passed into React components. it is a process for **passing data from a parent component to a child component**. Props allow you to **pass information in down or child component**. **child can access and use that data.**

⇨ Example :

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

function ParentComponent() {
  const name = Husen;
  const age = 20;

  return (
    <div>
      <ChildComponent name={name} age={age} />
    </div>
  );
}

export default ParentComponent;

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

function ChildComponent(props) {
```

```
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}

export default ChildComponent;
```

13) explain react state and props?

⇨ **Props :**

- Props (short for properties) are a way to pass data from a parent component to a child component.
- Props are read-only, meaning that the child component cannot modify the props it receives from the parent. They are immutable.
- Props enable communication and data sharing between components in a React application.

⇨ **State :**

- State, is like a memory for a component. It's used to manage and store data.
- state is managed within the component and can be modified by the component itself.
- When the state of a component changes, React will automatically re-render the component, updating the user interface to reflect the new state.

14) Explain about types of side effects in React component.

⇨ There are two types of side effects in React component. They are:

- **Effects without Cleanup:** This side effect will be used in useEffect which **does not restrict the browser from screen update**. It also improves the responsiveness of an application. A few common examples are network requests, Logging, manual DOM mutations, etc.

- **Effects with Cleanup:** Some of the **Hook effects will require the cleanup after updating of DOM is done**. For example, if you want to set up an external data source subscription, it requires cleaning up the memory else there might be a problem of memory leak. It is a known fact that React will carry out the cleanup of memory when the unmounting of components happens. But the effects will run for each render() method rather than for any specific method. Thus we can say that, before execution of the effects succeeding time the React will also cleanup effects from the preceding render.

15) What is Props Drilling in React?

⇨ Prop drilling is the process of passing data from one component via several interconnected components to the component that needs it.

⇨ Prop drilling in React refers to the process of passing props down through multiple levels of nested components, even if some intermediate components do not directly use those props.

⇨ Prop drilling is the act of **passing props through multiple layers of components**. (**Like:-** pass from A to B and B to C and C to D component)

⇨ **Disadvantage :**

- prop drilling is that the components that should otherwise be not aware of the data have access to the data.

16) What are Error Boundaries?

⇨ Error boundaries are React components that catch and handle errors that occur during rendering, in lifecycle methods, or in the constructor of any child component. They help prevent the entire React component tree from unmounting due to an error in a specific component.

17) What are React Hooks? Explain React Hooks. And also Must be Rules of React Hooks ?

⇨ **React Hooks :** Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

⇨ **Explain React Hooks :**

React Hooks cannot be used in class components. They let us write components **without class**.

React hooks were introduced in the **16.8 version** of React. Previously, **functional components were called stateless components**. Only **class components** were used for **state management and lifecycle methods**. The need to change a functional component to a class component, whenever state management or lifecycle methods were to be used, led to the development of Hooks.

⇨ **Rules of React Hooks :**

- React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.

- It is allowed to call the Hooks only from the React Function Components.

18) Why do React Hooks make use of refs?

⇨ Earlier, refs were only limited to class components but now it can also be accessible in function components through the useRef Hook in React.

⇨ The refs are used for :
- Managing focus, media playback, or text selection.
- Integrating with DOM libraries by third-party.
- Triggering the imperative animations.

19) What are Custom Hooks?

⇨ A Custom Hook is a function in Javascript whose name begins with 'use' and which calls other hooks. It is a part of React v16.8 hook update and permits you for reusing the stateful logic without any need for component hierarchy restructuring.

⇨ In almost all of the cases, custom hooks are considered to be sufficient for replacing render props and HoCs (Higher-Order components) and reducing the amount of nesting required.

⇨ Custom Hooks will allow you for avoiding multiple layers of abstraction or wrapper hell that might come along with Render Props and HoCs.

⇨ The **disadvantage** of Custom Hooks is it cannot be used inside of the classes.

Advance React Js Interview Question

20) Explain Strict Mode in React?

⇨ StrictMode is a tool added in version 16.3 of **React to highlight potential problems in an application**. It performs additional checks on the application.

⇨ To enable StrictMode, <React.StrictMode> tags need to be added inside the application:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
const rootElement = document.getElementById("root");
ReactDOM.render(
<React.StrictMode>
  <App />
</React.StrictMode>,
rootElement
);
```

⇨ StrictMode currently helps with the following issues:

- **Identifying components with unsafe lifecycle methods:**
  - StrictMode helps in providing us with a warning if any of the class components use an unsafe lifecycle method.

- **Warning about the usage of legacy string API:**
  - If one is using an older version of React, callback ref is the recommended way to manage refs instead of using the string refs. StrictMode gives a warning if we are using string refs to manage refs.

- **Warning about the usage of findDOMNode:**
  - Previously, findDOMNode( ) method was used to search the tree of a DOM node. This method is deprecated in React. Hence, the StrictMode gives us a warning about the usage of this method.

- **Warning about the usage of legacy context API (because the API is error-prone).**

21) How to prevents re-renders in React?

⇨ **Reason for re-renders in React:**

- Re-rendering of a component and its child components occur when props or the state of the component has been changed.
- Re-rendering components that are not updated, affects the performance of an application.

⇨ **How to prevent re-rendering:**

- Consider the following components:

```
class Parent extends React.Component {
state = { messageDisplayed: false };
componentDidMount() {
 this.setState({ messageDisplayed: true });
}
render() {
 console.log("Parent is getting rendered");
 return (
  <div className="App">
   <Message />
  </div>
 );
}
}
class Message extends React.Component {
constructor(props) {
 super(props);
 this.state = { message: "Hello, this is vivek" };
}
render() {
 console.log("Message is getting rendered");
 return (
  <div>
   <p>{this.state.message}</p>
  </div>
 );
}
}
```

- The Parent component is the parent component and the Message is the child component. Any change in the parent component will

lead to re-rendering of the child component as well. To prevent the re-rendering of child components, we use the **shouldComponentUpdate( ) method**:

Note - Use **shouldComponentUpdate( ) method only when you are sure that it's a **static component**.

```
class Message extends React.Component {
constructor(props) {
  super(props);
  this.state = { message: "Hello, this is vivek" };
}
shouldComponentUpdate() {
  console.log("Does not get rendered");
  return false;
}
render() {
  console.log("Message is getting rendered");
  return (
    <div>
      <p>{this.state.message}</p>
    </div>
  );
}
}
```

As one can see in the code above, we have returned **false** from the **shouldComponentUpdate( )** method, which prevents the child component from re-rendering.

22) What are the different ways to style a React component?

⇨ There are many different ways through which one can style a React component. Some of the ways are :

- **Inline Styling:** We can directly style an element using inline style attributes. Make sure the value of style is a JavaScript object:

```
class RandomComponent extends React.Component {
 render() {
  return (
    <div>
      <h3 style={{ color: "Yellow" }}>This is a heading</h3>
      <p style={{ fontSize: "32px" }}>This is a paragraph</p>
    </div>
  );
 }
}
```

- **Using JavaScript object:** We can create a separate JavaScript object and set the desired style properties. This object can be used as the value of the inline style attribute.

```
class RandomComponent extends React.Component {

 paragraphStyles = {

  color: "Red",

  fontSize: "32px"

 };

 headingStyles = {

  color: "blue",

  fontSize: "48px"

 };
```

```
 render() {
  return (
    <div>
      <h3 style={this.headingStyles}>This is a heading</h3>
      <p style={this.paragraphStyles}>This is a paragraph</p>
    </div>
  );
 }
}
```

- **CSS Stylesheet:** We can create a separate CSS file and write all the styles for the component inside that file. This file needs to be imported inside the component file.

```
import './RandomComponent.css';

class RandomComponent extends React.Component {
 render() {
  return (
    <div>
      <h3 className="heading">This is a heading</h3>
      <p className="paragraph">This is a paragraph</p>
    </div>
  );
 }
}
```

- **CSS Modules:** We can create a separate CSS module and import this module inside our component. Create a file with ".module.css"' extension, styles.module.css:

```
.paragraph{
 color:"red";
 border:1px solid black;
}
```

We can import this file inside the component and use it:

```
import styles from  './styles.module.css';

class RandomComponent extends React.Component {
 render() {
  return (
    <div>
      <h3 className="heading">This is a heading</h3>
      <p className={styles.paragraph} >This is a paragraph</p>
    </div>
  );
 }
}
```

23) Name a few techniques to optimize React app performance.

⇨ There are many ways through which one can optimize the performance of a React app, let's have a look at some of them:

- **Using useMemo( )** –
  - It is a React hook that is used for caching CPU-Expensive functions.

- Sometimes in a React app, a CPU-Expensive function gets called repeatedly due to re-renders of a component, which can lead to slow rendering.
- useMemo( ) hook can be used to cache such functions. By using **useMemo( ),** the CPU-Expensive function gets called only when it is needed.

- **Using React.PureComponent –**
  - It is a base component class that checks the state and props of a component to know whether the component should be updated.
  - Instead of using the simple React.Component, we can use **React.PureComponent to reduce the re-renders of a component unnecessarily.**

- **Maintaining State Colocation –**
  - This is a process of moving the state as close to where you need it as possible.
  - Sometimes in React app, we have a lot of unnecessary states inside the parent component which makes the code less readable and harder to maintain. Not to forget, having many states inside a single component leads to unnecessary re-renders for the component.
  - It is better to shift states which are less valuable to the parent component, to a separate component.

- **Lazy Loading –**
  - It is a technique used to reduce the load time of a React app. Lazy loading helps reduce the risk of web app performances to a minimum.

24) How to pass data between react components?

⇨ ***Parent Component to Child Component (using props)***

Consider the following Parent Component:

```
import ChildComponent from "./Child";
  function ParentComponent(props) {
   let [counter, setCounter] = useState(0);

   let increment = () => setCounter(++counter);

   return (
     <div>
       <button onClick={increment}>Increment Counter</button>
       <ChildComponent counterValue={counter} />
     </div>
   );
  }
```

As one can see in the code above, we are rendering the child component inside the parent component, by providing a prop called counterValue. The value of the counter is being passed from the parent to the child component.

We can use the data passed by the parent component in the following way:

```
function ChildComponent(props) {
return (
```

```
  <div>
    <p>Value of counter: {props.counterValue}</p>
  </div>
);
}
```

We use the props.counterValue to display the data passed on by the parent component.

⇨ ***Child Component to Parent Component (using callbacks)***

This one is a bit tricky. We follow the steps below:

- How to pass data between react components?Create a callback in the parent component which takes in the data needed as a parameter.
- Pass this callback as a prop to the child component.
- Send data from the child component using the callback.

We are considering the same example above but in this case, we are going to pass the updated counterValue from child to parent.

Step1 and Step2: Create a callback in the parent component, pass this callback as a prop.

```
function ParentComponent(props) {
let [counter, setCounter] = useState(0);
let callback = valueFromChild => setCounter(valueFromChild);
return (
  <div>
    <p>Value of counter: {counter}</p>
    <ChildComponent callbackFunc={callback} counterValue={counter} />
  </div>
);
```

```
}
```

As one can see in the code above, we created a function called callback which takes in the data received from the child component as a parameter.

Next, we passed the function callback as a prop to the child component.

Step3: Pass data from the child to the parent component.

```
function ChildComponent(props) {
let childCounterValue = props.counterValue;
return (
  <div>
    <button onClick={() => props.callbackFunc(++childCounterValue)}>
      Increment Counter
    </button>
  </div>
);
}
```

In the code above, we have used the props.counterValue and set it to a variable called childCounterValue.

Next, on button click, we pass the incremented childCounterValue to the **props.callbackFunc**.

This way, we can pass data from the child to the parent component.

25) What are the different phases of the component lifecycle?

⇨ There are four different phases in the lifecycle of React component. They are:

1. **initialization**
2. **mounting**
3. **updating,**
4. **unmounting.**

1) Initialization : This is the phase in which the component is going to start and setting state and props. This is mostly used constructor for assign value of state.

```
class Initialize extends React.Component {
   constructor(props)
   {
   // Calling the constructor of
   super(props);
   // initialization process
   this.state = {
     date : new Date(),
     clickedStatus: false
    }}
```

2) Mounting : Mounting is the phase in which our React component mounts on the DOM (i.e., is created and inserted into the DOM) means it is convert from jsx to javascript and insert in webpage.

There are two type :

- **componentWillMount() :** This method is called just before a component mounts on the DOM or the render method is called. After this method, the component gets mounted.

- **componentDidMount()** : This method is called after the component gets mounted on the DOM. Like componentWillMount, We can make API calls and update the state with the API response when this method is called.

Example :

```
class LifeCycle extends React.Component {
 componentWillMount() {
    console.log('Component will mount!')
 }
 componentDidMount() {
    console.log('Component did mount!')
    this.getList();
 }
 getList=()=>{
  /*** method to make api call***
 }
 render() {
    return (
      <div>
        <h3>Hello mounting methods!</h3>
      </div>
); }}
```

3) Updating : In this phase, the data of the component (state & props) updates in response to user events like clicking, typing and so on. This results in the re-rendering of the component.

- shouldComponentUpdate()
- componentWillUpdate()
- ComponentDidUpdate()

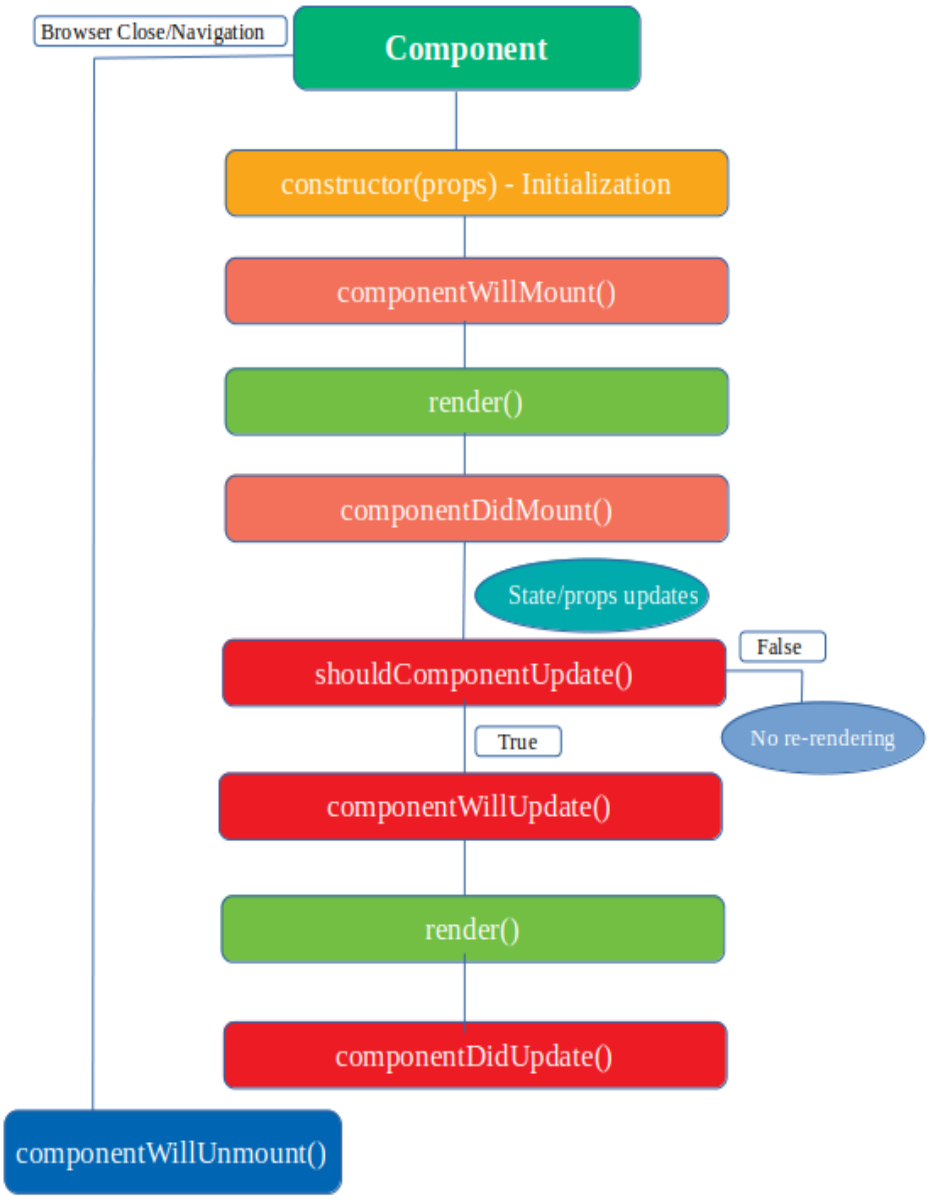
Example :

```
class LifeCycle extends React.Component {
    constructor(props)
    {
     super(props);
      this.state = {
        date : new Date(),
        clickedStatus: false,
        list:[]
      };
    }
    componentWillMount() {
       console.log('Component will mount!')
     }
    componentDidMount() {
       console.log('Component did mount!')
       this.getList();
     }
    getList=()=>{
     /*** method to make api call***
     fetch('https://api.mydomain.com')
       .then(response => response.json())
       .then(data => this.setState({ list:data }));
    }
     shouldComponentUpdate(nextProps, nextState){
      return this.state.list!==nextState.list
     }
     componentWillUpdate(nextProps, nextState) {
       console.log('Component will update!');
```

```
    }
    componentDidUpdate(prevProps, prevState) {
      console.log('Component did update!')
    }
    render() {
      return (
        <div>
          <h3>Hello Mounting Lifecycle Methods!</h3>
        </div>
)}}
```

4) Unmounting : This is the last phase in the component's lifecycle. As the name clearly suggests, the component gets unmounted from the DOM in this phase

- **componentWillUnmount() :** This method is called before the unmounting of the component takes place. Before the removal of the component from the DOM, 'componentWillUnMount' executes. This method denotes the end of the component's lifecycle.



26) What is Higher-Order Components?

⇨ A Higher-order component is a function that takes a component and returns a new component.

Higher-Order Component (HOC) is a pattern that allows you to **enhance** the behavior **or add additional features to a component**. HOCs are functions that **take a component as an argument and return a new component** with extended capabilities. They are commonly used for **code reuse, abstracting logic, and adding cross-cutting** concerns to components.

⇨ Example :

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

// Higher-Order Component (HOC) function
const withClickCounter = (WrappedComponent) => {
 const WithClickCounter = (props) => {
  const [clickCount, setClickCount] = useState(0);

  const handleIncrementClick = () => {
   setClickCount((prevCount) => prevCount + 1);
  };

  return (
   <WrappedComponent
    {...props}
    clickCount={clickCount}
    onIncrementClick={handleIncrementClick}
```

```
      />
    );
  };

  return WithClickCounter;
};

// Our regular component that we want to enhance with the click counter
feature
const ClickableComponent = ({ onClick, clickCount }) => {
  return (
    <div>
      <button onClick={onClick}>Click Me</button>
      <p>Click Count: {clickCount}</p>
    </div>
  );
};

// Using the HOC to enhance our ClickableComponent
const ClickableComponentWithClickCounter =
withClickCounter(ClickableComponent);

// Usage in a parent component or container
const App = () => {
  return (
    <div>
      <h1>Higher-Order Component Example</h1>
      <ClickableComponentWithClickCounter />
    </div>
  );
};

export default App;
```

⇨ In this example, **withClickCounter** is the HOC that takes **ClickableComponent** as an argument and returns an enhanced version of the component (**WithClickCounter**). The enhanced component tracks the number of times it's clicked and displays that count using the **clickCount** prop.

⇨ By using HOCs, we can keep the click-counting logic separate from the **ClickableComponent**, making it easier to manage and reuse in other components. Additionally, if you want to use this click-counting feature in other components, you can simply apply the **withClickCounter** HOC to them as well.

27) What are the lifecycle methods of React?

⇨ React lifecycle hooks will have the methods that will be automatically called at different phases in the component lifecycle and thus it provides good control over what happens at the invoked point. It provides the power to effectively control and manipulate what goes on throughout the component lifecycle.

⇨ For example, if you are developing the YouTube application, then the application will make use of a network for buffering the videos and it consumes the power of the battery (assume only these two). After playing the video if the user switches to any other application, then you should make sure that the resources like network and battery are being used most efficiently. You can stop or pause the video buffering which in turn stops the battery and network usage when the user switches to another application after video play.

⇨ **So we can say that the developer will be able to produce a quality application with the help of lifecycle methods** and it also helps

developers to make sure to plan what and how to do it at different points of birth, growth, or death of user interfaces.

⇨ The various lifecycle methods are:

- **constructor():** This method will be called when the component is initiated before anything has been done. It helps to set up the initial state and initial values.
- **getDerivedStateFromProps():** This method will be called just before element(s) rendering in the DOM. It helps to set up the state object depending on the initial props. The getDerivedStateFromProps() method will have a state as an argument and it returns an object that made changes to the state. This will be the first method to be called on an updating of a component.

- **render():** This method will output or re-render the HTML to the DOM with new changes. The render() method is an essential method and will be called always while the remaining methods are optional and will be called only if they are defined.

- **componentDidMount():** This method will be called after the rendering of the component. Using this method, you can run statements that need the component to be already kept in the DOM.

- **shouldComponentUpdate():** The Boolean value will be returned by this method which will specify whether React should proceed further with the rendering or not. The default value for this method will be True.

- **getSnapshotBeforeUpdate():** This method will provide access for the props as well as for the state before the update. It is possible to check the previously present value before the update, even after the update.

- **componentDidUpdate():** This method will be called after the component has been updated in the DOM.

- **componentWillUnmount():** This method will be called when the component removal from the DOM is about to happen.

28) Does React Hook work with static typing?

⇨ Static typing refers to the process of code check during the time of compilation for ensuring all variables will be statically typed. React Hooks are functions that are designed to make sure about all attributes must be statically typed. For enforcing stricter static typing within our code, we can make use of the React API with custom Hooks.

29) Explain about types of Hooks in React.

⇨ There are two types of Hooks in React. They are:

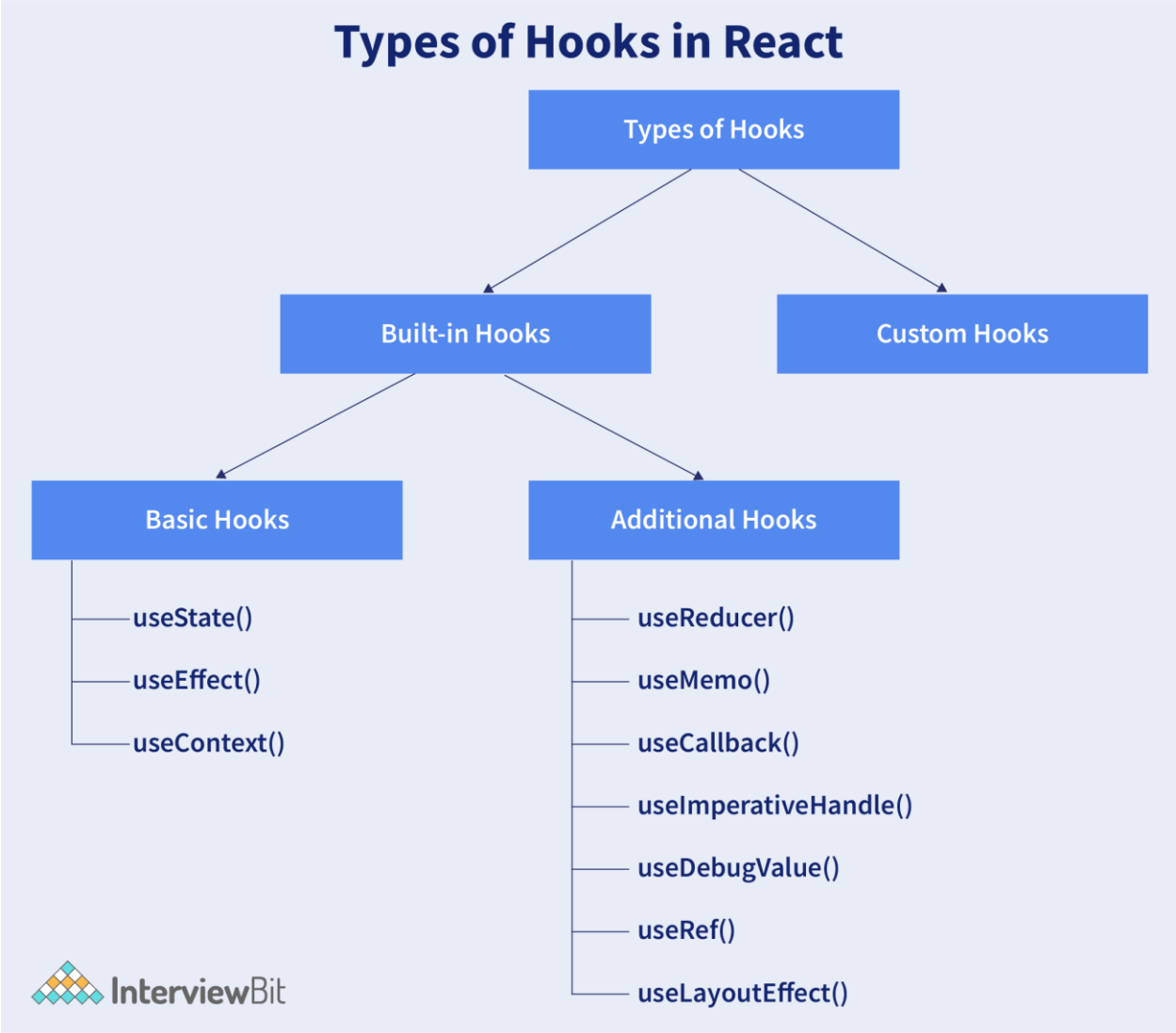
1) **Built-in Hooks:** The built-in Hooks are divided into 2 parts as given below:

- **Basic Hooks :**
  - **useState():** This functional component is used to set and retrieve the state.

- **useEffect():** It enables for performing the side effects in the functional components.
- **useContext():** It is used for creating common data that is to be accessed by the components hierarchy without having to pass the props down to each level.

- **Additional Hooks:**
  - **useReducer() :** It is used when there is a complex state logic that is having several sub-values or when the upcoming state is dependent on the previous state. It will also enable you to optimization of component performance that will trigger deeper updates as it is permitted to pass the dispatch down instead of callbacks.
  - **useMemo() :** This will be used for recomputing the memoized value when there is a change in one of the dependencies. This optimization will help for avoiding expensive calculations on each render.
  - **useCallback() :** This is useful while passing callbacks into the optimized child components and depends on the equality of reference for the prevention of unneeded renders.
  - **useImperativeHandle():** It will enable modifying the instance that will be passed with the ref object.
  - **useDebugValue():** It is used for displaying a label for custom hooks in React DevTools.
  - **useRef() :** It will permit creating a reference to the DOM element directly within the functional component.
  - **useLayoutEffect():** It is used for the reading layout from the DOM and re-rendering synchronously.

2) **Custom Hooks:** A custom Hook is basically a function of JavaScript. The Custom Hook working is similar to a regular function. The “use” at the beginning of the Custom Hook Name is required for React to understand that this is a custom Hook and also it will describe that this specific function follows the rules of Hooks. Moreover, developing custom Hooks will enable you for extracting component logic from within reusable functions.



30) Differentiate React Hook VS Classes.

React Hooks	Classes
It is used in functional components of React.	It is used in class-based components of React.
It will not require a declaration of any kind of constructor.	It is necessary to declare the constructor inside the class component.
It does not require the use of `this` keyword in state declaration or modification.	Keyword `this` will be used in state declaration (`this.state`) and in modification (`this.setState()`).
It is easier to use because of the `useState` functionality.	No specific function is available for helping us to access the state and its corresponding setState variable.
React Hooks can be helpful in implementing Redux and context API.	Because of the long setup of state declarations, class states are generally not preferred.

31) How does the performance of using Hooks will differ in comparison with the classes?

- React Hooks reduce lot of overheads like : object create, building of event etc,
- Hooks avoid nesting and rendering props, resulting in reduced React processing overhead.
- using Hooks in React instead of Higher Order Components (HOCs) can lead to more concise component structures and improved performance

32) Do Hooks cover all the functionalities provided by the classes?

⇨ There are no Hook equivalents for the following methods that are not introduced in Hooks yet:

- getSnapshotBeforeUpdate()

- getDerivedStateFromError()
- componentDidCatch()

33) What is React Router?

⇨ React Router refers to the standard library used for routing in React. It allow us for building a single-page web application in React. we can nevigate without refreshing page

The **major components** of React Router are given below:

- **BrowserRouter** : it enables client-side routing in a React application, allowing you to implement navigation and handle different URL paths with dynamic rendering of components.

- **Routes :** refer to a collection of route configurations, defining how different URLs map to components

- **Route** : it is specifies a URL path and render component when the path matches

- **Link :** It enables declarative navigation, preventing full page reloads and providing a smooth user experience

34) Can React Hook replaces Redux?

⇨ Yes we can say, React Hooks can replace Redux for state management in simpler applications by using local state and context. However, Redux is a first choice for complex global state management.

35) Explain conditional rendering in React ?

⇨ Conditional rendering used to the dynamic output of user interface markups based on a condition state. It works in the same as JavaScript conditions.

⇨ Using conditional rendering, API data rendering, hide or show elements, and so on.

⇨ We can use if-else and ternary operator for conditional rendering in react.

36) How to create a switching component for displaying different pages?

- Define an array of page components, each representing a different page.
- Use state or props to track the currently selected page index.
- Render the selected page component based on the index.

Example :

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

const Page1 = () => <div>Page 1 content</div>;

const Page2 = () => <div>Page 2 content</div>;

const Page3 = () => <div>Page 3 content</div>;


const SwitchingComponent = () => {

  const [currentPage, setCurrentPage] = useState(0);
```

```
const pages = [Page1, Page2, Page3];

const handlePageChange = (pageIndex) => {
  setCurrentPage(pageIndex);
};

const CurrentPageComponent = pages[currentPage];

return (
  <div>
    <button onClick={() => handlePageChange(0)}>Page 1</button>
    <button onClick={() => handlePageChange(1)}>Page 2</button>
    <button onClick={() => handlePageChange(2)}>Page 3</button>
    <div>
      <CurrentPageComponent />
    </div>
  </div>
) };
export default SwitchingComponent;
```

37) How to re-render the view when the browser is resized?

⇨ In functional component we can resize browser using **useEffect** method .

```
useEffect(() => {
const handleResize = () => {
        // Your logic to handle resizing
        // This will re-render the view when the browser is resized
};
        window.addEventListener('resize', handleResize);
        return () => window.removeEventListener('resize', handleResize);
}, []);
```

⇨ In class component we can use **ComponentDidMount()** and **componentWillUnmount()**

```
componentDidMount() {
        window.addEventListener('resize', this.handleResize);
 }
 componentWillUnmount() {
         window.removeEventListener('resize', this.handleResize);
 }
handleResize = () => {
         // Your logic to handle resizing
         // This will re-render the view when the browser is resized
};
```

38) How to pass data between sibling components using React router?

⇨ Passing data between sibling components of React is possible using React Router with the help of **history.push** and **match.params (this method is pending)**.

⇨ Here's an **example** of how you can pass data between sibling components using React Router:

⇨ Assuming you have two sibling components: **ComponentA** and **ComponentB**.

1) Setting up the Routes:

⇨ In your main App component, set up the routes using **react-router-dom:**

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import ComponentA from './ComponentA';
import ComponentB from './ComponentB';

function App() {
  return (
    <Router>
      <Route path="/componentA" component={ComponentA} />
      <Route path="/componentB" component={ComponentB} />
    </Router>
  );
}

export default App;
```

2) Passing Data:

⇨ Inside **ComponentA**, you want to navigate to **ComponentB** and pass some data along. You can use the **history** object provided by React Router to achieve this:

```
import React from 'react';
import { useHistory } from 'react-router-dom';

function ComponentA() {
 const history = useHistory();

 const handleClick = () => {
  const dataToPass = 'Hello from Component A';
  history.push({
   pathname: '/componentB',
   state: { data: dataToPass },
  });
 };

 return (
  <div>
   <button onClick={handleClick}>Go to Component B</button>
  </div>
 );
}

export default ComponentA;
```

3) Receiving Data:

⇨ Inside **ComponentB**, you can access the passed data using the **location** object:

```
import React from 'react';

function ComponentB({ location }) {

        const passedData = location.state.data;

return (

<div>

        <p>Data received from Component A: {passedData}</p>

</div>

);

}

export default ComponentB;
```

39) How to perform automatic redirect after login?

⇨ The react-router package will provide the component **<Redirect>** in React Router. Rendering of a **<Redirect>** component will navigate to a newer location. In the history stack, the current location will be overridden by the new location just like the server-side redirects.

```
import React, { Component } from 'react'

import { Redirect } from 'react-router'

export default class LoginDemoComponent extends Component {

 render() {

        if (this.state.isLoggedIn === true) {
```

```
            return <Redirect to="/your/redirect/page" />
        } else {
            return <div>{'Please complete login'}</div>
        }
    }
}
```

Extra React Js Interview Question

40) What is Pure Function ?

⇨ A Pure Function is a function (a block of code) that always returns the same result if the same arguments are passed and has no side effects. It means it does not modify external data or variables outside its scope.

⇨ Pure function makes the code easily readable, and testable and increases the performance.

⇨ Example :

```
// Pure function (no side effects)
function addNumbers(a, b) {
return a + b;
}
```

```
// Impure function (with side effects)

let result = 0;

function addNumbersImpure(a, b) {

result = a + b; // Modifying external variable (side effect)

return result;

}
```

41) What is Modularity in React ?

⇨ Modularity in react refers to a complex application into a smaller, self-contained and reusable components.

⇨ **Benefits :**

- Reusability : (promoting a "write once, use many times" approach)
- Sepration of Concerns
- Collaboration
- Scalability
- Maintainability
- Code Organization

⇨ **Best Example of Modularity :**

```
// BookList.js

import React from 'react';
import BookItem from './BookItem';

const BookList = ({ books }) => {
  return (
```

```
    <ul>
     {books.map((book) => (
       <li key={book.id}>
         <BookItem book={book} />
       </li>
     ))}
    </ul>
  );
};

export default BookList;


// BookItem.js

import React from 'react';

const BookItem = ({ book }) => {
  return (
    <div>
      <h2>{book.title}</h2>
      <p>Author: {book.author}</p>
      <p>Published: {book.publishedYear}</p>
    </div>
  );
};

export default BookItem;


// App.js

import React from 'react';
import BookList from './BookList';
```

```
const booksData = [
  {
    id: 1,
    title: 'Book 1',
    author: 'Author A',
    publishedYear: 2021,
  },
  {
    id: 2,
    title: 'Book 2',
    author: 'Author B',
    publishedYear: 2019,
  },
  // More book data...
];

const App = () => {
  return (
    <div>
      <h1>List of Books</h1>
      <BookList books={booksData} />
    </div>
  );
};

export default App;
```

42) Why React is Faster?

- Virtual DOM
- Efficient DOM Updates
- Component Rendering Optimization
- Fast Diffing Algorithm (DIFF Algorithm)
- Virtual DOM Diffing

- Asyncronous Rendering
- Code Splitting and Lazy Loading
- Server-Side Rendering (SSR)

43) Difference between map() and filter() function?

⇨ **Map() :**

- map() is used to **transform each item of an array** and **return a new array** with the transformed items. It is like a "transformation" function.

- The **length** of the **new array** will be the **same as the original array**.

- Example :

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

const doubledNumbers = numbers.map((number) => number * 2);

// Result: [2, 4, 6, 8, 10]
```

⇨ **Filter() :**

- **filter()** is used to **select** and **return a new array containing only the items** that **meet a specific condition**. It is like a "filtering" function.

- The **length** of the **new array** can be **smaller than the original array** if **some items** do not meet the condition specified in the function.

- Example :

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

const evenNumbers = numbers.filter((number) => number % 2 === 0);

// Result: [2, 4]
```

44) What is Conditional Rendering in React?

⇨ Conditional rendering in React refers to render different content or components based on certain conditions of the application.

⇨ It allows developers to control what gets displayed in the user interface dynamically, depending on the values of variables, props, or state.

⇨ Example : // conditional rendering using if-else statement

```
import React from 'react';

const ConditionalComponent = ({ isLoggedIn }) => {
 if (isLoggedIn) {
  return <div>Welcome, User!</div>;
 } else {
  return <div>Please log in to continue.</div>;
 }
```

```
};

export default ConditionalComponent;
```

⇨ **T**here are **different ways** to achieve **conditional rendering** in React:

- **If-else Statement :** (if(isLoggedin) return “true” else return “false”))
- **Ternary Operator :** (isLoggedin ? “true” : “false”)
- **Logical AND (&&) Operator :** (isLoggedin && “true”)
- **Rendering Null or Empty Fragment :** (isLoggedin ? “true” : null);

45) What is difference between package.json and package-lock.json file in React?

⇨ package.json file store the **approximate** version
⇨ package-lock.json file store the **axact** version

46) What is NPM?

⇨ NPM stands for **"Node Package Manager."** And it’s multiple name. It is a package manager for JavaScript and is widely used in the Node.js and **front-end development ecosystems**. NPM allows developers to **easily install, manage, and share reusable code** packages (also known as modules or libraries) that can be used in their projects.

47) Difference between ~ and ^ symbol in React?

⇨ **Major, Minor or Patch :**

⇨ **Major :** 2.0.0 (first is major version)
⇨ **Minor :** 2.1.0 (second is minor version)
⇨ **Patch :** 2.1.1 (third is patch version)

⇨ **Tilde (~):** Allows **updates** to the **latest patch version** while keeping the major and minor versions fixed.

⇨ Example of Tilde :

- If you have a dependency listed as **"react": "~17.0.2"**, it means that npm will update to the latest patch version of React 17 (e.g., 17.0.3, 17.0.4, etc.) but will not update to React 18 or any other major version.

⇨ **Caret (^):** Allows **updates** to the **latest minor or patch version** while keeping the major version fixed.

⇨ Example of Caret :

- If you have a dependency listed as **"react": "^17.0.2"**, it means that npm will update to the latest minor or patch version of React 17 (e.g., 17.1.0, 17.2.1, etc.) but will not update to React 18 or any other major version.

48) What is difference between custom Components and Hooks in React?

⇨ **Custom hooks** were **invented to share the logic** alone whereas **Component is used more for displaying the UI (JSX).** I would use

custom hook when there is no need for returning JSX and also when I just need to share logic like I normally do with regular functions.

49) What is minification ?

⇨ **Minification**, also known as **minimization**, is the process of **removing all unnecessary characters from JavaScript source code** without altering its functionality.

⇨ Example : here is a block of code before and after minification:

Before minification: five lines of code

```
function sayHi(name)
{
        console.log(“my name is : “ + name);
}
sayHi(“Husen”);
```

After minification: two lines of code

```
function sayHi(name){console.log(“my name is : “ + name)}sayHi(“Husen”);
```

50) what is browser list in package.json in react js ?

⇨ **Browserslist** is a tool that allows specifying **which browsers should be supported in your frontend app by specifying "queries" in a**

config file. It's used by frameworks/libraries such as React, Angular and Vue, but it's not limited to them.

⇨ **Why would we want it?**

- During development we want to use the latest javascript features (e.g ES6) as it makes our jobs easier, leads to cleaner code, possibly better performance.

51) What is React Element ?

⇨ Elements are the smallest building blocks of React apps.
⇨ An element describes what you want to see on the screen.
⇨ Like : **const element = <h1>Hello, world</h1>**

52) What is React Component ?

⇨ React Component is like a **building block or small piece of web page or code.**