**300+ REACT INTERVIEW QUESTIONS**

<https://dev.to/sakhnyuk/300-react-interview-questions-2ko4>

<https://github.com/sudheerj/reactjs-interview-questions>

<https://github.com/sudheerj/reactjs-interview-questions?tab=readme-ov-file#What-is-the-history-behind-React-evolution>

**REACT HOOKS**

<https://react.dev/reference/react/hooks>

**What is React?**

It is an open-source front-end JavaScript library that is used for building interactive and complex user interfaces, especially for single-page applications. It is used for handling view layer for web and mobile apps based on components . React.js was released for use in 2015 and since then it has become one of the most trusted and used technologies of recent time. It has built one of the largest developer communities around itself.

**Why react is library and not framework**

React is considered a library rather than a framework because it primarily focuses on the "View" aspect of an application, providing UI rendering and state management. Unlike frameworks, React offers flexibility and doesn't enforce a specific structure or set of tools for an entire application. It relies on third-party libraries for additional functionality like routing and state management, allowing developers to choose tools as needed. This modularity and flexibility give developers more control over their projects, which is a key characteristic of libraries.

**Difference between Library and framework.**

| **Aspect** | **Library** | **Framework** |
| --- | --- | --- |
| **Definition** | A collection of pre-written code for specific tasks. | A complete structure and set of tools for building an application. |
| **Control** | Developer controls the application flow; calls library functions. | Framework controls the application flow; calls developer's code. |
| **Flexibility** | More flexible; can be integrated with various other tools and libraries. | More restrictedted; enforces certain rules and patterns. |
| **Integration** | Used as needed; does not dictate application structure. | Provides a structured environment; dictates application architecture. |
| **Usage** | Developers decide where and when to use the library functions. | Developers follow the framework's guidelines and conventions. |
| **Examples** | jQuery, React, Lodash | Angular, Django, Ruby on Rails |

**How React works behind the scenes?**   
**1. Virtual DOM**: React creates a lightweight copy of the actual DOM, known as the virtual DOM. When a component's state changes, React updates this virtual DOM instead of the real DOM, making the process faster.  
**2. Diffing Algorithm**: React compares the new virtual DOM with the previous version using a diffing algorithm. This helps identify what has changed.  
**3. Reconciliation**: Once React knows what has changed, it updates only the specific parts of the actual DOM that need to be changed, rather than re-rendering the entire UI. This process is called reconciliation.  
**4. Component-Based Architecture**: React applications are built using components, which are independent, reusable pieces of UI. Each component can manage its own state and lifecycle, making the development process more modular and easier to manage.  
**5. React Fiber**: React Fiber is the new reconciliation engine in React 16 and above. It breaks rendering work into small units and spreads it out over multiple frames, resulting in a smoother user experience, especially for animations and complex applications.  
**6. Hooks**: React hooks allow you to use state and other React features in functional components, providing more flexibility and a simpler API for managing state and side effects.

**List some of React.js’ features.**

* Instead of a real DOM, there is Virtual DOM
* In React.js, the data can be passed to the other parts of an application only in one direction. In other words, there is a unidirectional flow of data.
* **Component-Based Architecture**: React encourages building encapsulated components that manage their own state, making it easier to build complex UIs by composing components.
* **JSX (JavaScript XML)**: JSX is a syntax extension that allows mixing HTML with JavaScript. It simplifies writing and adding HTML in React.
* **Virtual DOM**: React uses a virtual DOM to optimize updates. It updates only the parts of the actual DOM that have changed, improving performance.

**What are the advantages of React.js?**

#### Virtual DOM for Improved Performance: React uses a virtual DOM to optimize updates and rendering, ensuring that only the necessary components are re-rendered, which improves performance.

#### Component-Based Architecture: React's component-based structure allows for reusable, maintainable, and testable code. Components can be nested, managed, and handled independently.

#### One-Way Data Binding: React uses one-way data binding, meaning that data flows in one direction, which makes the application easier to debug and more predictable.

#### JSX Syntax: React's JSX syntax allows for writing HTML-like code within JavaScript, making it more intuitive to create UI components.

**What are the disadvantages of React.js?**

#### Steep Learning Curve: While the basics of React can be picked up relatively quickly, mastering its ecosystem, including tools like Redux for state management, can be challenging.

#### Rapid Pace of Development: The React ecosystem evolves rapidly, which can make it difficult for developers to keep up with the latest best practices and tools.

#### Complex State Management: For larger applications, managing state can become complex. Although tools like Redux or Context API help, they add additional layers of complexity.

#### View Layer Only: React is focused only on the view layer of the application, so developers need to rely on other libraries and frameworks for full-fledged development, like routing (React Router) and form handling (Formik).

#### Performance Issues with Large Applications: While React's virtual DOM improves performance for most applications, very large and complex applications can still experience performance bottlenecks if not optimized properly.

**What is JSX?**

JSX, or JavaScript XML, is a syntax extension for JavaScript used in React to describe what the UI should look like. JSX allows developers to write HTML-like code within JavaScript, making it easier to create and visualize UI components. Although it looks like HTML, JSX is not a string or HTML; it gets transformed into JavaScript objects by tools like Babel.

While it resembles HTML, it is not valid JavaScript and cannot be understood by browsers directly. To make it work, JSX must be transformed into JavaScript code that browsers can understand. This transformation is usually done by a tool like Babel.

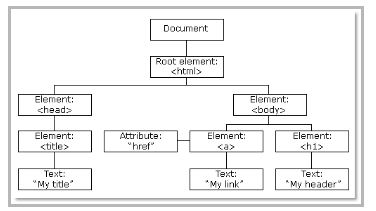
**Can web browsers read JSX directly?**

* Web browsers cannot read JSX directly. This is because they are built to only read regular JS objects and JSX is not a regular JavaScript object
* For a web browser to read a JSX file, the file needs to be transformed into a regular JavaScript object. For this, we use Babel

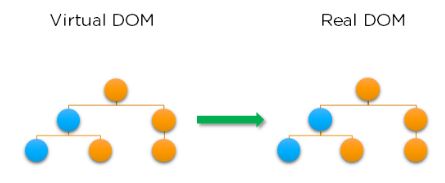


**What is the virtual DOM?**

DOM stands for Document Object Model. The DOM represents an HTML document with a logical tree structure. Each branch of the tree ends in a node, and each node contains objects.



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. The following are some of the most frequently asked react interview questions.



**Describe an event in React.js?**

An event is an action or occurrence that happens in the browser, triggered by various user interactions or by the browser itself. which the browser can detect and respond to. Events are used to trigger specific functionality or behavior within a web page. Common events include clicks, key presses, mouse movements, form submissions, and more.

When a user presses a key, clicks the mouse, or performs any action on the machine or when the machine itself triggers an action, these actions are registered as events in React.js.

* In React.js, we use camelCase to name events, instead of the lowercase in HTML
* In React.js, because of JSX, a function is passed as an event handler, instead of the string in HTML

**Is HTML used in React?**

No, it uses an HTML-in JavaScript syntax called JSX (JavaScript and XML) that converts HTML tags to React elements.

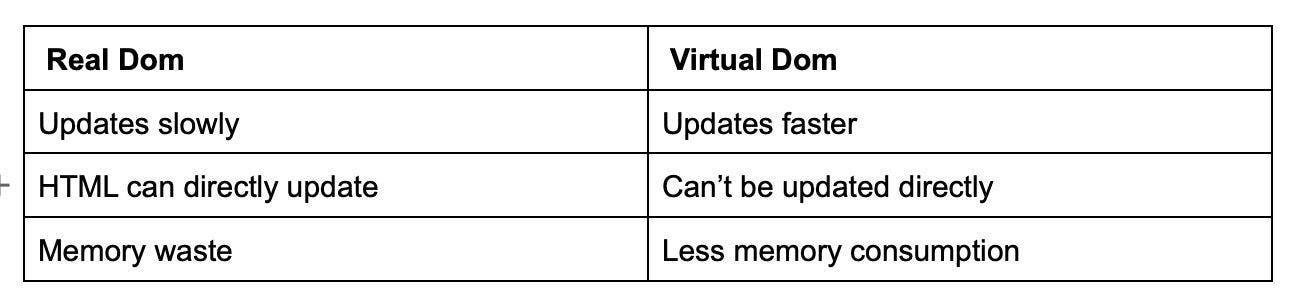
**What is the release date of React?** March 2013

**Can you tell two downsides of React?**

It is difficult to integrate with an MVC framework like Rails. Also, users need to have the knowledge about integrating user interface into MVC framework.

**Can you outline the differences between Real DOM and Virtual DOM?**

Real DOM and Virtual DOM differ in following ways:



**What is Redux in React?**

An open-source JavaScript library for front-end development and acts as a container for JavaScript applications that is used for the applications state management.

**What is state in react**

In React, "state" refers to a built-in object that is used to store property values that belong to a component. State is a key concept in React, as it allows components to create, update, and manage dynamic data that can change over time, leading to a re-render of the component and its children.

**Explain the ‘Store’ feature.**

It is a feature of Redux that allows the application to save the entire State at one place. This allows you to get regular updates from the store directly.

**What is a high order component in React?**

A Higher-Order Component (HOC) in React is an advanced pattern for reusing component logic. An HOC is a function that takes a component and returns a new component with additional props or behavior. This allows for code reuse, logic abstraction, and enhances component functionalities without modifying the original component.

What is the Presentational segment?

It is a component in React that just renders HTML and its only function is presentation markup. In React, the term "presentational component" refers to a type of component whose primary responsibility is to display data (the UI) and receive data through props. They are also known as "dumb components" or "pure components" because they do not manage state or handle complex logic; instead, they rely on parent components (often called container or smart components) to provide them with the data and behavior they need.

**Explain Props in ReactJS.**

In ReactJS, "props" (short for properties) are a mechanism for passing data from a parent component to a child component. Props are read-only and immutable within the child component, meaning that the child component cannot modify the props it receives. This ensures a unidirectional flow of data and helps maintain the predictability and simplicity of the application state.

**List the two types of React component.**

Two types of React component are as follows:

* Function component
* Class component

**Class Components**:

* Class components are ES6 classes that extend from React.Component.
* They use the render() method to return React elements that represent the UI.
* Class components can manage their own state using this.state and lifecycle methods like componentDidMount, componentDidUpdate, etc.

Example:

import React, { Component } from 'react';

class MyComponent extends Component {

render() {

return <h1>Hello, Class Component!</h1>;

}

}

export default MyComponent;

**Functional Components** (also known as Stateless or Pure Components):

* Functional components are JavaScript functions that accept props as arguments and return React elements.
* They are simpler and more lightweight than class components.
* Functional components do not manage state or have lifecycle methods (until the introduction of hooks in React).

Example:

import React from 'react';

const MyComponent = (props) => {

return <h1>Hello, Functional Component!</h1>;

};

export default MyComponent;

Here's a comparison of class components and functional components in React, summarizing their differences, advantages, disadvantages, and common use cases in a tabular format:

| **Feature** | **Class Components** | **Functional Components** |
| --- | --- | --- |
| **Definition** | ES6 classes that extend React.Component. | JavaScript functions that return JSX. |
| **State Management** | Can manage state using this.state and setState. | Can manage state using useState hook (with React hooks). |
| **Lifecycle Methods** | Has lifecycle methods (componentDidMount, componentDidUpdate, etc.). | No lifecycle methods (before hooks), but can use useEffect hook for side effects. |
| **Complexity** | More verbose with more boilerplate code. | Simplified syntax and less boilerplate code. |
| **Performance** | Slightly slower due to JavaScript classes. | Lightweight and optimized for performance. |
| **Reusability** | Less reusable due to tight coupling of state and lifecycle methods. | Highly reusable with custom hooks and functional composition. |
| **Integration** | Common in older React applications and with third-party libraries. | Modern approach in new React applications. |
| **Advantages** | - State management and lifecycle methods.<br>- Backward compatibility.<br>- Established pattern.<br>- Suitable for complex components. | - Simplicity and readability.<br>- Lightweight and optimized performance.<br>- Easier to test and maintain.<br>- Promotes functional programming principles. |
| **Disadvantages** | - Complexity and boilerplate.<br>- Steeper learning curve for beginners.<br>- Less reusable code.<br>- Performance considerations with large applications. | - Limited state management without hooks (pre-hooks).<br>- May require refactoring existing class components.<br>- Integrating with older codebases using class components. |
| **Use Cases** | - Components with complex state and lifecycle management.<br>- Integrating with legacy codebases.<br>- Third-party libraries requiring class components. | - Presentational components.<br>- Components focusing on UI rendering.<br>- Optimizing performance with lightweight components.<br>- Building reusable logic with custom hooks. |

**In Redux, what is an action?**

In Redux, an action is an object that contains information about an event that has occurred in an application.

**What are stateless components?**

Stateless components, also known as functional components in React, are JavaScript functions that accept props as input and return React elements to describe the UI. They do not manage state internally or use lifecycle methods. Instead, they are primarily focused on rendering UI based on the props they receive.

**What is React Router?**

React Router is a popular library for routing in React applications. It enables navigation and handling of different views or pages within a single-page application (SPA) by managing the URL and rendering the corresponding components based on the current URL.

**Explain React State.**

React State is an object used to store dynamic data and manage component behavior in React. It allows components to respond to user input, network responses, and other changes. When the state changes, React re-renders the component to reflect the new state. State is typically managed using the useState hook in functional components or the this.state and this.setState methods in class components.

**Can a State be updated in ReactJS?**

Yes, it can be updated directly or indirectly. To implement this one can use either this. setState function or the updater function.

**Explain Context.**

React Context provides a way to pass data through the component tree without having to pass props down manually at every level. It is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language.

**What is the use of Webpack?**

Webpack is a command line tool used to bundle different code files and assets into single large file.. It has been built over Node.js and helps minimize Javascript, CSS files, images, etc.

**What is Babel?**

It is an open-source JavaScript compiler that converts the latest JavaScript versions to old ones.

**How are ReactJS and React Native different?**

Where ReactJS is a front end open-source JavaScript library for UIs, React Native is an open-source mobile framework for platforms such as Android and iOS.

**What is the most advised way for naming components?**

By using reference rather than using displayName because it provides a clear and concise name for the component.

**Mention one difference between Props and State.**

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| **Definition** | Read-only attributes passed from parent | Mutable object that stores local data |
| **Mutability** | Immutable | Mutable |
| **Purpose** | Pass data and event handlers from parent to child | Manage and control component's internal data |
| **Access** | Via this.props in class components, function parameters in functional components | Via this.state and this.setState in class components, useState hook in functional components |
| **Usage** | <ChildComponent message="Hello, World!" /> | this.setState({ count: 1 }) (class), const [count, setCount] = useState(0) (functional) |
| **Lifecycle** | Passed down the component tree | Defined and managed within the component |
| **Trigger** | Does not trigger re-rendering of the parent component | Triggers re-rendering of the component when updated |
| **Control** | Controlled by parent component | Controlled by the component itself |

**What are pure components in ReactJS?**

Pure components are the components which render the same output for the same state and props. In function components, you can achieve these pure components through memoized React.memo() API wrapping around the component. This API prevents unnecessary re-renders by comparing the previous props and new props using shallow comparison. So it will be helpful for performance optimizations.

**What is ‘create-React-app’?**

It provides a development environment for creating React applications using the JavaScript features as it creats a basic setup for the application without requiring to configure everything. It is very helpful in building a single-page application in React.

**What are error boundaries?**

Error boundaries in React are components that catch JavaScript errors anywhere in their child component tree, log the error, and display a fallback UI instead of the crashed component tree. They are created by implementing componentDidCatch and getDerivedStateFromError lifecycle methods.

**Why will you use empty tags <> </>?**

Empty tags `<> </>` in React are used as a shorthand for creating fragments. Fragments allow you to group multiple children elements without adding extra nodes to the DOM. They are useful when you need to return multiple elements from a component's render method without wrapping them in a parent `<div>` or other container.

**Tell us about strict mode.**

Strict Mode is a development mode feature in React that helps identify and highlight potential problems in your application's code. It performs additional checks and warnings to improve code quality and maintainability during development. Key aspects of Strict Mode include:

**Identifying Unsafe Practices**: Strict Mode enables additional checks and warnings for potential unsafe practices, such as deprecated lifecycle methods, legacy string ref usage, and more.

**Detecting Legacy Features**: It helps detect and highlight the use of legacy features that are deprecated and may be removed in future versions of React.

**Encouraging Best Practices**: Strict Mode encourages the adoption of best practices by identifying issues like component side effects during rendering and improperly declared PropTypes.

**Isolation of Effects**: It isolates unexpected side effects by invoking certain lifecycle methods twice and ensuring that they are free from side effects.

**Usage in Development**: Strict Mode is intended for development use only and does not impact the production build. It allows developers to catch and fix problems early in the development process.

**What is React Fiber?**

React Fiber is a complete rewrite of React's core algorithm, designed to improve the way React manages the rendering, reconciliation, and scheduling of updates. It was introduced to address performance issues and enable new features like async rendering and suspense.

React Fiber was a significant internal refactor of React's reconciliation algorithm, not directly exposed to developers through API changes. It was gradually rolled out in React versions starting from React 16, improving performance and opening the door for future optimizations and features.

In essence, React Fiber represents React's commitment to performance, concurrency, and enhanced user experiences by rethinking how updates are prioritized and managed within the framework.

**Incremental Rendering**: Fiber allows React to split rendering work into chunks or "fibers," prioritizing and scheduling them accordingly. This enables smoother animations and more responsive user interfaces.

**Improved Concurrency**: It supports asynchronous rendering, allowing React to pause and resume work as needed, prioritizing high-priority updates (like user interactions) over less critical ones.

**Better Error Handling**: Fiber improves error boundaries and stack trace accuracy, making it easier to debug and handle errors in React applications.

**Prioritization and Scheduling**: It introduces a new scheduling algorithm that can prioritize different types of updates, ensuring that important tasks are handled promptly while less critical tasks are deferred.

**Support for Suspense**: Fiber enables the implementation of React Suspense, a mechanism for handling async operations such as data fetching and code splitting, with better user experience and loading state management.

**Can you differentiate between createElement and cloneElement?**

| **Feature** | **createElement** | **cloneElement** |
| --- | --- | --- |
| **Definition** | Function to create a React element | Function to clone and return a React element |
| **Primary Use** | Used to create new elements in JSX | Used to clone and modify existing elements |
| **Usage** | Creates a new React element from a component or HTML tag with props and children | Clones and returns a new React element with modified props and children from an existing element |
| **Props Handling** | Props are passed as an object | Existing element's props can be modified or extended |
| **Children Handling** | Children are passed as subsequent arguments or nested arrays | Existing element's children can be replaced or appended |
| **Component Type** | Can create elements for both functional and class components | Typically used with existing elements from render functions |
| **Return Value** | Returns a React element | Returns a cloned React element |
| **Direct Use Case** | Often used in JSX transformations under the hood | Used for modifying elements dynamically in React applications |

**Explain Lifting State Up in React.**

When multiple components need to share the same data, it is advised to lift the shared state up to their parent. This means when two child components share the same data from their parent, the state is lifted up to the parent instead of the child components.

**Explain stateful components.**

If the behavior of the component holds dependency on the state of the component, we term it as a stateful component.

**Tell me, how will you memoize a component in React?**

The useMemo hook in React is used for memoizing expensive calculations so that they are only recomputed when necessary. It takes a function and an array of dependencies, and it returns a memoized value. Here's how you can use useMemo:

### Memoizing a Value:

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

function ParentComponent() {

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

// Memoizing a value based on count

const memoizedValue = useMemo(() => {

// Expensive computation

return count \* 2;

}, [count]); // Dependency array: recompute when count changes

return (

<div>

<p>Count: {count}</p>

<p>Memoized Value: {memoizedValue}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

export default ParentComponent;

### Explanation:

* **useMemo Hook**: useMemo takes a function as its first argument, which performs the expensive computation, and an array of dependencies as its second argument. It memoizes the function and re-runs it only when one of the dependencies changes.
* **Memoized Value**: In this example, memoizedValue is recalculated only when count changes. Otherwise, React returns the memoized value from the previous render, optimizing performance.

### Memoizing a Component:

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

const ExpensiveComponent = ({ prop }) => {

console.log('Rendering ExpensiveComponent');

// Expensive computation based on prop

return <div>Computed value: {prop \* 2}</div>;

};

function ParentComponent() {

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

// Memoizing a component based on count

const memoizedComponent = useMemo(() => {

return <ExpensiveComponent prop={count} />;

}, [count]); // Dependency array: re-render when count changes

return (

<div>

<p>Count: {count}</p>

{memoizedComponent}

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

export default ParentComponent;

### Explanation:

* **Memoized Component**: memoizedComponent is a memoized instance of ExpensiveComponent. It will only be re-rendered when count changes, optimizing rendering performance.
* **Performance Benefit**: By memoizing the component with useMemo, React ensures that ExpensiveComponent is not re-rendered unnecessarily when its props remain the same.

### When to Use useMemo:

* Use useMemo when you need to optimize performance by memoizing expensive calculations or components that rely on props.
* It's particularly useful in scenarios where computations are computationally expensive or when you want to prevent unnecessary re-renders.

Memoization in React can be achieved using the React.memo higher-order component or the useMemo hook to optimize functional components. Here's a simple example using React.memo to memoize a functional component:

import React from 'react';

// Regular functional component

const MyComponent = ({ name }) => {

console.log('Rendering MyComponent');

return <div>Hello, {name}!</div>;

};

// Memoized component using React.memo

const MemoizedComponent = React.memo(MyComponent);

// Usage in parent component

function ParentComponent() {

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

const handleClick = () => {

setCount(count + 1);

};

return (

<div>

<button onClick={handleClick}>Increment</button>

<MemoizedComponent name="Alice" />

<p>Count: {count}</p>

</div>

);

}

export default ParentComponent;

### Explanation:

* **MyComponent**: This is a regular functional component that takes a name prop and renders a simple greeting message.
* **MemoizedComponent**: By wrapping MyComponent with React.memo, React will memoize the component based on its props. It will re-render only if the props (name) change.
* **ParentComponent**: Demonstrates the usage of MemoizedComponent within a parent component. Clicking the "Increment" button updates the count state, but MemoizedComponent will only re-render when its props (name) change.

### Benefits of Memoization:

* **Performance Optimization**: Memoization avoids unnecessary re-renders of components when their input props haven't changed, improving performance.
* **Simpler Code**: It allows developers to write straightforward components without worrying about unnecessary re-renders impacting performance.

**How to use production mode in React?**

It can be used through Webpack's DefinePlugin method to set NODE\_ENV to production that takes out propType validation and extra warnings.

**Explain switching components.**

It is a component that renders one of many components. For this you need to use object to map prop values to components. Basically, it allows the user to turn the setting on or off.

**How will you update props in React?**

It is not possible as we can’t update props in ReactJS as they are read-only.

**What do you understand by the term ‘Restructuring’?**

In React, "restructuring" refers to the process of reorganizing or refactoring components, state management, or overall architecture of a React application to improve its efficiency, maintainability, or other aspects of its development.

**What do you understand by the term ‘Destructuring’?**

In React (and JavaScript in general), "destructuring" refers to a convenient way of extracting multiple properties from an array or object and assigning them to variables. Destructuring makes it easier to access and use data, especially within React components where props and state are often passed as objects.

### Array Destructuring

Array destructuring allows you to unpack values from arrays into distinct variables. Here’s an example:

const numbers = [1, 2, 3];

// Destructuring the array

const [first, second, third] = numbers;

console.log(first); // Output: 1

console.log(second); // Output: 2

console.log(third); // Output: 3

### Object Destructuring

Object destructuring allows you to unpack properties from objects into distinct variables. This is especially useful in React when dealing with props and state.

#### Example with Props:

function Greeting({ name, age }) {

return (

<div>

<p>Hello, {name}!</p>

<p>You are {age} years old.</p>

</div>

);

}

// Usage

<Greeting name="Alice" age={25} />

### Nested Destructuring

Destructuring can also be applied to nested objects and arrays.

const user = {

name: 'Alice',

address: {

city: 'Wonderland',

zip: '12345'

}

};

// Nested object destructuring

const { name, address: { city, zip } } = user;

console.log(name); // Output: Alice

console.log(city); // Output: Wonderland

console.log(zip); // Output: 12345

**Can the values of props be updated?**

No, as props are immutable and top-down. This signifies that parent can transmit prop values to a child but the latter can’t modify them.

**Can I use web components in React application?**

Yes, you can. While it is not used by many developers, it is required when using third-party UI components.

**Why is ‘prop-types’ library used?**

The prop-types library in React is used to perform runtime type checking for the props that are passed to components. This helps ensure that the data your components receive is of the correct type and structure, reducing the likelihood of bugs and making your code more robust and easier to maintain.

**What are controlled components?**

Controlled components in React are components where the form data is handled by the React component itself through its state. In other words, a controlled component has its state managed by React, making the component's state the "single source of truth" for the form data.

### Key Characteristics of Controlled Components:

1. **State Management**:
   * The value of the form input is controlled by the state of the React component.
   * Any changes to the input's value update the component's state, and any state changes update the input's value.
2. **Event Handling**:
   * Controlled components use event handlers (like onChange) to update the state with the new input value whenever the user types into the input field.
3. **Synchronized Data**:
   * The input value and the component state are always in sync, ensuring consistent and predictable behavior.

### Example of a Controlled Component:

import React, { useState } from 'react';

function ControlledInput() {

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

const handleChange = (event) => {

setInputValue(event.target.value);

};

const handleSubmit = (event) => {

event.preventDefault();

alert('Input value: ' + inputValue);

};

return (

<form onSubmit={handleSubmit}>

<label>

Name:

<input type="text" value={inputValue} onChange={handleChange} />

</label>

<button type="submit">Submit</button>

</form>

);

}

export default ControlledInput;

### Benefits of Controlled Components:

* **Predictable Behavior**: Since the form data is controlled by the component's state, it is easier to predict and manage the component's behavior.
* **Validation and Formatting**: It is easier to implement input validation and formatting since you can control and manipulate the input value directly in the event handler.
* **Integration with State Management**: Controlled components integrate well with React's state management, making it easier to manage complex form interactions and state dependencies.

**What are uncontrolled components?**

Uncontrolled components in React are components where the form data is handled by the DOM itself, rather than by the React component's state. This means that the input elements maintain their own state, and the React component accesses the input values using references (refs) when needed.

### Key Characteristics of Uncontrolled Components:

1. **DOM-Managed State**:
   * The form input values are managed by the DOM.
   * The React component does not directly control the state of the form inputs.
2. **Refs for Access**:
   * React's useRef hook (or createRef in class components) is used to create a reference to the input element.
   * The reference allows the component to access the input value when necessary (e.g., on form submission).
3. **Less Code for State Management**:
   * Since the state is managed by the DOM, there's no need for state management code in the component.
   * This can make the component code simpler, but less predictable compared to controlled components.

### Example of an Uncontrolled Component:

import React, { useRef } from 'react';

function UncontrolledInput() {

const inputRef = useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

alert('Input value: ' + inputRef.current.value);

};

return (

<form onSubmit={handleSubmit}>

<label>

Name:

<input type="text" ref={inputRef} />

</label>

<button type="submit">Submit</button>

</form>

);

}

export default UncontrolledInput;

### Benefits of Uncontrolled Components:

* **Simplicity**: They require less boilerplate code for managing state, making the component code simpler in some cases.
* **Less Overhead**: No need to handle state updates in the component, which can reduce the overhead of state management in simple forms.

### Drawbacks of Uncontrolled Components:

* **Less Predictable**: Since the state is managed by the DOM, it can be harder to predict and manage the component's behavior.
* **Limited Control**: Implementing validation, conditional rendering, or other dynamic behaviors can be more challenging compared to controlled components.
* **Harder to Test**: Testing can be more difficult because the component relies on the actual DOM for its state, rather than the React state.

**Comparison of controlled & uncontrolled Components:**

| **Feature** | **Controlled Components** | **Uncontrolled Components** |
| --- | --- | --- |
| State Management | Managed by the React component's state | Managed by the DOM |
| Access to Values | Through component state | Through refs |
| Event Handling | Uses onChange to update state | Uses refs to read values when needed |
| Complexity | More code for state management | Simpler, less boilerplate |
| Predictability | More predictable and consistent | Less predictable, relies on DOM state |
| Validation and Formatting | Easier to implement in event handlers | More challenging, often done on submit |

**What are refs?**

In React, "refs" (short for references) are a feature that allows you to directly access and manipulate DOM elements or React components in your application. They are commonly used for:

1. **Accessing DOM Elements**: Refs can be used to interact with DOM elements directly, for instance, to focus an input, select text, or perform animations.
2. **Storing Mutable Values**: They can store values that don't cause re-rendering when updated, which is useful for managing mutable data outside of React's state system.
3. **Triggering Imperative Animations**: Refs are often used in scenarios where you need to trigger animations or perform actions outside the typical React data flow.

**How do you implement routing in ReactJS?**

In React, routing is commonly implemented using the react-router-dom library. This library provides a set of components and hooks for managing navigation and rendering different components based on the URL path. Here's a basic guide to setting up routing in a React application:

1. **Installation**: First, install react-router-dom using npm or yarn:

npm install react-router-dom

1. **Setting Up the Router**: Wrap your application's root component with the BrowserRouter component. This component provides the routing context to the rest of your app.

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Home from './Home';

import About from './About';

import Contact from './Contact';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

</Routes>

</Router>

);

}

export default App;

1. **Defining Routes**: Use the Routes component to define all your routes, and the Route component for each individual route. Each Route component specifies a path prop for the URL and an element prop for the component to render.
2. path: The URL path that triggers the route.
3. element: The component to render when the path matches.
4. **Navigation**: Use the Link component or the useNavigate hook for navigation within the app without causing a full page reload.

import { Link } from 'react-router-dom';

function Navbar() {

return (

<nav>

<ul>

<li><Link to="/">Home</Link></li>

<li><Link to="/about">About</Link></li>

<li><Link to="/contact">Contact</Link></li>

</ul>

</nav>

);

}

export default Navbar;

1. **Dynamic Routes**: For dynamic routing (e.g., user profiles), you can use path parameters:

import { useParams } from 'react-router-dom';

function UserProfile() {

const { userId } = useParams();

// Fetch and display user data based on userId

return <div>User Profile: {userId}</div>;

}

// In your routes setup

<Route path="/user/:userId" element={<UserProfile />} />

In this setup, navigating to /user/123 would render the UserProfile component with userId set to 123.

**Can you attach a JSX element to other JSX components?**

Yes, you can add JSX elements to other JSX components.

import React from 'react';

// Child component

function ChildComponent({ element }) {

return (

<div>

<h2>Child Component</h2>

{element}

</div>

);

}

// Parent component

function ParentComponent() {

const element = <p>This is a passed JSX element!</p>;

return (

<div>

<h1>Parent Component</h1>

<ChildComponent element={element} />

</div>

);

}

export default ParentComponent;

With pure components, what is the issue of using render props?

When creating a function inside the render method, it negates the performance of the pure component.

**Explain the windowing technique.**

It is a technique that furnishes only a small number of rows at any given time. This reduces the time to re-render not only the elements but also the DOM nodes.

The windowing (or virtualization) technique in React is used to optimize the rendering of large lists or grids of items. Instead of rendering all the items at once, windowing renders only the visible items and a few buffer items around the visible area. This improves performance and reduces memory usage by limiting the number of DOM nodes that need to be managed.

**How can you fetch data with React Hooks?**

We can use useEffect that fetches the data through axios from the API.

**What is Formik?**

It is a third-party library that aids in form programming and validation.

**Can you list some middleware choices for tackling asynchronous calls in Redux?**

Redux Promise, Redux Saga Redux Thunk

**What is one-way data flow in React?**

One-way data flow in React refers to the principle where data flows in a single direction through the components of a React application. This means that data is passed down from parent components to child components via props, and any changes to that data or state occur via callbacks or events passed back up to parent components.

**Explain Concurrent Rendering.**

Concurrent rendering is a feature introduced in React to improve the responsiveness and performance of user interfaces, especially for applications with complex and interactive UIs. It allows React to work on multiple tasks (like rendering, updating state, handling events) simultaneously, prioritizing more important updates and deferring less critical ones.

**Are concurrent mode and async mode different?**

### Concurrent Mode:

* **Purpose**: Enhances React's rendering pipeline for improved performance and responsiveness.
* **Key Features**: Time slicing breaks rendering tasks into smaller chunks, while Suspense allows components to suspend rendering during async operations.
* **Benefits**: Improves UI responsiveness by prioritizing critical updates and managing rendering efficiently.
* **Implementation**: Managed internally by React's core architecture without explicit enabling.

### Async Mode:

* **Purpose**: Handles asynchronous operations within individual components, such as data fetching or side effects.
* **Usage**: Components use async/await or Promises to manage delays in async tasks without blocking the main thread.
* **Features**: Facilitates handling of async code and errors, ensuring components remain responsive.
* **Implementation**: Implemented within component logic using JavaScript's async/await or Promise APIs.

### Differences:

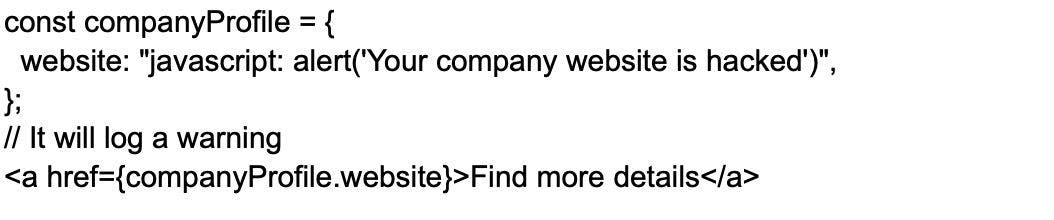
* **Scope**: Concurrent mode optimizes React's rendering and state management, while async mode deals with async operations within component logic.
* **Focus**: Concurrent mode focuses on performance and rendering efficiency, whereas async mode focuses on managing async tasks within components.

### Relationship:

* Both modes complement each other in creating performant and responsive React applications.
* Concurrent mode enhances overall rendering performance, while async mode ensures components handle async operations effectively without blocking UI updates.

**Can JavaScript urls be used in React 16.9?**

Yes, but it will give a warning in the console.



**Tell us about React Server components.**

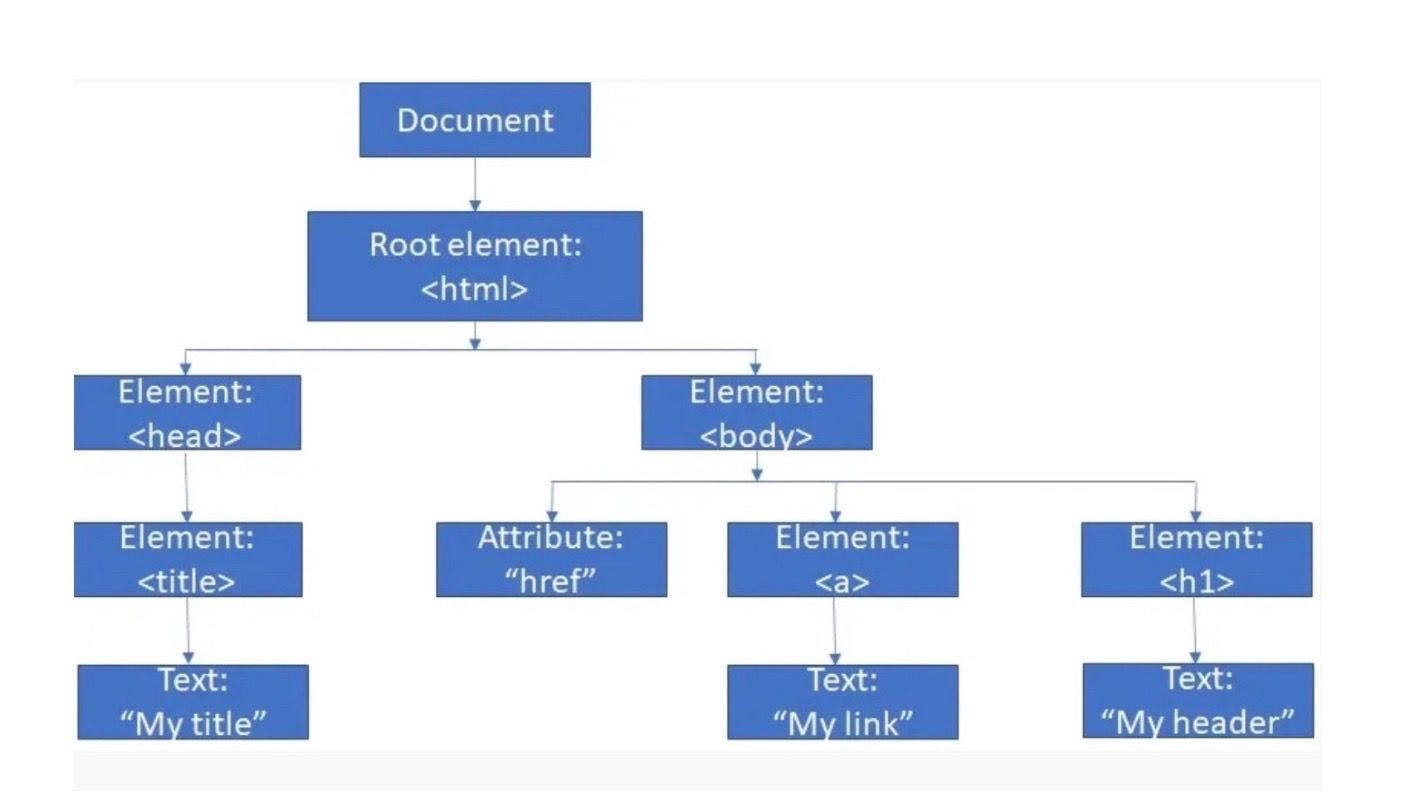
React Server Components are a feature introduced to allow developers to build modern user interfaces that are more efficient and faster by leveraging the server to render parts of a React application. This approach combines the best parts of client-side and server-side rendering, enabling developers to offload more work to the server while maintaining a seamless, interactive experience on the client.

**What is prop drilling?**

It is a situation where the same data is sent at every interdependent level until it reaches the final level.

Prop drilling refers to the process of passing data from a parent component down to deeply nested child components through intermediate components via props. This can lead to problems when components that do not need the data directly have to pass it along just to ensure that it reaches the necessary component, leading to excessive and sometimes unnecessary prop passing.

**What is virtual DOM?**



DOM is the abbreviated version of the Document Object Model. The DOM shows the logical tree structure for an HTML document. There is a node at the end of each tree branch, with every tree node containing one or more objects.

In ReactJS, a lighter version of real DOM is stored in the memory. It is called the Virtual DOM. When there are any changes in specific objects, instead of updating all objects, only those specific changes are updated in the real DOM. This increases the efficiency and speed of the program. For a ReactJS interview question like this, if you have the opportunity to illustrate through a diagram, you must, as it shows the depth of your understanding.

**Why do we use keys in Lists?**

The main reason why we use keys in lists are as follows

* It is an identifier to find which items have changed, updated, or deleted from lists
* Find out which items need to be re-rendered

**Explain React forms?**

React forms allow users to interact with web applications. They help users to enter the information when required. They contain buttons, text fields, checkboxes, etc.

**What are the components in React?**

In a React application, they are independent and reusable bits of code that can be processed separately. A single app may therefore contain multiple components.

**Tell us about MobX.**

It is a simple, scalable, and battle-tested state management solution. It is based on Reactive programming principles.

To use it, the following packages need to be installed

**npm install mobx --save**

**npm install mobx-React --save**

**What is the use of eslint plugin for hooks?**

It enforces the Rules of Hooks. Precisely, it is a static code analysis tool for finding out faulty patterns in JavaScript code.

The ESLint plugin for React Hooks, eslint-plugin-react-hooks, is used to enforce the rules of Hooks and to help developers avoid common pitfalls when using Hooks in their React applications. This plugin ensures that Hooks are used correctly and consistently, helping to maintain code quality and preventing potential bugs.

**What are React Dev Tools?**

It lets you analyze the component hierarchy and includes component props and state. You can find [React Dev Tools](https://www.turing.com/kb/react-developer-tools-to-make-software-development-easy) both as a standalone app and as a browser extension.

React DevTools are a set of tools that help developers inspect, debug, and optimize React applications. They provide a user-friendly interface integrated into the browser's developer tools to visualize and interact with the React component hierarchy, state, and props.

**Why would you do it if DevTools does not load in Chrome for local files?**

On opening a local HTML file in your browser, you must first open Chrome Extensions and check Allow access to file URLs.

**Tell me some advantages of React over VueJS.**

Here's a comparison between React, Angular, and Vue in a tabular format:

| **Feature** | **React** | **Angular** | **Vue** |
| --- | --- | --- | --- |
| **Developer** | Facebook | Google | Evan You (individual) |
| **Initial Release** | 2013 | 2010 | 2014 |
| **Architecture** | Library for building UIs | Full-fledged MVC/MVVM framework | Progressive framework for building UIs |
| **Language** | JavaScript/TypeScript | TypeScript | JavaScript/TypeScript |
| **Data Binding** | One-way binding | Two-way binding | Two-way binding |
| **DOM Manipulation** | Virtual DOM | Real DOM (with change detection) | Virtual DOM |
| **Learning Curve** | Moderate | Steep | Gentle |
| **State Management** | Context API, Redux, MobX | Built-in with RxJS (NgRx for advanced) | Vuex |
| **Templating** | JSX | HTML with Angular directives | HTML with Vue directives |
| **Component Communication** | Props and state | Input/Output decorators | Props and events |
| **Routing** | React Router | Angular Router | Vue Router |
| **Forms** | No built-in solution (use libraries) | Built-in with Angular Forms (Template & Reactive) | Vue Formulate, Vuelidate |
| **Performance** | High | High | High |
| **Community and Ecosystem** | Large, strong community | Large, strong community | Growing, vibrant community |
| **Size** | Lightweight | Larger due to being a full framework | Lightweight |
| **Usage** | Single-page applications (SPAs), mobile apps, server-rendered apps | Large-scale applications, enterprise-level apps | Single-page applications (SPAs), small to medium-sized apps |
| **Official CLI** | Create React App | Angular CLI | Vue CLI |
| **Official State Management** | No (Redux, MobX, etc. are popular choices) | NgRx | Vuex |
| **Mobile Development** | React Native | NativeScript | Vue Native (Community-driven) |
| **Integration** | Easier to integrate with other libraries | Harder to integrate into existing projects due to its comprehensive nature | Easier to integrate with other projects |

### React

**Advantages over Angular:**

1. **Flexibility**: React is more flexible and unopinionated, allowing developers to choose their own libraries for state management, routing, etc., whereas Angular is more prescriptive.
2. **Learning Curve**: React's learning curve is generally considered to be gentler than Angular's, especially for developers familiar with JavaScript.
3. **Performance**: React’s use of a virtual DOM can lead to better performance in some scenarios, especially with dynamic and interactive UIs.
4. **Component Reusability**: React’s component-based architecture promotes reusability and ease of testing.
5. **Ecosystem and Community**: Large ecosystem and community with numerous third-party libraries, tools, and a wide range of tutorials and resources.

**Advantages over Vue:**

1. **Ecosystem and Community Size**: React has a larger community and a more mature ecosystem compared to Vue, providing more resources and third-party libraries.
2. **Job Market**: React has a larger presence in the job market, making it a safer choice for developers looking for employment opportunities.
3. **Performance**: React's virtual DOM and fiber architecture offer high performance for large-scale applications.

### Angular

**Advantages over React:**

1. **Full-Fledged Framework**: Angular provides a comprehensive set of tools out of the box, including routing, forms, HTTP client, and more, reducing the need to integrate third-party libraries.
2. **Two-Way Data Binding**: Simplifies the synchronization between the model and the view, making it easier to manage form inputs and other dynamic elements.
3. **Dependency Injection**: Built-in dependency injection system promotes better code organization and testability.
4. **TypeScript**: Angular’s use of TypeScript offers advantages like better code maintainability, refactoring, and catching errors at compile time.
5. **Enterprise-Level Features**: Angular is often favored for large, complex applications due to its robust features and structured approach.

**Advantages over Vue:**

1. **Enterprise Support**: Angular, being developed by Google, has strong backing and is often preferred for large enterprise applications.
2. **Comprehensive Documentation**: Angular's documentation is extensive and thorough, which is beneficial for larger teams and complex projects.
3. **Mature Tooling**: Angular CLI is a powerful tool that streamlines development processes like scaffolding, testing, and deployment.

### Vue

**Advantages over React:**

1. **Gentle Learning Curve**: Vue is often praised for its simplicity and ease of learning, making it a great choice for beginners.
2. **Integrated Solutions**: Vue provides a more integrated approach with Vuex for state management and Vue Router for routing, reducing the need to find and integrate third-party solutions.
3. **Reactivity System**: Vue’s reactivity system is more straightforward and powerful, making it easier to work with reactive data.
4. **Smaller Bundle Size**: Vue applications typically have a smaller bundle size compared to React applications, which can lead to better performance.

**Advantages over Angular:**

1. **Simplicity**: Vue is simpler and more intuitive compared to Angular, which can be over-engineered for smaller applications.
2. **Flexibility**: Vue offers a balance between the prescriptive nature of Angular and the flexibility of React, making it easier to integrate into existing projects.
3. **Performance**: Vue’s virtual DOM implementation is highly optimized, often resulting in better performance in smaller to medium-sized applications.
4. **Less Verbose**: Vue’s templates are less verbose compared to Angular’s, leading to cleaner and more readable code.

### Summary

* **React**: Offers flexibility, a gentle learning curve, high performance, and a large ecosystem.
* **Angular**: Provides a comprehensive set of tools, two-way data binding, TypeScript support, and is suited for large enterprise applications.
* **Vue**: Boasts simplicity, ease of integration, an efficient reactivity system, and a smaller bundle size.

**What is render hijacking in React?**

It is the ability to control what one component will output from another component. Here you wrap a component with a higher order component.

Render hijacking in React refers to the technique where a higher-order component (HOC) modifies or controls the rendering behavior of a wrapped component. Essentially, the HOC intercepts the render method of the wrapped component and alters its output before it is displayed. This allows developers to add, modify, or suppress certain aspects of the component's rendering process.

**How can you pass numbers to the React component?**

You can pass numbers to React components using curly braces({}) where as strings in quotes

In React, you pass numbers and strings to components as props using different syntaxes:

* **Numbers**: Use curly braces {}.
* **Strings**: Use either quotes "" or curly braces {}.

Here's a concise explanation with examples:

### Example: Passing Number and String Props

1. **Define the Child Component**: Create a component that will receive the number and string as props.

// ChildComponent.jsx

import React from 'react';

const ChildComponent = ({ numberProp, stringProp }) => {

return (

<div>

<p>Number: {numberProp}</p>

<p>String: {stringProp}</p>

</div>

);

};

export default ChildComponent;

1. **Pass Props from the Parent Component**: Use the child component within a parent component, passing the number and string as props.

// ParentComponent.jsx

import React from 'react';

import ChildComponent from './ChildComponent';

const ParentComponent = () => {

return (

<div>

<h1>Passing Props Example</h1>

<ChildComponent numberProp={42} stringProp="Hello, World!" />

</div>

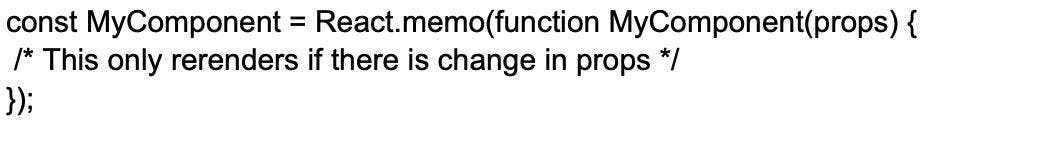
);

};

export default ParentComponent;

**Explain React memo function.**

Using PureComponent or shouldComponentUpdate you can restrict class components from re-rendering when input properties are the same. You can do the same with the React memo function.



**How uncaught errors behave in React ?**

It results in unmounting of the entire React component tree. In React, uncaught errors can disrupt the normal rendering process and potentially crash the application.

**What are default props?**

It is a React component property to set default values for the props argument. They are used for undefined props only and not for null props.

Default props in React are a way to specify default values for a component's props in case they are not provided by the parent component. This ensures that the component still works as expected even if some props are missing.

import React from 'react';

// Define the Greeting component

const Greeting = ({ greeting }) => {

return (

<div>

<h1>{greeting}</h1>

</div>

);

};

// Set default props

Greeting.defaultProps = {

greeting: 'Hello, World!',

};

export default Greeting;

import React from 'react';

import Greeting from './Greeting';

const App = () => {

return (

<div>

{/\* This will use the default greeting \*/}

<Greeting />

{/\* This will use the provided greeting \*/}

<Greeting greeting="Hi there!" />

</div>

);

};

export default App;

**What browser React applications support?**

It supports all popular browsers including Internet Explorer and above. However, it can support old browsers as well through polyfills.

#### Is it necessary for keys to be globally unique?

While you can use the same keys for two different arrays, they don't need to be globally unique.

**What methods are invoked during error handling?**

static getDerivedStateFromError()

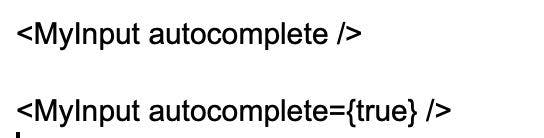
componentDidCatch()

**Do you know the use of the unmountComponentAtNode method?**

The unmountComponentAtNode method in React is used to unmount a React component from the DOM and clean up its event handlers and state. This method is often used when cleaning up the DOM after a React component has been removed or replaced.

**On what condition, do component props defaults to true?**

When you pass no value for a prop. For example, these two are equivalent,



**How will you build a search filter using React?**

You can build a search filter using following steps:

* First declare React states for input values.
* For entering the search term, create HTML input state
* Now, on Change function update state
* Add Array filter () to list of items with search term value

**How can you create a simple counter using React?**

* First, create a React state that can store the count value.
* Then declare JS functions to increment or decrement the value using setState().
* Finally add HTML buttons with onClick to JSX code.

**How can you display a list in React?**

* First, declare a list of items using JS Array.
* Now to access each item use Array.map()
* Finally, return JSX code for each item

**How will you create an Image Slider using ReactJS?**

* Create an array of objects that contain an id and image URL.
* To enable pre/next functionality, we decrease or increase the index value of the currently active image.
* Finally update the active image index using the clicked option.

**What are the steps to add Font Awesome icons in React?**

npm i font-awesome

import 'font-awesome/css/font-awesome.min.css'



**How will you use TypeScript in create-React-app applications?**

There is built-in support. You just have to pass --typescript option,

npx create-React-app my-app --typescript

**Can the statics object work with ES classes?**

No, it works only with React.createClass():

**How will you make Redux Form initialValues get updated from state?**

For this, we use enableReinitialize : true setting.

**What are hook in react**

In React, "hooks" are special functions that allow you to use state and other React features in functional components, which were traditionally only available in class components. Hooks provide a more streamlined and functional approach to manage component state, lifecycle events, and other functionalities.

### useState Hook

#### Theory

The useState hook is used to add state to functional components. It returns an array with two elements: the current state value and a function to update it.

#### Syntax

const [state, setState] = useState(initialState);

#### Use Cases

1. **Managing Form Inputs**: Store values of input fields.
2. **Toggle Components**: Show or hide components.
3. **Counter Implementation**: Simple increment/decrement logic.
4. **Fetch Data**: Manage loading, error, and data states.
5. [Adding state to a component](https://react.dev/reference/react/useState#adding-state-to-a-component)
6. [Updating state based on the previous state](https://react.dev/reference/react/useState#updating-state-based-on-the-previous-state)
7. [Updating objects and arrays in state](https://react.dev/reference/react/useState#updating-objects-and-arrays-in-state)
8. [Avoiding recreating the initial state](https://react.dev/reference/react/useState#avoiding-recreating-the-initial-state)
9. [Resetting state with a key](https://react.dev/reference/react/useState#resetting-state-with-a-key)
10. [Storing information from previous renders](https://react.dev/reference/react/useState#storing-information-from-previous-renders)

### useEffect Hook

#### Theory

The useEffect hook lets you perform side effects in functional components. It serves the purpose of componentDidMount, componentDidUpdate, and componentWillUnmount in class components. You can optionally specify dependencies to control when the effect runs.

#### Syntax

useEffect(() => {

// Side effect code

return () => {

// Cleanup code (optional)

};

}, [dependencies]);

#### Use Cases

1. **Fetching Data**: Perform API calls when the component mounts or dependencies change.
2. **Event Listeners**: Add event listeners and clean them up on unmount.
3. **Timers**: Set up intervals or timeouts and clear them on unmount.
4. **Updating Document Title**: Change the document title based on component state or props.
5. [Connecting to an external system](https://react.dev/reference/react/useEffect#connecting-to-an-external-system)
6. [Wrapping Effects in custom Hooks](https://react.dev/reference/react/useEffect#wrapping-effects-in-custom-hooks)
7. [Controlling a non-React widget](https://react.dev/reference/react/useEffect#controlling-a-non-react-widget)
8. [Fetching data with Effects](https://react.dev/reference/react/useEffect#fetching-data-with-effects)
9. [Specifying reactive dependencies](https://react.dev/reference/react/useEffect#specifying-reactive-dependencies)
10. [Updating state based on previous state from an Effect](https://react.dev/reference/react/useEffect#updating-state-based-on-previous-state-from-an-effect)
11. [Removing unnecessary object dependencies](https://react.dev/reference/react/useEffect#removing-unnecessary-object-dependencies)
12. [Removing unnecessary function dependencies](https://react.dev/reference/react/useEffect#removing-unnecessary-function-dependencies)
13. [Reading the latest props and state from an Effect](https://react.dev/reference/react/useEffect#reading-the-latest-props-and-state-from-an-effect)
14. [Displaying different content on the server and the client](https://react.dev/reference/react/useEffect#displaying-different-content-on-the-server-and-the-client)

### useContext Hook

#### Theory

The useContext hook allows functional components to access context values. Context provides a way to pass data through the component tree without having to pass props down manually at every level.

#### Syntax

const contextValue = useContext(MyContext);

#### Use Cases

1. **Global State Management**: Manage state accessible by many components without prop drilling.
2. **Theme Switching**: Manage themes across an application.
3. **Authentication**: Share user authentication status and information.
4. **Localization**: Provide and manage translations and locale data.
5. [Passing data deeply into the tree](https://react.dev/reference/react/useContext#passing-data-deeply-into-the-tree)
6. [Updating data passed via context](https://react.dev/reference/react/useContext#updating-data-passed-via-context)
7. [Specifying a fallback default value](https://react.dev/reference/react/useContext#specifying-a-fallback-default-value)
8. [Overriding context for a part of the tree](https://react.dev/reference/react/useContext#overriding-context-for-a-part-of-the-tree)
9. [Optimizing re-renders when passing objects and functions](https://react.dev/reference/react/useContext#optimizing-re-renders-when-passing-objects-and-functions)

### useCallback Hook

#### Theory

The useCallback hook returns a memoized version of the callback function that only changes if one of the dependencies has changed. This is useful for optimizing performance by preventing unnecessary re-renders.

#### Syntax

const memoizedCallback = useCallback(() => {

// Callback logic

}, [dependencies]);

#### Use Cases

1. **Preventing Re-creation of Functions**: Useful in optimizing child components that rely on functions passed as props.
2. **Event Handlers**: Pass stable event handlers to optimized child components.
3. **Performance Optimization**: Reduce the number of times a function is re-created, improving performance in large applications.
4. [Skipping re-rendering of components](https://react.dev/reference/react/useCallback#skipping-re-rendering-of-components)
5. [Updating state from a memoized callback](https://react.dev/reference/react/useCallback#updating-state-from-a-memoized-callback)
6. [Preventing an Effect from firing too often](https://react.dev/reference/react/useCallback#preventing-an-effect-from-firing-too-often)
7. [Optimizing a custom Hook](https://react.dev/reference/react/useCallback#optimizing-a-custom-hook)

### useMemo Hook

#### Theory

The useMemo hook returns a memoized value that only recalculates if one of the dependencies has changed. It is used to optimize performance by avoiding expensive calculations on every render.

#### Syntax

const memoizedValue = useMemo(() => {

// Calculation logic

return result;

}, [dependencies]);

#### Use Cases

1. **Expensive Calculations**: Avoid recalculating heavy computations on every render.
2. **Optimizing Render Performance**: Memoize derived state to prevent unnecessary re-renders.
3. **Referential Equality**: Ensure that a memoized value has stable references for performance optimization.
4. [Skipping expensive recalculations](https://react.dev/reference/react/useMemo#skipping-expensive-recalculations)
5. [Skipping re-rendering of components](https://react.dev/reference/react/useMemo#skipping-re-rendering-of-components)
6. [Memoizing a dependency of another Hook](https://react.dev/reference/react/useMemo#memoizing-a-dependency-of-another-hook)
7. [Memoizing a function](https://react.dev/reference/react/useMemo#memoizing-a-function)

### useRef Hook

#### Theory

The useRef hook returns a mutable object which persists across renders. It is often used to directly access and manipulate DOM elements, as well as store any mutable value that doesn’t cause a re-render when updated.

#### Syntax

const refContainer = useRef(initialValue);

#### Use Cases

1. **Accessing DOM Elements**: Focus, scroll, or other direct manipulations.
2. **Storing Mutable Values**: Store timers, intervals, or other values that don’t need to trigger re-renders.
3. **Persisting Values**: Hold on to values between renders without causing re-renders.
4. [Referencing a value with a ref](https://react.dev/reference/react/useRef#referencing-a-value-with-a-ref)
5. [Manipulating the DOM with a ref](https://react.dev/reference/react/useRef#manipulating-the-dom-with-a-ref)
6. [Avoiding recreating the ref contents](https://react.dev/reference/react/useRef#avoiding-recreating-the-ref-contents)

**useReducer**

The useReducer hook in React is an alternative to useState for managing state in functional components, particularly when the state logic is complex or involves multiple sub-values. It is based on the concept of reducers, which is a common pattern in state management libraries like Redux.

Use Cases

[Adding a reducer to a component](https://react.dev/reference/react/useReducer#adding-a-reducer-to-a-component)

[Writing the reducer function](https://react.dev/reference/react/useReducer#writing-the-reducer-function)

[Avoiding recreating the initial state](https://react.dev/reference/react/useReducer#avoiding-recreating-the-initial-state)