***What is Redux?***

[***Redux***](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-with-redux)***is an open-source, JavaScript library used to manage the application state. React uses Redux to build the user interface***

***\*Action \*Reducer \*Store \*View***

***PORTAL:***

***Portal* is a way to render children into a DOM node, that exists outside the DOM hierarchy of the parent component.**

**SWITCHING COMPONENT:**

**A *switching component* is a component that renders one of many components.**

1. **What is the difference between setState() and replaceState() methods?**

When you use setState() the current and previous states are merged. replaceState() throws out the current state, and replaces it with only what you provide. Usually setState() is used unless you really need to remove all previous keys for some reason

1. **Why you can't update props in React?**

The React philosophy is that props should be *immutable* and *top-down*. This means that a parent can send any prop values to a child, but the child can't modify received props.

1. **What is React Router?**

**React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly**

1. **What is render hijacking in react?**

**The concept of render hijacking is the ability to control what a component will output from another component**

1. **What is Relay?**

**Relay is a JavaScript framework for providing a data layer and client-server communication to web applications**

**What is React memo function?**

**Restricts functional component from rendering**

Class components can be restricted from rendering using **PureComponent or shouldComponentUpdate**. Now you can do the same with function components by wrapping them in **React.memo**.

| **>>>>>>>>>>>>Flux** | **Redux** |
| --- | --- |
| State is mutable | State is immutable |
| The Store contains both state and change logic | The Store and change logic are separate |
| There are multiple stores exist | There is only one store exist |
| All the stores are disconnected and flat | Single store with hierarchical reducers |

**DIFFLING ALGORITHM:**

**The diffing algorithms is generating the minimum number of operations to transform one tree into another**

**. Explain Flux.**

**Flux is an architectural pattern which enforces the uni-directional data flow**

**46. What is React Router?**

React Router is a powerful routing library built on top of React

**1. What are the advantages of using React?**

MVC is generally abbreviated as Model View Controller.

* 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.
* 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.
* 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.
* 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.
* React provides you the freedom to choose the tools, libraries, and architecture for developing an application based on your requirement.

**2. What is JSX?**

JSX stands for JavaScript XML.  
It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).  
As stated in the official docs of React, JSX provides syntactic sugar for React.createElement( ) function.  
**\*\*Note- We can create react applications without using JSX as well.**  
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);

As one can see in the code above, we are directly using HTML inside JavaScript.

**3. What are the differences between functional and class components?**

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on feature basis. After the introduction of Hooks, functional components are equivalent to class components.  
Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these both components differ.  
On the following basis let’s compare functional and class components:

**4. What is the virtual DOM? How does react use the virtual DOM to render the UI?**

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM.

**Why was virtual DOM introduced?** DOM manipulation is an integral part of any web application, but DOM manipulation is quite slow when compared to other operations in JavaScript.  
The efficiency of the application gets affected when several DOM manipulations are being done. Most JavaScript frameworks update the entire DOM even when a small part of the DOM changes.  
For example, consider a list that is being rendered inside the DOM. If one of the items in the list changes, the entire list gets rendered again instead of just rendering the item that was changed/updated. This is called inefficient updating.  
To address the problem of inefficient updating, the react team introduced the concept of virtual DOM.  
  
**How does it work?**

For every DOM object, there is a corresponding virtual DOM object(copy), which has the same properties.  
The main difference between the real DOM object and the virtual DOM object is that any changes in the virtual DOM object will not reflect on the screen directly. Consider a virtual DOM object as a blueprint of the real DOM object.  
Whenever a JSX element gets rendered, every virtual DOM object gets updated.  
  
**\*\*Note- One may think updating every virtual DOM object might be inefficient, but that’s not the case. Updating the virtual DOM is much faster than updating the real DOM since we are just updating the blueprint of the real DOM.**  
  
***React uses two virtual DOMs to render the user interface. One of them is used to store the current state of the objects and the other to store the previous state of the objects.***  
Whenever the virtual DOM gets updated, react compares the two virtual DOMs and gets to know about which virtual DOM objects were updated.  
After knowing which objects were updated, react renders only those objects inside the real DOM instead of rendering the complete real DOM.  
This way, with the use of virtual DOM, react solves the problem of inefficient updating.

**5. What are the differences between controlled and uncontrolled components?**

Controlled and uncontrolled components are just different approaches to handling input form elements in react.

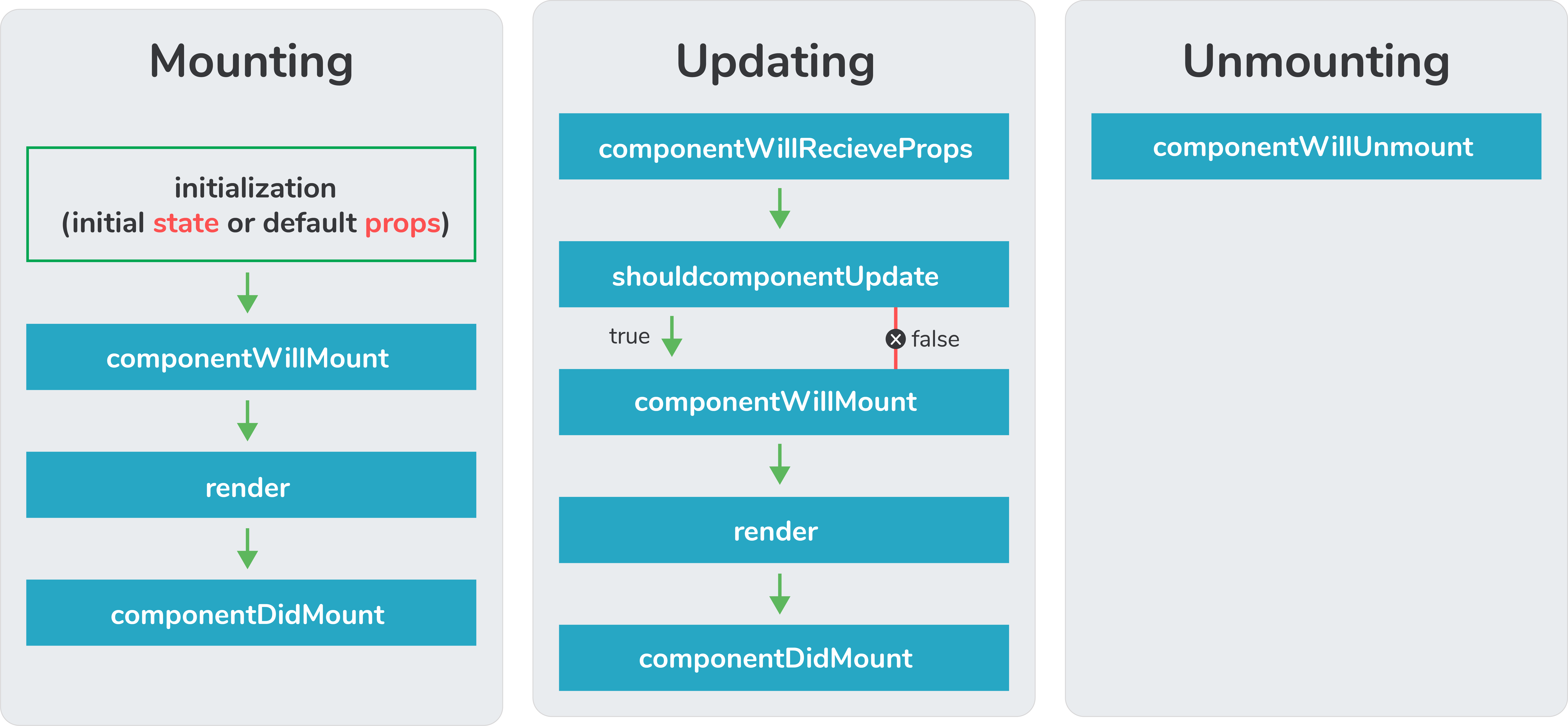
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Controlled component In a controlled component, the value of the input element is controlled by React.**

**Uncontrolled component In an uncontrolled component, the value of the input element is handled by the DOM itself.**

**6. What are the different lifecycle methods in React?**

Every component in React has lifecycle methods that we can tap into, to trigger changes at a particular phase of the life cycle.  
Each component in react goes through three phases: **Mounting**, **Updating**, and **Unmounting**.  
There are corresponding lifecycle methods for each of the three phases:  
  
**\*\*Note- In this article, we are discussing the use of lifecycle methods in class components. For utilising lifecycle methods in functional components, react hooks are used.**



**7. Explain Strict Mode in React.**

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

**8. How to prevent re-renders in React?**

**Reason for re-renders in React:**  
Re-rendering of a component and it’s child components occur when props or 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>

);

}

}

**Parent** component is the parent component and **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 component, 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.

**9. Explain React state and props.**

|  |  |
| --- | --- |
| **Props** | **State** |
| Immutable | Owned by its component |
| Has better performance | Locally scoped |
| Can be passed to child components | Witeable/Mutable |
|  | has setState() method to modify properties |
|  | Changes to state can be asynchronous |
|  | can only be passed as props |

**React State**  
Every component in react has a built-in **state** object, which contains all the property values that belong to that component.  
In other words, **the state object** controls the behaviour of a component. Any change in the property values of the state object leads to re-rendering of the component.  
  
**\*\*Note- State object is not available in functional components but, we can use React Hooks to add state to a functional component.**  
  
**How to declare a state object?**  
Example:

**class** Car **extends** React.Component{

constructor(props){

**super**(props);

**this**.state = {

brand: "BMW",

color: "black"

}

}

}

**How to use and update the state object?**

**class** Car **extends** React.Component {

constructor(props) {

**super**(props);

**this**.state = {

brand: "BMW",

color: "Black"

};

}

changeColor() {

**this**.setState(prevState => {

**return** { color: "Red" };

});

}

render() {

**return** (

<div>

<button onClick={() => **this**.changeColor()}>Change Color</button>

<p>{**this**.state.color}</p>

</div>

);

}

}

As one can see in the code above, we can use the state by calling **this.state.propertyName** and we can change the state object property using **setState** method.  
**React Props**  
Every react component, accepts a single object argument called **props** (which stands for “properties”).  
These props can be passed to a component using HTML attributes and the component accepts these props as an argument.  
Using props, we can pass data from one component to another.  
Passing props to a component:  
While rendering a component, we can pass the props as a HTML attribute:

<Car brand="Mercedes"/>

The component receives the props:  
In Class component:

**class** Car **extends** React.Component {

constructor(props) {

**super**(props);

**this**.state = {

brand: **this**.props.brand,

color: "Black"

};

}

}

In Functional component:

**function** Car(props) {

**let** [brand, setBrand] = useState(props.brand);

}

**\*\*\*\*Note- Props are read-only. They cannot be manipulated or changed inside a component.**

**10. Explain React Hooks.**

**What are Hooks?** Hooks are functions that let us “hook into” React state and lifecycle features from **a functional component**.  
React Hooks **cannot** be used in class components. They let us write components without class.  
  
**Why were Hooks introduced in React?**  
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.  
**Example of a hook:**  
**useState hook:**  
In functional components, useState hook lets us define state for a component:

**function** Person(props) {

// We are declaring a state variable called name.

// setName is a function to update/change the value of name

**let** [name, setName] = useState('');

}

The state variable “name” can be directly used inside the HTML.

**11. 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>

);

}

}

**12. 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 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 minimal.

**13. What are keys in React?**

***A key is a special string attribute that needs to be included when using lists of elements.  
Example of a list using key:***

const ids = [**1**,**2**,**3**,**4**,**5**];

const listElements = ids.map(**(id)=>**{

**return**(

<li key={id.toString()}>

{id}

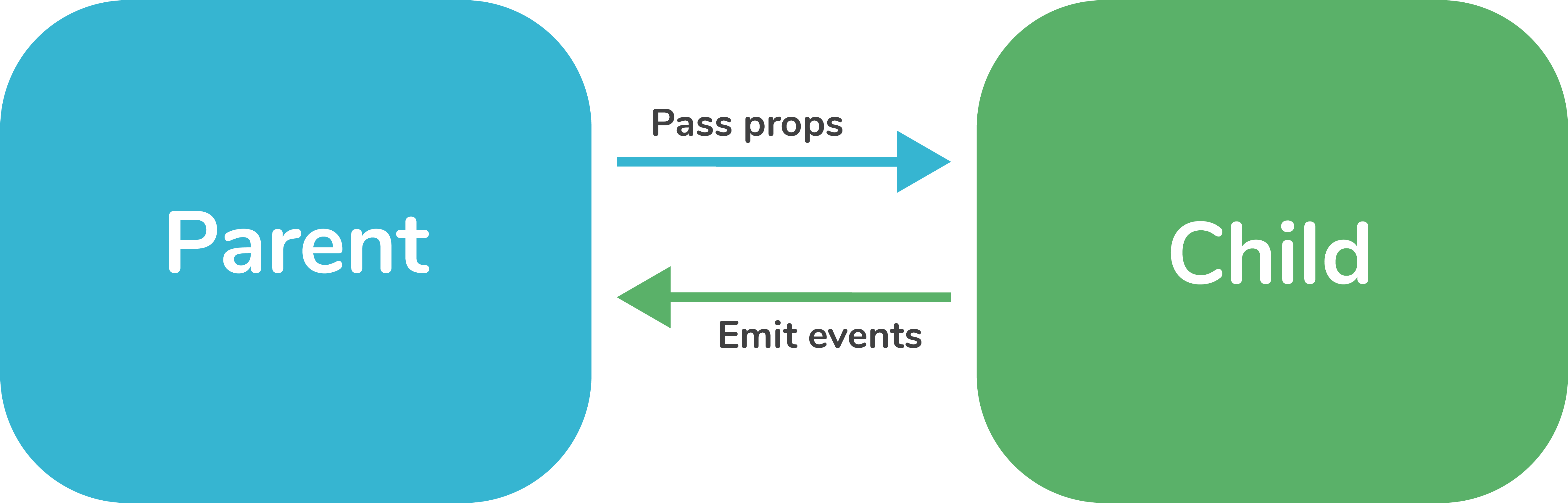
</li>

)

})

**Importance of keys**  
Keys help react identify which elements were added, changed or removed.  
Keys should be given to array elements for providing a unique identity for each element.  
Without keys, React does not understand the order or uniqueness of 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.  
**\*\*\*Note- Keys used within arrays should be unique among siblings. They need not be globally unique.**

**14. How to pass data between react components?**



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

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

**.**

**15. What are Higher Order Components?**

Simply put,

**Higher Order Component(HOC) is a function that takes in a component and returns a new component.**

**16. What is prop drilling in React?**

**Topmost to innermost hierarchy mei data transfer karna hota hai within components-🡪 to send it data has to pass from each component till it reaches the innermost component or desired component. Which is unnecessary-🡪 it is reffered to as prop drilling**

Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested.  
To pass data between such components, we pass props from a source component, and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.  
The disadvantage of using prop drilling is that the components that should otherwise be not aware of the data have access to the data.

1. **What is React?**

React is an **open-source frontend JavaScript library** which is used for building user interfaces especially for single page applications. It is used for handling view layer for web and mobile apps. React was created by [Jordan Walke](https://github.com/jordwalke), a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the major features of React?**

The major features of React are:

* + It uses **VirtualDOM** instead of RealDOM considering that RealDOM manipulations are expensive.
  + Supports **server-side rendering**.
  + Follows **Unidirectional** data flow or data binding.
  + Uses **reusable/composable** UI components to develop the view.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is JSX?**

*JSX* is a XML-like syntax extension to ECMAScript (the acronym stands for *JavaScript XML*). Basically it just provides syntactic sugar for the React.createElement() function, giving us expressiveness of JavaScript along with HTML like template syntax.

In the example below text inside <h1> tag is returned as JavaScript function to the render function.

class App extends React.Component {

render() {

return(

<div>

<h1>{'Welcome to React world!'}</h1>

</div>

)

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between Element and Component?**

An *Element* is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. *Elements* can contain other *Elements* in their props. Creating a React element is cheap. Once an element is created, it is never mutated.

The object representation of React Element would be as follows:

const element = React.createElement(

'div',

{id: 'login-btn'},

'Login'

)

The above React.createElement() function returns an object:

{

type: 'div',

props: {

children: 'Login',

id: 'login-btn'

}

}

And finally it renders to the DOM using ReactDOM.render():

<div id='login-btn'>Login</div>

Whereas a **component** can be declared in several different ways. It can be a class with a render() method. Alternatively, in simple cases, it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output:

const Button = ({ onLogin }) =>

<div id={'login-btn'} onClick={onLogin}>Login</div>

Then JSX gets transpiled to a React.createElement() function tree:

const Button = ({ onLogin }) => React.createElement(

'div',

{ id: 'login-btn', onClick: onLogin },

'Login'

)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to create components in React?**

There are two possible ways to create a component.

* + **Function Components:** This is the simplest way to create a component. Those are pure JavaScript functions that accept props object as first parameter and return React elements:
  + function Greeting({ message }) {
  + return <h1>{`Hello, ${message}`}</h1>

}

* + **Class Components:** You can also use ES6 class to define a component. The above function component can be written as:
  + class Greeting extends React.Component {
  + render() {
  + return <h1>{`Hello, ${this.props.message}`}</h1>
  + }

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **When to use a Class Component over a Function Component?**

If the component needs *state or lifecycle methods* then use class component otherwise use function component. *However, from React 16.8 with the addition of Hooks, you could use state , lifecycle methods and other features that were only available in class component right in your function component.*

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

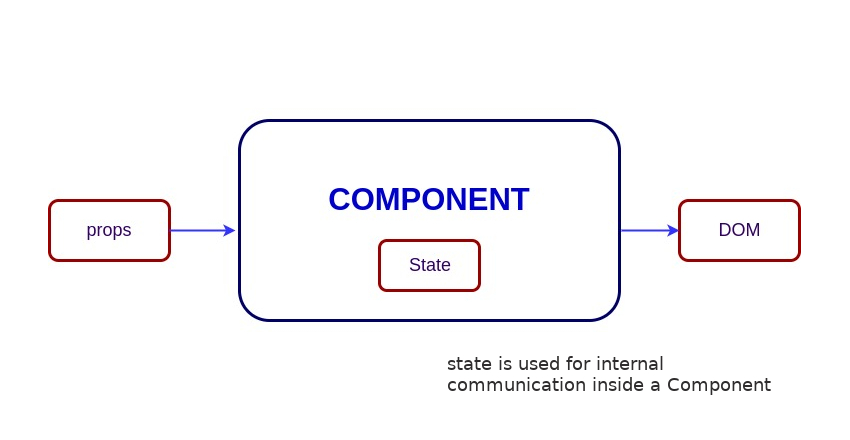
1. **What are Pure Components?**

*React.PureComponent* is exactly the same as *React.Component* except that it handles the shouldComponentUpdate() method for you. When props or state changes, *PureComponent* will do a shallow comparison on both props and state. *Component* on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

**What is state in React?**

*State* of a component is an object that holds some information that may change over the lifetime of the component.

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/state.jpg)

State is similar to props, but it is private and fully controlled by the component. i.e, It is not accessible to any other component til the owner component decides to pass it.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are props in React?**

*Props* are inputs to components.

They are single values or objects containing a set of values that are passed to components on creation using a naming convention similar to HTML-tag attributes. They are data passed down from a parent component to a child component.

1. **What is the difference between state and props?**

Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why should we not update the state directly?**

**If you try to update state directly then it won't re-render the component.**

**Note:** You can directly assign to the state object either in *constructor* or using latest javascript's class field declaration syntax.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of callback function as an argument of setState()?**

The callback function is invoked when setState finished and the component gets rendered. Since setState() is **asynchronous** the callback function is used for any post action.

**Note:** It is recommended to use lifecycle method rather than this callback function.

setState({ name: 'John' }, () => console.log('The name has updated and component re-rendered'))

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between HTML and React event handling?**

Below are some of the main differences between HTML and React event handling,

* 1. In HTML, the event name usually represents in *lowercase* as a convention:

<button onclick='activateLasers()'>

Whereas in React it follows *camelCase* convention:

<button onClick={activateLasers}>

* 1. In HTML, you can return false to prevent default behavior:

<a href='#' onclick='console.log("The link was clicked."); return false;' />

Whereas in React you must call preventDefault() explicitly:

function handleClick(event) {

event.preventDefault()

console.log('The link was clicked.')

}

* 1. In HTML, you need to invoke the function by appending () Whereas in react you should not append () with the function name. (refer "activateLasers" function in the first point for example)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to bind methods or event handlers in JSX callbacks?**

There are 3 possible ways to achieve this:

* 1. **Binding in Constructor:** In JavaScript classes, the methods are not bound by default. The same thing applies for React event handlers defined as class methods. Normally we bind them in constructor.
  2. class Component extends React.Component {
  3. constructor(props) {
  4. super(props)
  5. this.handleClick = this.handleClick.bind(this)
  6. }
  7. handleClick() {
  8. // ...
  9. }

}

* 1. **Public class fields syntax:** If you don't like to use bind approach then *public class fields syntax* can be used to correctly bind callbacks.
  2. handleClick = () => {
  3. console.log('this is:', this)

}

<button onClick={this.handleClick}>

{'Click me'}

</button>

* 1. **Arrow functions in callbacks:** You can use *arrow functions* directly in the callbacks.
  2. <button onClick={(event) => this.handleClick(event)}>
  3. {'Click me'}

</button>

**Note:** If the callback is passed as prop to child components, those components might do an extra re-rendering. In those cases, it is preferred to go with .bind() or *public class fields syntax* approach considering performance.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to pass a parameter to an event handler or callback?**

You can use an *arrow function* to wrap around an *event handler* and pass parameters:

<button onClick={() => this.handleClick(id)} />

This is an equivalent to calling .bind:

<button onClick={this.handleClick.bind(this, id)} />

Apart from these two approaches, you can also pass arguments to a function which is defined as arrow function

<button onClick={this.handleClick(id)} />

handleClick = (id) => () => {

console.log("Hello, your ticket number is", id)

};

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are synthetic events in React?**

**SyntheticEvent is a cross-browser wrapper around the browser's native event.**

It's API is same as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are inline conditional expressions?**

You can use either *if statements* or *ternary expressions* which are available from JS to conditionally render expressions. Apart from these approaches, you can also embed any expressions in JSX by wrapping them in curly braces and then followed by JS logical operator &&.

<h1>Hello!</h1>

{

messages.length > 0 && !isLogin?

<h2>

You have {messages.length} unread messages.

</h2>

:

<h2>

You don't have unread messages.

</h2>

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

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

A key is a special string attribute you **should** include when creating arrays of elements. *Key* prop helps React identify which items have changed, are added, or are removed.

Most often we use ID from our data as *key*:

const todoItems = todos.map((todo) =>

<li key={todo.id}>

{todo.text}

</li>

)

When you don't have stable IDs for rendered items, you may use the item *index* as a *key* as a last resort:

const todoItems = todos.map((todo, index) =>

<li key={index}>

{todo.text}

</li>

)

**Note:**

* 1. Using *indexes* for *keys* is **not recommended** if the order of items may change. This can negatively impact performance and may cause issues with component state.
  2. If you extract list item as separate component then apply *keys* on list component instead of li tag.
  3. There will be a warning message in the console if the key prop is not present on list items.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the use of refs?**

The *ref* is used to return a reference to the element. They *should be avoided* in most cases, however, they can be useful when you need a direct access to the DOM element or an instance of a component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to create refs?**

There are two approaches

* 1. This is a recently added approach. *Refs* are created using React.createRef() method and attached to React elements via the ref attribute. In order to use *refs* throughout the component, just assign the *ref* to the instance property within constructor.
  2. class MyComponent extends React.Component {
  3. constructor(props) {
  4. super(props)
  5. this.myRef = React.createRef()
  6. }
  7. render() {
  8. return <div ref={this.myRef} />
  9. }

}

* 1. You can also use ref callbacks approach regardless of React version. For example, the search bar component's input element accessed as follows,
  2. class SearchBar extends Component {
  3. constructor(props) {
  4. super(props);
  5. this.txtSearch = null;
  6. this.state = { term: '' };
  7. this.setInputSearchRef = e => {
  8. this.txtSearch = e;
  9. }
  10. }
  11. onInputChange(event) {
  12. this.setState({ term: this.txtSearch.value });
  13. }
  14. render() {
  15. return (
  16. <input
  17. value={this.state.term}
  18. onChange={this.onInputChange.bind(this)}
  19. ref={this.setInputSearchRef} />
  20. );
  21. }

}

You can also use *refs* in function components using **closures**. **Note**: You can also use inline ref callbacks even though it is not a recommended approach

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are forward refs?**

*Ref forwarding* is a feature that lets some components take a *ref* they receive, and pass it further down to a child.

const ButtonElement = React.forwardRef((props, ref) => (

<button ref={ref} className="CustomButton">

{props.children}

</button>

));

// Create ref to the DOM button:

const ref = React.createRef();

<ButtonElement ref={ref}>{'Forward Ref'}</ButtonElement>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Which is preferred option with in callback refs and findDOMNode()?**

It is preferred to use *callback refs* over findDOMNode() API. Because findDOMNode() prevents certain improvements in React in the future.

The **legacy** approach of using findDOMNode:

class MyComponent extends Component {

componentDidMount() {

findDOMNode(this).scrollIntoView()

}

render() {

return <div />

}

}

The recommended approach is:

class MyComponent extends Component {

constructor(props){

super(props);

this.node = createRef();

}

componentDidMount() {

this.node.current.scrollIntoView();

}

render() {

return <div ref={this.node} />

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why are String Refs legacy?**

If you worked with React before, you might be familiar with an older API where the ref attribute is a string, like ref={'textInput'}, and the DOM node is accessed as this.refs.textInput. We advise against it because *string refs have below issues*, and are considered legacy. String refs were **removed in React v16**.

* 1. They *force React to keep track of currently executing component*. This is problematic because it makes react module stateful, and thus causes weird errors when react module is duplicated in the bundle.
  2. They are *not composable* — if a library puts a ref on the passed child, the user can't put another ref on it. Callback refs are perfectly composable.
  3. They *don't work with static analysis* like Flow. Flow can't guess the magic that framework does to make the string ref appear on this.refs, as well as its type (which could be different). Callback refs are friendlier to static analysis.
  4. It doesn't work as most people would expect with the "render callback" pattern (e.g. )
  5. class MyComponent extends Component {
  6. renderRow = (index) => {
  7. // This won't work. Ref will get attached to DataTable rather than MyComponent:
  8. return <input ref={'input-' + index} />;
  9. // This would work though! Callback refs are awesome.
  10. return <input ref={input => this['input-' + index] = input} />;
  11. }
  12. render() {
  13. return <DataTable data={this.props.data} renderRow={this.renderRow} />
  14. }

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Virtual DOM?**

The *Virtual DOM* (VDOM) is an in-memory representation of *Real DOM*. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called *reconciliation*.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How Virtual DOM works?**

The *Virtual DOM* works in three simple steps.

* 1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
  2. Then the difference between the previous DOM representation and the new one is calculated.
  3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between Shadow DOM and Virtual DOM?**

The *Shadow DOM* is a browser technology designed primarily for scoping variables and CSS in *web components*. The *Virtual DOM* is a concept implemented by libraries in JavaScript on top of browser APIs.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React Fiber?**

**Fiber is the new *reconciliation* engine** , Its main feature is **incremental rendering**:

or reimplementation of core algorithm in React v16. The goal of React Fiber is to increase its suitability for areas like animation, layout, gestures, ability to pause, abort, or reuse work and assign priority to different types of updates; and new concurrency primitives.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the main goal of React Fiber?**

Its main feature is **incremental rendering**:

the ability to split rendering work into chunks and spread it out over multiple frames.

*from documentation*

Its main goals are:

* 1. Ability to split interruptible work in chunks.
  2. Ability to prioritize, rebase and reuse work in progress.
  3. Ability to yield back and forth between parents and children to support layout in React.
  4. Ability to return multiple elements from render().
  5. Better support for error boundaries.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are controlled components?**

A component that controls the input elements within the forms on subsequent user input is called **Controlled Component**, i.e, every state mutation will have an associated handler function.

For example, to write all the names in uppercase letters, we use handleChange as below,

handleChange(event) {

this.setState({value: event.target.value.toUpperCase()})

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are uncontrolled components?**

The **Uncontrolled Components** are the ones that store their own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

In the below UserProfile component, the name input is accessed using ref.

class UserProfile extends React.Component {

constructor(props) {

super(props)

this.handleSubmit = this.handleSubmit.bind(this)

this.input = React.createRef()

}

handleSubmit(event) {

alert('A name was submitted: ' + this.input.current.value)

event.preventDefault()

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

{'Name:'}

<input type="text" ref={this.input} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

In most cases, it's recommend to use controlled components to implement forms.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between createElement and cloneElement?**

JSX elements will be transpiled to React.createElement() functions to create React elements which are going to be used for the object representation of UI. Whereas cloneElement is used to clone an element and pass it new props.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Lifting State Up in React?**

**When several components share the same changing data then it is recommended to *lift the shared state up* to their closest common ancestor.**

**That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.**

**What are the different phases of component lifecycle?**

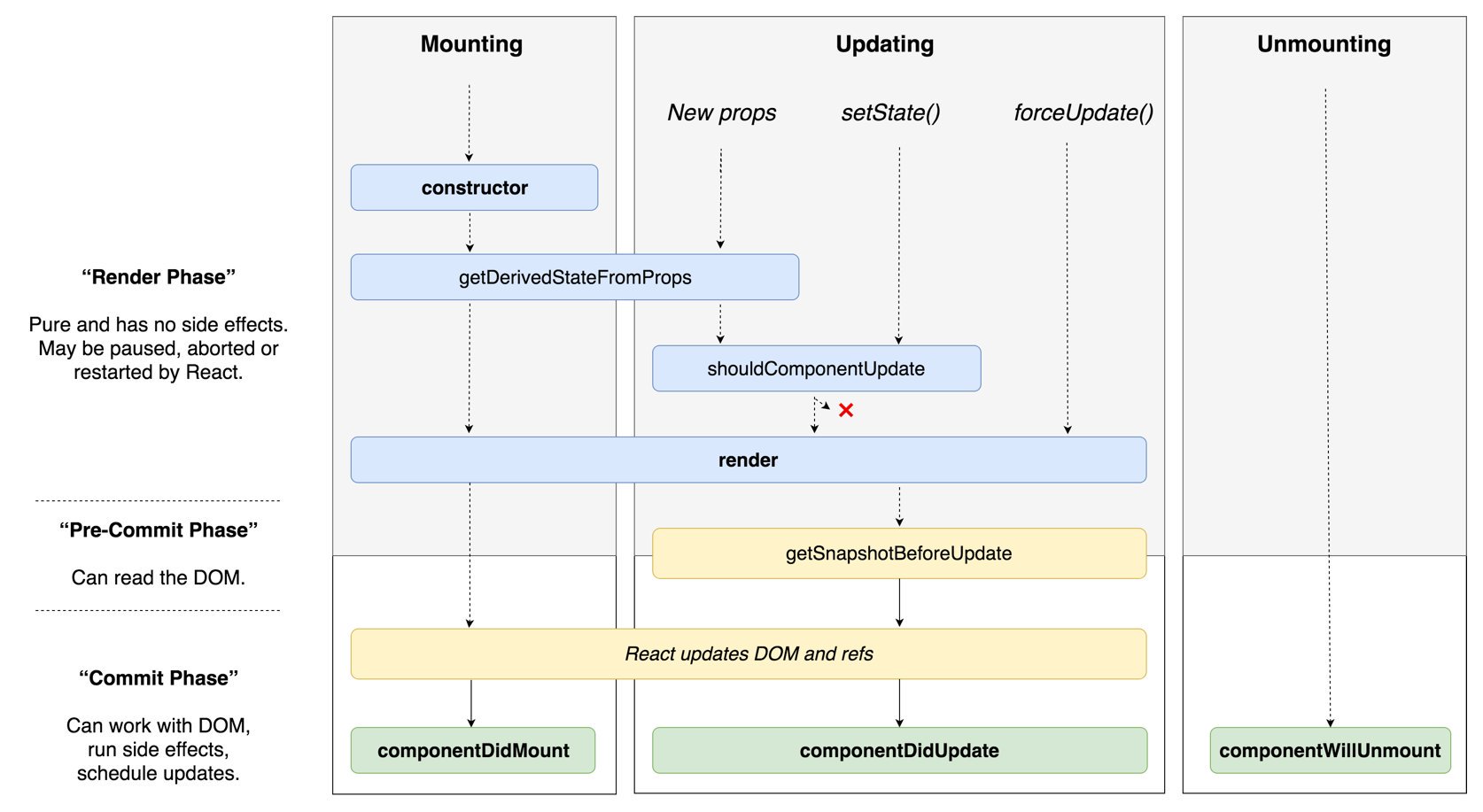
The component lifecycle has three distinct lifecycle phases:

* 1. **Mounting:** The component is ready to mount in the browser DOM. This phase covers initialization from constructor(), getDerivedStateFromProps(), render(), and componentDidMount() lifecycle methods.
  2. **Updating:** In this phase, the component get updated in two ways, sending the new props and updating the state either from setState() or forceUpdate(). This phase covers getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate() and componentDidUpdate() lifecycle methods.
  3. **Unmounting:** In this last phase, the component is not needed and get unmounted from the browser DOM. This phase includes componentWillUnmount() lifecycle method.

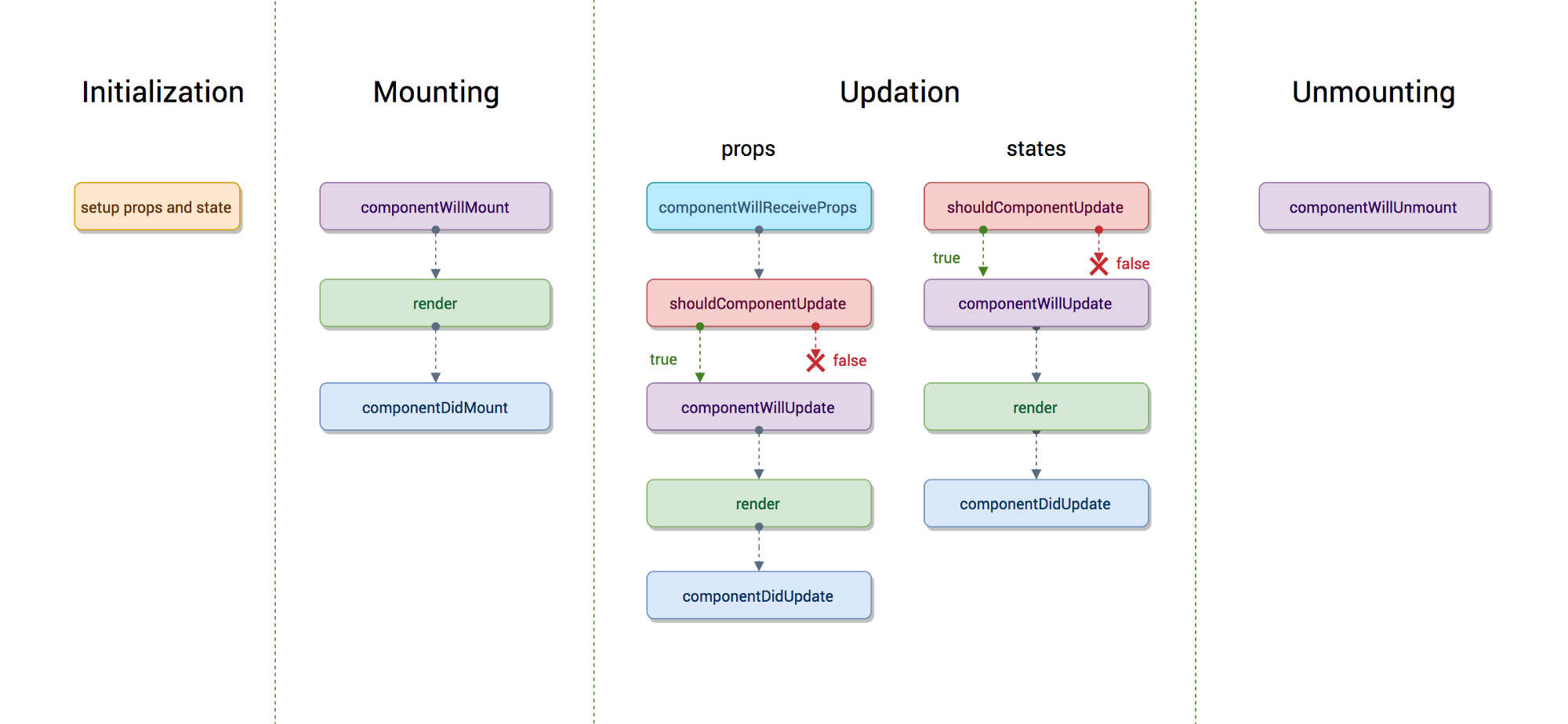
It's worth mentioning that React internally has a concept of phases when applying changes to the DOM. They are separated as follows

* 1. **Render** The component will render without any side-effects. This applies for Pure components and in this phase, React can pause, abort, or restart the render.
  2. **Pre-commit** Before the component actually applies the changes to the DOM, there is a moment that allows React to read from the DOM through the getSnapshotBeforeUpdate().
  3. **Commit** React works with the DOM and executes the final lifecycles respectively componentDidMount() for mounting, componentDidUpdate() for updating, and componentWillUnmount() for unmounting.

React 16.3+ Phases (or an [interactive version](http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/))

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/phases16.3.jpg)

Before React 16.3

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/phases.png)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the lifecycle methods of React?**

Before React 16.3

* + **componentWillMount:** Executed before rendering and is used for App level configuration in your root component.
  + **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
  + **componentWillReceiveProps:** Executed when particular prop updates to trigger state transitions.
  + **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
  + **componentWillUpdate:** Executed before re-rendering the component when there are props & state changes confirmed by shouldComponentUpdate() which returns true.
  + **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes.
  + **componentWillUnmount:** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

React 16.3+

* + **getDerivedStateFromProps:** Invoked right before calling render() and is invoked on *every* render. This exists for rare use cases where you need derived state. Worth reading [if you need derived state](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html).
  + **componentDidMount:** Executed after first rendering and where all AJAX requests, DOM or state updates, and set up event listeners should occur.
  + **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
  + **getSnapshotBeforeUpdate:** Executed right before rendered output is committed to the DOM. Any value returned by this will be passed into componentDidUpdate(). This is useful to capture information from the DOM i.e. scroll position.
  + **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes. This will not fire if shouldComponentUpdate() returns false.
  + **componentWillUnmount** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are Higher-Order Components?**

A *higher-order component* (*HOC*) is a function that takes a component and returns a new component

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is context?**

***It Basically prevents prop drilling***

***Context* provides a way to pass data through the component tree without having to pass props down manually at every level.**

For example, authenticated user, locale preference, UI theme need to be accessed in the application by many components.

const {Provider, Consumer} = React.createContext(defaultValue)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is children prop?**

*Children* is a prop (this.props.children) that allow you to pass components as data to other components, just like any other prop you use. Component tree put between component's opening and closing tag will be passed to that component as children prop.

There are a number of methods available in the React API to work with this prop. These include React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray.

A simple usage of children prop looks as below,

const MyDiv = React.createClass({

render: function() {

return <div>{this.props.children}</div>

}

})

ReactDOM.render(

<MyDiv>

<span>{'Hello'}</span>

<span>{'World'}</span>

</MyDiv>,

node

)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to write comments in React?**

The comments in React/JSX are similar to JavaScript Multiline comments but are wrapped in curly braces.

**Single-line comments:**

<div>

{/\* Single-line comments(In vanilla JavaScript, the single-line comments are represented by double slash(//)) \*/}

{`Welcome ${user}, let's play React`}

</div>

**Multi-line comments:**

<div>

{/\* Multi-line comments for more than

one line \*/}

{`Welcome ${user}, let's play React`}

</div>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of using super constructor with props argument?**

A child class constructor cannot make use of this reference until super() method has been called. The same applies for ES6 sub-classes as well. The main reason of passing props parameter to super() call is to access this.props in your child constructors.

**Passing props:**

class MyComponent extends React.Component {

constructor(props) {

super(props)

console.log(this.props) // prints { name: 'John', age: 42 }

}

}

**Not passing props:**

class MyComponent extends React.Component {

constructor(props) {

super()

console.log(this.props) // prints undefined

// but props parameter is still available

console.log(props) // prints { name: 'John', age: 42 }

}

render() {

// no difference outside constructor

console.log(this.props) // prints { name: 'John', age: 42 }

}

}

The above code snippets reveals that this.props is different only within the constructor. It would be the same outside the constructor.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is reconciliation?**

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called *reconciliation*.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to set state with a dynamic key name?**

If you are using ES6 or the Babel transpiler to transform your JSX code then you can accomplish this with *computed property names*.

handleInputChange(event) {

this.setState({ [event.target.id]: event.target.value })

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What would be the common mistake of function being called every time the component renders?**

You need to make sure that function is not being called while passing the function as a parameter.

render() {

// Wrong: handleClick is called instead of passed as a reference!

return <button onClick={this.handleClick()}>{'Click Me'}</button>

}

Instead, pass the function itself without parenthesis:

render() {

// Correct: handleClick is passed as a reference!

return <button onClick={this.handleClick}>{'Click Me'}</button>

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is lazy function supports named exports?**

No, currently React.lazy function supports default exports only. If you would like to import modules which are named exports, you can create an intermediate module that reexports it as the default. It also ensures that tree shaking keeps working and don’t pull unused components. Let's take a component file which exports multiple named components,

// MoreComponents.js

export const SomeComponent = /\* ... \*/;

export const UnusedComponent = /\* ... \*/;

and reexport MoreComponents.js components in an intermediate file IntermediateComponent.js

// IntermediateComponent.js

export { SomeComponent as default } from "./MoreComponents.js";

Now you can import the module using lazy function as below,

import React, { lazy } from 'react';

const SomeComponent = lazy(() => import("./IntermediateComponent.js"));

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why React uses className over class attribute?**

class is a keyword in JavaScript, and JSX is an extension of JavaScript. That's the principal reason why React uses className instead of class. Pass a string as the className prop.

render() {

return <span className={'menu navigation-menu'}>{'Menu'}</span>

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are fragments?**

It's common pattern in React which is used for a component to return multiple elements. *Fragments* let you group a list of children without adding extra nodes to the DOM.

render() {

return (

<React.Fragment>

<ChildA />

<ChildB />

<ChildC />

</React.Fragment>

)

}

There is also a *shorter syntax*, but it's not supported in many tools:

render() {

return (

<>

<ChildA />

<ChildB />

<ChildC />

</>

)

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why fragments are better than container divs?**

Below are the list of reasons,

* 1. Fragments are a bit faster and use less memory by not creating an extra DOM node. This only has a real benefit on very large and deep trees.
  2. Some CSS mechanisms like *Flexbox* and *CSS Grid* have a special parent-child relationships, and adding divs in the middle makes it hard to keep the desired layout.
  3. The DOM Inspector is less cluttered.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are portals in React?**

***Portal* is a recommended way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.**

ReactDOM.createPortal(child, container)

The first argument is any render-able React child, such as an element, string, or fragment. The second argument is a DOM element.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are stateless components?**

If the behaviour is independent of its state then it can be a stateless component. You can use either a function or a class for creating stateless components. But unless you need to use a lifecycle hook in your components, you should go for function components. There are a lot of benefits if you decide to use function components here; they are easy to write, understand, and test, a little faster, and you can avoid the this keyword altogether.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are stateful components?**

If the behaviour of a component is dependent on the *state* of the component then it can be termed as stateful component. These *stateful components* are always *class components* and have a state that gets initialized in the constructor.

class App extends Component {

constructor(props) {

super(props)

this.state = { count: 0 }

}

render() {

// ...

}

}

**React 16.8 Update:**

Hooks let you use state and other React features without writing classes.

*The Equivalent Functional Component*

import React, {useState} from 'react';

const App = (props) => {

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

return (

// JSX

)

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to apply validation on props in React?**

When the application is running in *development mode*, React will automatically check all props that we set on components to make sure they have *correct type*. If the type is incorrect, React will generate warning messages in the console. It's disabled in *production mode* due to performance impact. The mandatory props are defined with isRequired.

The set of predefined prop types:

* 1. PropTypes.number
  2. PropTypes.string
  3. PropTypes.array
  4. PropTypes.object
  5. PropTypes.func
  6. PropTypes.node
  7. PropTypes.element
  8. PropTypes.bool
  9. PropTypes.symbol
  10. PropTypes.any

We can define propTypes for User component as below:

import React from 'react'

import PropTypes from 'prop-types'

class User extends React.Component {

static propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

}

render() {

return (

<>

<h1>{`Welcome, ${this.props.name}`}</h1>

<h2>{`Age, ${this.props.age}`}</h2>

</>

)

}

}

**Note:** In React v15.5 *PropTypes* were moved from React.PropTypes to prop-types library.

*The Equivalent Functional Component*

import React from 'react'

import PropTypes from 'prop-types'

function User() {

return (

<>

<h1>{`Welcome, ${this.props.name}`}</h1>

<h2>{`Age, ${this.props.age}`}</h2>

</>

)

}

User.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

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

Below are the list of main advantages of React,

* 1. Increases the application's performance with *Virtual DOM*.
  2. JSX makes code easy to read and write.
  3. It renders both on client and server side (*SSR*).
  4. Easy to integrate with frameworks (Angular, Backbone) since it is only a view library.
  5. Easy to write unit and integration tests with tools such as Jest.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the limitations of React?**

Apart from the advantages, there are few limitations of React too,

* 1. React is just a view library, not a full framework.
  2. There is a learning curve for beginners who are new to web development.
  3. Integrating React into a traditional MVC framework requires some additional configuration.
  4. The code complexity increases with inline templating and JSX.
  5. Too many smaller components leading to over engineering or boilerplate.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are error boundaries in React v16?**

*Error boundaries* are components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

A class component becomes an error boundary if it defines a new lifecycle method called componentDidCatch(error, info) or static getDerivedStateFromError() :

class ErrorBoundary extends React.Component {

constructor(props) {

super(props)

this.state = { hasError: false }

}

componentDidCatch(error, info) {

// You can also log the error to an error reporting service

logErrorToMyService(error, info)

}

static getDerivedStateFromError(error) {

// Update state so the next render will show the fallback UI.

return { hasError: true };

}

render() {

if (this.state.hasError) {

// You can render any custom fallback UI

return <h1>{'Something went wrong.'}</h1>

}

return this.props.children

}

}

After that use it as a regular component:

<ErrorBoundary>

<MyWidget />

</ErrorBoundary>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How error boundaries handled in React v15?**

React v15 provided very basic support for *error boundaries* using unstable\_handleError method. It has been renamed to componentDidCatch in React v16.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the recommended ways for static type checking?**

Normally we use *PropTypes library* (React.PropTypes moved to a prop-types package since React v15.5) for *type checking* in the React applications. For large code bases, it is recommended to use *static type checkers* such as Flow or TypeScript, that perform type checking at compile time and provide auto-completion features.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the use of react-dom package?**

The react-dom package provides *DOM-specific methods* that can be used at the top level of your app. Most of the components are not required to use this module. Some of the methods of this package are:

* 1. render()
  2. hydrate()
  3. unmountComponentAtNode()
  4. findDOMNode()
  5. createPortal()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of render method of react-dom?**

This method is used to render a React element into the DOM in the supplied container and return a reference to the component. If the React element was previously rendered into container, it will perform an update on it and only mutate the DOM as necessary to reflect the latest changes.

ReactDOM.render(element, container[, callback])

If the optional callback is provided, it will be executed after the component is rendered or updated.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is ReactDOMServer?**

The ReactDOMServer object enables you to render components to static markup (typically used on node server). This object is mainly used for *server-side rendering* (SSR). The following methods can be used in both the server and browser environments:

* 1. renderToString()
  2. renderToStaticMarkup()

For example, you generally run a Node-based web server like Express, Hapi, or Koa, and you call renderToString to render your root component to a string, which you then send as response.

// using Express

import { renderToString } from 'react-dom/server'

import MyPage from './MyPage'

app.get('/', (req, res) => {

res.write('<!DOCTYPE html><html><head><title>My Page</title></head><body>')

res.write('<div id="content">')

res.write(renderToString(<MyPage/>))

res.write('</div></body></html>')

res.end()

})

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use innerHTML in React?**

The dangerouslySetInnerHTML attribute is React's replacement for using innerHTML in the browser DOM. Just like innerHTML, it is risky to use this attribute considering cross-site scripting (XSS) attacks. You just need to pass a \_\_html object as key and HTML text as value.

In this example MyComponent uses dangerouslySetInnerHTML attribute for setting HTML markup:

function createMarkup() {

return { \_\_html: 'First &middot; Second' }

}

function MyComponent() {

return <div dangerouslySetInnerHTML={createMarkup()} />

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use styles in React?**

The style attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM style JavaScript property, is more efficient, and prevents XSS security holes.

const divStyle = {

color: 'blue',

backgroundImage: 'url(' + imgUrl + ')'

};

function HelloWorldComponent() {

return <div style={divStyle}>Hello World!</div>

}

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes in JavaScript (e.g. node.style.backgroundImage).

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How events are different in React?**

Handling events in React elements has some syntactic differences:

* 1. React event handlers are named using camelCase, rather than lowercase.
  2. With JSX you pass a function as the event handler, rather than a string.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

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

When you use setState(), then apart from assigning to the object state React also re-renders the component and all its children. You would get error like this: *Can only update a mounted or mounting component.* So we need to use this.state to initialize variables inside constructor.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the impact of indexes as keys?**

Keys should be stable, predictable, and unique so that React can keep track of elements.

In the below code snippet each element's key will be based on ordering, rather than tied to the data that is being represented. This limits the optimizations that React can do.

{todos.map((todo, index) =>

<Todo

{...todo}

key={index}

/>

)}

If you use element data for unique key, assuming todo.id is unique to this list and stable, React would be able to reorder elements without needing to reevaluate them as much.

{todos.map((todo) =>

<Todo {...todo}

key={todo.id} />

)}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it good to use setState() in componentWillMount() method?**

Yes, it is safe to use setState() inside componentWillMount() method. But at the same it is recommended to avoid async initialization in componentWillMount() lifecycle method. componentWillMount() is invoked immediately before mounting occurs. It is called before render(), therefore setting state in this method will not trigger a re-render. Avoid introducing any side-effects or subscriptions in this method. We need to make sure async calls for component initialization happened in componentDidMount() instead of componentWillMount().

componentDidMount() {

axios.get(`api/todos`)

.then((result) => {

this.setState({

messages: [...result.data]

})

})

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What will happen if you use props in initial state?**

If the props on the component are changed without the component being refreshed, the new prop value will never be displayed because the constructor function will never update the current state of the component. The initialization of state from props only runs when the component is first created.

The below component won't display the updated input value:

class MyComponent extends React.Component {

constructor(props) {

super(props)

this.state = {

records: [],

inputValue: this.props.inputValue

};

}

render() {

return <div>{this.state.inputValue}</div>

}

}

Using props inside render method will update the value:

class MyComponent extends React.Component {

constructor(props) {

super(props)

this.state = {

record: []

}

}

render() {

return <div>{this.props.inputValue}</div>

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you conditionally render components?**

In some cases you want to render different components depending on some state. JSX does not render false or undefined, so you can use conditional *short-circuiting* to render a given part of your component only if a certain condition is true.

const MyComponent = ({ name, address }) => (

<div>

<h2>{name}</h2>

{address &&

<p>{address}</p>

}

</div>

)

If you need an if-else condition then use *ternary operator*.

const MyComponent = ({ name, address }) => (

<div>

<h2>{name}</h2>

{address

? <p>{address}</p>

: <p>{'Address is not available'}</p>

}

</div>

)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why we need to be careful when spreading props on DOM elements?**

When we *spread props* we run into the risk of adding unknown HTML attributes, which is a bad practice. Instead we can use prop destructuring with ...rest operator, so it will add only required props.

For example,

const ComponentA = () =>

<ComponentB isDisplay={true} className={'componentStyle'} />

const ComponentB = ({ isDisplay, ...domProps }) =>

<div {...domProps}>{'ComponentB'}</div>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How you use decorators in React?**

You can *decorate* your *class* components, which is the same as passing the component into a function. **Decorators** are flexible and readable way of modifying component functionality.

@setTitle('Profile')

class Profile extends React.Component {

//....

}

/\*

title is a string that will be set as a document title

WrappedComponent is what our decorator will receive when

put directly above a component class as seen in the example above

\*/

const setTitle = (title) => (WrappedComponent) => {

return class extends React.Component {

componentDidMount() {

document.title = title

}

render() {

return <WrappedComponent {...this.props} />

}

}

}

**Note:** Decorators are a feature that didn't make it into ES7, but are currently a *stage 2 proposal*.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you memoize a component?**

There are memoize libraries available which can be used on function components.

For example moize library can memoize the component in another component.

import moize from 'moize'

import Component from './components/Component' // this module exports a non-memoized component

const MemoizedFoo = moize.react(Component)

const Consumer = () => {

<div>

{'I will memoize the following entry:'}

<MemoizedFoo/>

</div>

}

**Update:** Since React v16.6.0, we have a React.memo. It provides a higher order component which memoizes component unless the props change. To use it, simply wrap the component using React.memo before you use it.

const MemoComponent = React.memo(function MemoComponent(props) {

/\* render using props \*/

});

OR

export default React.memo(MyFunctionComponent);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How you implement Server Side Rendering or SSR?**

React is already equipped to handle rendering on Node servers. A special version of the DOM renderer is available, which follows the same pattern as on the client side.

import ReactDOMServer from 'react-dom/server'

import App from './App'

ReactDOMServer.renderToString(<App />)

This method will output the regular HTML as a string, which can be then placed inside a page body as part of the server response. On the client side, React detects the pre-rendered content and seamlessly picks up where it left off.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to enable production mode in React?**

You should use Webpack's DefinePlugin method to set NODE\_ENV to production, by which it strip out things like propType validation and extra warnings. Apart from this, if you minify the code, for example, Uglify's dead-code elimination to strip out development only code and comments, it will drastically reduce the size of your bundle.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is CRA and its benefits?**

The create-react-app CLI tool allows you to quickly create & run React applications with no configuration step.

Let's create Todo App using *CRA*:

# Installation

$ npm install -g create-react-app

# Create new project

$ create-react-app todo-app

$ cd todo-app

# Build, test and run

$ npm run build

$ npm run test

$ npm start

It includes everything we need to build a React app:

* 1. React, JSX, ES6, and Flow syntax support.
  2. Language extras beyond ES6 like the object spread operator.
  3. Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
  4. A fast interactive unit test runner with built-in support for coverage reporting.
  5. A live development server that warns about common mistakes.
  6. A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the lifecycle methods order in mounting?**

The lifecycle methods are called in the following order when an instance of a component is being created and inserted into the DOM.

* 1. constructor()
  2. static getDerivedStateFromProps()
  3. render()
  4. componentDidMount()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the lifecycle methods going to be deprecated in React v16?**

The following lifecycle methods going to be unsafe coding practices and will be more problematic with async rendering.

* 1. componentWillMount()
  2. componentWillReceiveProps()
  3. componentWillUpdate()

Starting with React v16.3 these methods are aliased with UNSAFE\_ prefix, and the unprefixed version will be removed in React v17.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of getDerivedStateFromProps() lifecycle method?**

The new static getDerivedStateFromProps() lifecycle method is invoked after a component is instantiated as well as before it is re-rendered. It can return an object to update state, or null to indicate that the new props do not require any state updates.

class MyComponent extends React.Component {

static getDerivedStateFromProps(props, state) {

// ...

}

}

This lifecycle method along with componentDidUpdate() covers all the use cases of componentWillReceiveProps().

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of getSnapshotBeforeUpdate() lifecycle method?**

The new getSnapshotBeforeUpdate() lifecycle method is called right before DOM updates. The return value from this method will be passed as the third parameter to componentDidUpdate().

class MyComponent extends React.Component {

getSnapshotBeforeUpdate(prevProps, prevState) {

// ...

}

}

This lifecycle method along with componentDidUpdate() covers all the use cases of componentWillUpdate().

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Do Hooks replace render props and higher order components?**

Both render props and higher-order components render only a single child but in most of the cases Hooks are a simpler way to serve this by reducing nesting in your tree.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the recommended way for naming components?**

It is recommended to name the component by reference instead of using displayName.

Using displayName for naming component:

export default React.createClass({

displayName: 'TodoApp',

// ...

})

The **recommended** approach:

export default class TodoApp extends React.Component {

// ...

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the recommended ordering of methods in component class?**

*Recommended* ordering of methods from *mounting* to *render stage*:

* 1. static methods
  2. constructor()
  3. getChildContext()
  4. componentWillMount()
  5. componentDidMount()
  6. componentWillReceiveProps()
  7. shouldComponentUpdate()
  8. componentWillUpdate()
  9. componentDidUpdate()
  10. componentWillUnmount()
  11. click handlers or event handlers like onClickSubmit() or onChangeDescription()
  12. getter methods for render like getSelectReason() or getFooterContent()
  13. optional render methods like renderNavigation() or renderProfilePicture()
  14. render()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is a switching component?**

A *switching component* is a component that renders one of many components. We need to use object to map prop values to components.

For example, a switching component to display different pages based on page prop:

import HomePage from './HomePage'

import AboutPage from './AboutPage'

import ServicesPage from './ServicesPage'

import ContactPage from './ContactPage'

const PAGES = {

home: HomePage,

about: AboutPage,

services: ServicesPage,

contact: ContactPage

}

const Page = (props) => {

const Handler = PAGES[props.page] || ContactPage

return <Handler {...props} />

}

// The keys of the PAGES object can be used in the prop types to catch dev-time errors.

Page.propTypes = {

page: PropTypes.oneOf(Object.keys(PAGES)).isRequired

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why we need to pass a function to setState()?**

The reason behind for this is that setState() is an asynchronous operation. React batches state changes for performance reasons, so the state may not change immediately after setState() is called. That means you should not rely on the current state when calling setState() since you can't be sure what that state will be. The solution is to pass a function to setState(), with the previous state as an argument. By doing this you can avoid issues with the user getting the old state value on access due to the asynchronous nature of setState().

Let's say the initial count value is zero. After three consecutive increment operations, the value is going to be incremented only by one.

// assuming this.state.count === 0

this.setState({ count: this.state.count + 1 })

this.setState({ count: this.state.count + 1 })

this.setState({ count: this.state.count + 1 })

// this.state.count === 0, not 3

If we pass a function to setState(), the count gets incremented correctly.

this.setState((prevState, props) => ({

count: prevState.count + props.increment

}))

// this.state.count === 3 as expected

**(OR)**

**Why function is preferred over object for setState()?**

React may batch multiple setState() calls into a single update for performance. Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

This counter example will fail to update as expected:

// Wrong

this.setState({

counter: this.state.counter + this.props.increment,

})

The preferred approach is to call setState() with function rather than object. That function will receive the previous state as the first argument, and the props at the time the update is applied as the second argument.

// Correct

this.setState((prevState, props) => ({

counter: prevState.counter + props.increment

}))

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is strict mode in React?**

React.StrictMode is a useful component for highlighting potential problems in an application. Just like <Fragment>, <StrictMode> does not render any extra DOM elements. It activates additional checks and warnings for its descendants. These checks apply for *development mode* only.

import React from 'react'

function ExampleApplication() {

return (

<div>

<Header />

<React.StrictMode>

<div>

<ComponentOne />

<ComponentTwo />

</div>

</React.StrictMode>

<Header />

</div>

)

}

In the example above, the *strict mode* checks apply to <ComponentOne> and <ComponentTwo> components only.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are React Mixins?**

*Mixins* are a way to totally separate components to have a common functionality. Mixins **should not be used** and can be replaced with *higher-order components* or *decorators*.

One of the most commonly used mixins is PureRenderMixin. You might be using it in some components to prevent unnecessary re-renders when the props and state are shallowly equal to the previous props and state:

const PureRenderMixin = require('react-addons-pure-render-mixin')

const Button = React.createClass({

mixins: [PureRenderMixin],

// ...

})

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why is isMounted() an anti-pattern and what is the proper solution?**

The primary use case for isMounted() is to avoid calling setState() after a component has been unmounted, because it will emit a warning.

if (this.isMounted()) {

this.setState({...})

}

Checking isMounted() before calling setState() does eliminate the warning, but it also defeats the purpose of the warning. Using isMounted() is a code smell because the only reason you would check is because you think you might be holding a reference after the component has unmounted.

An optimal solution would be to find places where setState() might be called after a component has unmounted, and fix them. Such situations most commonly occur due to callbacks, when a component is waiting for some data and gets unmounted before the data arrives. Ideally, any callbacks should be canceled in componentWillUnmount(), prior to unmounting.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the Pointer Events supported in React?**

*Pointer Events* provide a unified way of handling all input events. In the old days we had a mouse and respective event listeners to handle them but nowadays we have many devices which don't correlate to having a mouse, like phones with touch surface or pens. We need to remember that these events will only work in browsers that support the *Pointer Events* specification.

The following event types are now available in *React DOM*:

* 1. onPointerDown
  2. onPointerMove
  3. onPointerUp
  4. onPointerCancel
  5. onGotPointerCapture
  6. onLostPointerCapture
  7. onPointerEnter
  8. onPointerLeave
  9. onPointerOver
  10. onPointerOut

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why should component names start with capital letter?**

If you are rendering your component using JSX, the name of that component has to begin with a capital letter otherwise React will throw an error as unrecognized tag. This convention is because only HTML elements and SVG tags can begin with a lowercase letter.

class SomeComponent extends Component {

// Code goes here

}

You can define component class which name starts with lowercase letter, but when it's imported it should have capital letter. Here lowercase is fine:

class myComponent extends Component {

render() {

return <div />

}

}

export default myComponent

While when imported in another file it should start with capital letter:

import MyComponent from './MyComponent'

**What are the exceptions on React component naming?**

The component names should start with a uppercase letter but there are few exceptions on this convention. The lowercase tag names with a dot (property accessors) are still considered as valid component names. For example the below tag can be compiled to a valid component,

render() {

return (

<obj.component/> // `React.createElement(obj.component)`

)

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Are custom DOM attributes supported in React v16?**

Yes. In the past, React used to ignore unknown DOM attributes. If you wrote JSX with an attribute that React doesn't recognize, React would just skip it.

For example, let's take a look at the below attribute:

<div mycustomattribute={'something'} />

Would render an empty div to the DOM with React v15:

<div />

In React v16 any unknown attributes will end up in the DOM:

<div mycustomattribute='something' />

This is useful for supplying browser-specific non-standard attributes, trying new DOM APIs, and integrating with opinionated third-party libraries.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between constructor and getInitialState?**

You should initialize state in the constructor when using ES6 classes, and getInitialState() method when using React.createClass().

**Using ES6 classes:**

class MyComponent extends React.Component {

constructor(props) {

super(props)

this.state = { /\* initial state \*/ }

}

}

**Using React.createClass():**

const MyComponent = React.createClass({

getInitialState() {

return { /\* initial state \*/ }

}

})

**Note:** React.createClass() is deprecated and removed in React v16. Use plain JavaScript classes instead.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can you force a component to re-render without calling setState?**

By default, when your component's state or props change, your component will re-render. If your render() method depends on some other data, you can tell React that the component needs re-rendering by calling forceUpdate().

component.forceUpdate(callback)

It is recommended to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between super() and super(props) in React using ES6 classes?**

When you want to access this.props in constructor() then you should pass props to super() method.

**Using super(props):**

class MyComponent extends React.Component {

constructor(props) {

super(props)

console.log(this.props) // { name: 'John', ... }

}

}

**Using super():**

class MyComponent extends React.Component {

constructor(props) {

super()

console.log(this.props) // undefined

}

}

Outside constructor() both will display same value for this.props.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to loop inside JSX?**

You can simply use Array.prototype.map with ES6 *arrow function* syntax.

For example, the items array of objects is mapped into an array of components:

<tbody>

{items.map(item => <SomeComponent key={item.id} name={item.name} />)}

</tbody>

But you can't iterate using for loop:

<tbody>

for (let i = 0; i < items.length; i++) {

<SomeComponent key={items[i].id} name={items[i].name} />

}

</tbody>

This is because JSX tags are transpiled into *function calls*, and you can't use statements inside expressions. This may change thanks to do expressions which are *stage 1 proposal*.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you access props in attribute quotes?**

React (or JSX) doesn't support variable interpolation inside an attribute value. The below representation won't work:

<img className='image' src='images/{this.props.image}' />

But you can put any JS expression inside curly braces as the entire attribute value. So the below expression works:

<img className='image' src={'images/' + this.props.image} />

Using *template strings* will also work:

<img className='image' src={`images/${this.props.image}`} />

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React proptype array with shape?**

If you want to pass an array of objects to a component with a particular shape then use React.PropTypes.shape() as an argument to React.PropTypes.arrayOf().

ReactComponent.propTypes = {

arrayWithShape: React.PropTypes.arrayOf(React.PropTypes.shape({

color: React.PropTypes.string.isRequired,

fontSize: React.PropTypes.number.isRequired

})).isRequired

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to conditionally apply class attributes?**

You shouldn't use curly braces inside quotes because it is going to be evaluated as a string.

<div className="btn-panel {this.props.visible ? 'show' : 'hidden'}">

Instead you need to move curly braces outside (don't forget to include spaces between class names):

<div className={'btn-panel ' + (this.props.visible ? 'show' : 'hidden')}>

*Template strings* will also work:

<div className={`btn-panel ${this.props.visible ? 'show' : 'hidden'}`}>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between React and ReactDOM?**

The react package contains React.createElement(), React.Component, React.Children, and other helpers related to elements and component classes. You can think of these as the isomorphic or universal helpers that you need to build components. The react-dom package contains ReactDOM.render(), and in react-dom/server we have *server-side rendering* support with ReactDOMServer.renderToString() and ReactDOMServer.renderToStaticMarkup().

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why ReactDOM is separated from React?**

The React team worked on extracting all DOM-related features into a separate library called *ReactDOM*. React v0.14 is the first release in which the libraries are split. By looking at some of the packages, react-native, react-art, react-canvas, and react-three, it has become clear that the beauty and essence of React has nothing to do with browsers or the DOM.

To build more environments that React can render to, React team planned to split the main React package into two: react and react-dom. This paves the way to writing components that can be shared between the web version of React and React Native.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use React label element?**

If you try to render a <label> element bound to a text input using the standard for attribute, then it produces HTML missing that attribute and prints a warning to the console.

<label for={'user'}>{'User'}</label>

<input type={'text'} id={'user'} />

Since for is a reserved keyword in JavaScript, use htmlFor instead.

<label htmlFor={'user'}>{'User'}</label>

<input type={'text'} id={'user'} />

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to combine multiple inline style objects?**

You can use *spread operator* in regular React:

<button style={{...styles.panel.button, ...styles.panel.submitButton}}>{'Submit'}</button>

If you're using React Native then you can use the array notation:

<button style={[styles.panel.button, styles.panel.submitButton]}>{'Submit'}</button>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to re-render the view when the browser is resized?**

You can listen to the resize event in componentDidMount() and then update the dimensions (width and height). You should remove the listener in componentWillUnmount() method.

class WindowDimensions extends React.Component {

constructor(props){

super(props);

this.updateDimensions = this.updateDimensions.bind(this);

}

componentWillMount() {

this.updateDimensions()

}

componentDidMount() {

window.addEventListener('resize', this.updateDimensions)

}

componentWillUnmount() {

window.removeEventListener('resize', this.updateDimensions)

}

updateDimensions() {

this.setState({width: window.innerWidth, height: window.innerHeight})

}

render() {

return <span>{this.state.width} x {this.state.height}</span>

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between setState() and replaceState() methods?**

When you use setState() the current and previous states are merged. replaceState() throws out the current state, and replaces it with only what you provide. Usually setState() is used unless you really need to remove all previous keys for some reason. You can also set state to false/null in setState() instead of using replaceState().

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to listen to state changes?**

The componentDidUpdate lifecycle method will be called when state changes. You can compare provided state and props values with current state and props to determine if something meaningful changed.

componentDidUpdate(object prevProps, object prevState)

**Note:** The previous releases of ReactJS also uses componentWillUpdate(object nextProps, object nextState) for state changes. It has been deprecated in latest releases.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the recommended approach of removing an array element in React state?**

The better approach is to use Array.prototype.filter() method.

For example, let's create a removeItem() method for updating the state.

removeItem(index) {

this.setState({

data: this.state.data.filter((item, i) => i !== index)

})

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it possible to use React without rendering HTML?**

It is possible with latest version (>=16.2). Below are the possible options:

render() {

return false

}

render() {

return null

}

render() {

return []

}

render() {

return <React.Fragment></React.Fragment>

}

render() {

return <></>

}

Returning undefined won't work.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to pretty print JSON with React?**

We can use <pre> tag so that the formatting of the JSON.stringify() is retained:

const data = { name: 'John', age: 42 }

class User extends React.Component {

render() {

return (

<pre>

{JSON.stringify(data, null, 2)}

</pre>

)

}

}

React.render(<User />, document.getElementById('container'))

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why you can't update props in React?**

The React philosophy is that props should be *immutable* and *top-down*. This means that a parent can send any prop values to a child, but the child can't modify received props.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to focus an input element on page load?**

You can do it by creating *ref* for input element and using it in componentDidMount():

class App extends React.Component{

componentDidMount() {

this.nameInput.focus()

}

render() {

return (

<div>

<input

defaultValue={'Won\'t focus'}

/>

<input

ref={(input) => this.nameInput = input}

defaultValue={'Will focus'}

/>

</div>

)

}

}

ReactDOM.render(<App />, document.getElementById('app'))

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the possible ways of updating objects in state?**
   1. **Calling setState() with an object to merge with state:**
      * Using Object.assign() to create a copy of the object:
      * const user = Object.assign({}, this.state.user, { age: 42 })

this.setState({ user })

* + - Using *spread operator*:
    - const user = { ...this.state.user, age: 42 }

this.setState({ user })

* 1. **Calling setState() with a function:**
  2. this.setState(prevState => ({
  3. user: {
  4. ...prevState.user,
  5. age: 42
  6. }

}))

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How can we find the version of React at runtime in the browser?**

You can use React.version to get the version.

const REACT\_VERSION = React.version

ReactDOM.render(

<div>{`React version: ${REACT\_VERSION}`}</div>,

document.getElementById('app')

)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the approaches to include polyfills in your create-react-app?**

There are approaches to include polyfills in create-react-app,

* 1. **Manual import from core-js:**

Create a file called (something like) polyfills.js and import it into root index.js file. Run npm install core-js or yarn add core-js and import your specific required features.

import 'core-js/fn/array/find'

import 'core-js/fn/array/includes'

import 'core-js/fn/number/is-nan'

* 1. **Using Polyfill service:**

Use the polyfill.io CDN to retrieve custom, browser-specific polyfills by adding this line to index.html:

<script src='https://cdn.polyfill.io/v2/polyfill.min.js?features=default,Array.prototype.includes'></script>

In the above script we had to explicitly request the Array.prototype.includes feature as it is not included in the default feature set.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use https instead of http in create-react-app?**

You just need to use HTTPS=true configuration. You can edit your package.json scripts section:

"scripts": {

"start": "set HTTPS=true && react-scripts start"

}

or just run set HTTPS=true && npm start

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to avoid using relative path imports in create-react-app?**

Create a file called .env in the project root and write the import path:

NODE\_PATH=src/app

After that restart the development server. Now you should be able to import anything inside src/app without relative paths.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to add Google Analytics for React Router?**

Add a listener on the history object to record each page view:

history.listen(function (location) {

window.ga('set', 'page', location.pathname + location.search)

window.ga('send', 'pageview', location.pathname + location.search)

})

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to update a component every second?**

You need to use setInterval() to trigger the change, but you also need to clear the timer when the component unmounts to prevent errors and memory leaks.

componentDidMount() {

this.interval = setInterval(() => this.setState({ time: Date.now() }), 1000)

}

componentWillUnmount() {

clearInterval(this.interval)

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you apply vendor prefixes to inline styles in React?**

React *does not* apply *vendor prefixes* automatically. You need to add vendor prefixes manually.

<div style={{

transform: 'rotate(90deg)',

WebkitTransform: 'rotate(90deg)', // note the capital 'W' here

msTransform: 'rotate(90deg)' // 'ms' is the only lowercase vendor prefix

}} />

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to import and export components using React and ES6?**

You should use default for exporting the components

import React from 'react'

import User from 'user'

export default class MyProfile extends React.Component {

render(){

return (

<User type="customer">

//...

</User>

)

}

}

With the export specifier, the MyProfile is going to be the member and exported to this module and the same can be imported without mentioning the name in other components.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why is a component constructor called only once?**

React's *reconciliation* algorithm assumes that without any information to the contrary, if a custom component appears in the same place on subsequent renders, it's the same component as before, so reuses the previous instance rather than creating a new one.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to define constants in React?**

You can use ES7 static field to define constant.

class MyComponent extends React.Component {

static DEFAULT\_PAGINATION = 10

}

*Static fields* are part of the *Class Fields* stage 3 proposal.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to programmatically trigger click event in React?**

You could use the ref prop to acquire a reference to the underlying HTMLInputElement object through a callback, store the reference as a class property, then use that reference to later trigger a click from your event handlers using the HTMLElement.click method.

This can be done in two steps:

* 1. Create ref in render method:

<input ref={input => this.inputElement = input} />

* 1. Apply click event in your event handler:

this.inputElement.click()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it possible to use async/await in plain React?**

If you want to use async/await in React, you will need *Babel* and [transform-async-to-generator](https://babeljs.io/docs/en/babel-plugin-transform-async-to-generator) plugin. React Native ships with Babel and a set of transforms.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the common folder structures for React?**

There are two common practices for React project file structure.

* 1. **Grouping by features or routes:**

One common way to structure projects is locate CSS, JS, and tests together, grouped by feature or route.

common/

├─ Avatar.js

├─ Avatar.css

├─ APIUtils.js

└─ APIUtils.test.js

feed/

├─ index.js

├─ Feed.js

├─ Feed.css

├─ FeedStory.js

├─ FeedStory.test.js

└─ FeedAPI.js

profile/

├─ index.js

├─ Profile.js

├─ ProfileHeader.js

├─ ProfileHeader.css

└─ ProfileAPI.js

* 1. **Grouping by file type:**

Another popular way to structure projects is to group similar files together.

api/

├─ APIUtils.js

├─ APIUtils.test.js

├─ ProfileAPI.js

└─ UserAPI.js

components/

├─ Avatar.js

├─ Avatar.css

├─ Feed.js

├─ Feed.css

├─ FeedStory.js

├─ FeedStory.test.js

├─ Profile.js

├─ ProfileHeader.js

└─ ProfileHeader.css

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the popular packages for animation?**

*React Transition Group* and *React Motion* are popular animation packages in React ecosystem.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the benefit of styles modules?**

It is recommended to avoid hard coding style values in components. Any values that are likely to be used across different UI components should be extracted into their own modules.

For example, these styles could be extracted into a separate component:

export const colors = {

white,

black,

blue

}

export const space = [

0,

8,

16,

32,

64

]

And then imported individually in other components:

import { space, colors } from './styles'

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the popular React-specific linters?**

ESLint is a popular JavaScript linter. There are plugins available that analyse specific code styles. One of the most common for React is an npm package called eslint-plugin-react. By default, it will check a number of best practices, with rules checking things from keys in iterators to a complete set of prop types.

Another popular plugin is eslint-plugin-jsx-a11y, which will help fix common issues with accessibility. As JSX offers slightly different syntax to regular HTML, issues with alt text and tabindex, for example, will not be picked up by regular plugins.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to make AJAX call and in which component lifecycle methods should I make an AJAX call?**

You can use AJAX libraries such as Axios, jQuery AJAX, and the browser built-in fetch. You should fetch data in the componentDidMount() lifecycle method. This is so you can use setState() to update your component when the data is retrieved.

For example, the employees list fetched from API and set local state:

class MyComponent extends React.Component {

constructor(props) {

super(props)

this.state = {

employees: [],

error: null

}

}

componentDidMount() {

fetch('https://api.example.com/items')

.then(res => res.json())

.then(

(result) => {

this.setState({

employees: result.employees

})

},

(error) => {

this.setState({ error })

}

)

}

render() {

const { error, employees } = this.state

if (error) {

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

} else {

return (

<ul>

{employees.map(employee => (

<li key={employee.name}>

{employee.name}-{employee.experience}

</li>

))}

</ul>

)

}

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are render props?**

**Render Props** is a simple technique for sharing code between components using a prop whose value is a function. The below component uses render prop which returns a React element.

<DataProvider render={data => (

<h1>{`Hello ${data.target}`}</h1>

)}/>

Libraries such as React Router and DownShift are using this pattern.

**React Router**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React Router?**

React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How React Router is different from history library?**

React Router is a wrapper around the history library which handles interaction with the browser's window.history with its browser and hash histories. It also provides memory history which is useful for environments that don't have global history, such as mobile app development (React Native) and unit testing with Node.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the <Router> components of React Router v4?**

React Router v4 provides below 3 <Router> components:

* 1. <BrowserRouter>
  2. <HashRouter>
  3. <MemoryRouter>

The above components will create *browser*, *hash*, and *memory* history instances. React Router v4 makes the properties and methods of the history instance associated with your router available through the context in the router object.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of push() and replace() methods of history?**

A history instance has two methods for navigation purpose.

* 1. push()
  2. replace()

If you think of the history as an array of visited locations, push() will add a new location to the array and replace() will replace the current location in the array with the new one.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you programmatically navigate using React Router v4?**

There are three different ways to achieve programmatic routing/navigation within components.

* 1. **Using the withRouter() higher-order function:**

The withRouter() higher-order function will inject the history object as a prop of the component. This object provides push() and replace() methods to avoid the usage of context.

import { withRouter } from 'react-router-dom' // this also works with 'react-router-native'

const Button = withRouter(({ history }) => (

<button

type='button'

onClick={() => { history.push('/new-location') }}

>

{'Click Me!'}

</button>

))

* 1. **Using <Route> component and render props pattern:**

The <Route> component passes the same props as withRouter(), so you will be able to access the history methods through the history prop.

import { Route } from 'react-router-dom'

const Button = () => (

<Route render={({ history }) => (

<button

type='button'

onClick={() => { history.push('/new-location') }}

>

{'Click Me!'}

</button>

)} />

)

* 1. **Using context:**

This option is not recommended and treated as unstable API.

const Button = (props, context) => (

<button

type='button'

onClick={() => {

context.history.push('/new-location')

}}

>

{'Click Me!'}

</button>

)

Button.contextTypes = {

history: React.PropTypes.shape({

push: React.PropTypes.func.isRequired

})

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to get query parameters in React Router v4?**

The ability to parse query strings was taken out of React Router v4 because there have been user requests over the years to support different implementation. So the decision has been given to users to choose the implementation they like. The recommended approach is to use query strings library.

const queryString = require('query-string');

const parsed = queryString.parse(props.location.search);

You can also use URLSearchParams if you want something native:

const params = new URLSearchParams(props.location.search)

const foo = params.get('name')

You should use a *polyfill* for IE11.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why you get "Router may have only one child element" warning?**

You have to wrap your Route's in a <Switch> block because <Switch> is unique in that it renders a route exclusively.

At first you need to add Switch to your imports:

import { Switch, Router, Route } from 'react-router'

Then define the routes within <Switch> block:

<Router>

<Switch>

<Route {/\* ... \*/} />

<Route {/\* ... \*/} />

</Switch>

</Router>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to pass params to history.push method in React Router v4?**

While navigating you can pass props to the history object:

this.props.history.push({

pathname: '/template',

search: '?name=sudheer',

state: { detail: response.data }

})

The search property is used to pass query params in push() method.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to implement *default* or *NotFound* page?**

A <Switch> renders the first child <Route> that matches. A <Route> with no path always matches. So you just need to simply drop path attribute as below

<Switch>

<Route exact path="/" component={Home}/>

<Route path="/user" component={User}/>

<Route component={NotFound} />

</Switch>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to get history on React Router v4?**

Below are the list of steps to get history object on React Router v4,

* 1. Create a module that exports a history object and import this module across the project.

For example, create history.js file:

import { createBrowserHistory } from 'history'

export default createBrowserHistory({

/\* pass a configuration object here if needed \*/

})

* 1. You should use the <Router> component instead of built-in routers. Imported the above history.js inside index.js file:
  2. import { Router } from 'react-router-dom'
  3. import history from './history'
  4. import App from './App'
  5. ReactDOM.render((
  6. <Router history={history}>
  7. <App />
  8. </Router>

), holder)

* 1. You can also use push method of history object similar to built-in history object:
  2. // some-other-file.js
  3. import history from './history'

history.push('/go-here')

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to perform automatic redirect after login?**

The react-router package provides <Redirect> component in React Router. Rendering a <Redirect> will navigate to a new location. Like server-side redirects, the new location will override the current location in the history stack.

import React, { Component } from 'react'

import { Redirect } from 'react-router'

export default class LoginComponent extends Component {

render() {

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

return <Redirect to="/your/redirect/page" />

} else {

return <div>{'Login Please'}</div>

}

}

}

**React Internationalization**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React Intl?**

The *React Intl* library makes internalization in React straightforward, with off-the-shelf components and an API that can handle everything from formatting strings, dates, and numbers, to pluralization. React Intl is part of *FormatJS* which provides bindings to React via its components and API.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the main features of React Intl?**

Below are the main features of React Intl,

* 1. Display numbers with separators.
  2. Display dates and times correctly.
  3. Display dates relative to "now".
  4. Pluralize labels in strings.
  5. Support for 150+ languages.
  6. Runs in the browser and Node.
  7. Built on standards.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the two ways of formatting in React Intl?**

The library provides two ways to format strings, numbers, and dates:

* 1. **Using react components:**
  2. <FormattedMessage
  3. id={'account'}
  4. defaultMessage={'The amount is less than minimum balance.'}

/>

* 1. **Using an API:**
  2. const messages = defineMessages({
  3. accountMessage: {
  4. id: 'account',
  5. defaultMessage: 'The amount is less than minimum balance.',
  6. }
  7. })

formatMessage(messages.accountMessage)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use <FormattedMessage> as placeholder using React Intl?**

The <Formatted... /> components from react-intl return elements, not plain text, so they can't be used for placeholders, alt text, etc. In that case, you should use lower level API formatMessage(). You can inject the intl object into your component using injectIntl() higher-order component and then format the message using formatMessage() available on that object.

import React from 'react'

import { injectIntl, intlShape } from 'react-intl'

const MyComponent = ({ intl }) => {

const placeholder = intl.formatMessage({id: 'messageId'})

return <input placeholder={placeholder} />

}

MyComponent.propTypes = {

intl: intlShape.isRequired

}

export default injectIntl(MyComponent)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to access current locale with React Intl?**

You can get the current locale in any component of your application using injectIntl():

import { injectIntl, intlShape } from 'react-intl'

const MyComponent = ({ intl }) => (

<div>{`The current locale is ${intl.locale}`}</div>

)

MyComponent.propTypes = {

intl: intlShape.isRequired

}

export default injectIntl(MyComponent)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to format date using React Intl?**

The injectIntl() higher-order component will give you access to the formatDate() method via the props in your component. The method is used internally by instances of FormattedDate and it returns the string representation of the formatted date.

import { injectIntl, intlShape } from 'react-intl'

const stringDate = this.props.intl.formatDate(date, {

year: 'numeric',

month: 'numeric',

day: 'numeric'

})

const MyComponent = ({intl}) => (

<div>{`The formatted date is ${stringDate}`}</div>

)

MyComponent.propTypes = {

intl: intlShape.isRequired

}

export default injectIntl(MyComponent)

**React Testing**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Shallow Renderer in React testing?**

*Shallow rendering* is useful for writing unit test cases in React. It lets you render a component *one level deep* and assert facts about what its render method returns, without worrying about the behavior of child components, which are not instantiated or rendered.

For example, if you have the following component:

function MyComponent() {

return (

<div>

<span className={'heading'}>{'Title'}</span>

<span className={'description'}>{'Description'}</span>

</div>

)

}

Then you can assert as follows:

import ShallowRenderer from 'react-test-renderer/shallow'

// in your test

const renderer = new ShallowRenderer()

renderer.render(<MyComponent />)

const result = renderer.getRenderOutput()

expect(result.type).toBe('div')

expect(result.props.children).toEqual([

<span className={'heading'}>{'Title'}</span>,

<span className={'description'}>{'Description'}</span>

])

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is TestRenderer package in React?**

This package provides a renderer that can be used to render components to pure JavaScript objects, without depending on the DOM or a native mobile environment. This package makes it easy to grab a snapshot of the platform view hierarchy (similar to a DOM tree) rendered by a ReactDOM or React Native without using a browser or jsdom.

import TestRenderer from 'react-test-renderer'

const Link = ({page, children}) => <a href={page}>{children}</a>

const testRenderer = TestRenderer.create(

<Link page={'https://www.facebook.com/'}>{'Facebook'}</Link>

)

console.log(testRenderer.toJSON())

// {

// type: 'a',

// props: { href: 'https://www.facebook.com/' },

// children: [ 'Facebook' ]

// }

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of ReactTestUtils package?**

*ReactTestUtils* are provided in the with-addons package and allow you to perform actions against a simulated DOM for the purpose of unit testing.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Jest?**

*Jest* is a JavaScript unit testing framework created by Facebook based on Jasmine and provides automated mock creation and a jsdom environment. It's often used for testing components.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the advantages of Jest over Jasmine?**

There are couple of advantages compared to Jasmine:

* + Automatically finds tests to execute in your source code.
  + Automatically mocks dependencies when running your tests.
  + Allows you to test asynchronous code synchronously.
  + Runs your tests with a fake DOM implementation (via jsdom) so that your tests can be run on the command line.
  + Runs tests in parallel processes so that they finish sooner.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Give a simple example of Jest test case**

Let's write a test for a function that adds two numbers in sum.js file:

const sum = (a, b) => a + b

export default sum

Create a file named sum.test.js which contains actual test:

import sum from './sum'

test('adds 1 + 2 to equal 3', () => {

expect(sum(1, 2)).toBe(3)

})

And then add the following section to your package.json:

{

"scripts": {

"test": "jest"

}

}

Finally, run yarn test or npm test and Jest will print a result:

$ yarn test

PASS ./sum.test.js

✓ adds 1 + 2 to equal 3 (2ms)

**React Redux**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is flux?**

*Flux* is an *application design paradigm* used as a replacement for the more traditional MVC pattern. It is not a framework or a library but a new kind of architecture that complements React and the concept of Unidirectional Data Flow. Facebook uses this pattern internally when working with React.

The workflow between dispatcher, stores and views components with distinct inputs and outputs as follows:

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/flux.png)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Redux?**

*Redux* is a predictable state container for JavaScript apps based on the *Flux design pattern*. Redux can be used together with React, or with any other view library. It is tiny (about 2kB) and has no dependencies.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the core principles of Redux?**

Redux follows three fundamental principles:

* 1. **Single source of truth:** The state of your whole application is stored in an object tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
  2. **State is read-only:** The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state.
  3. **Changes are made with pure functions:** To specify how the state tree is transformed by actions, you write reducers. Reducers are just pure functions that take the previous state and an action as parameters, and return the next state.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the downsides of Redux compared to Flux?**

Instead of saying downsides we can say that there are few compromises of using Redux over Flux. Those are as follows:

* 1. **You will need to learn to avoid mutations:** Flux is un-opinionated about mutating data, but Redux doesn't like mutations and many packages complementary to Redux assume you never mutate the state. You can enforce this with dev-only packages like redux-immutable-state-invariant, Immutable.js, or instructing your team to write non-mutating code.
  2. **You're going to have to carefully pick your packages:** While Flux explicitly doesn't try to solve problems such as undo/redo, persistence, or forms, Redux has extension points such as middleware and store enhancers, and it has spawned a rich ecosystem.
  3. **There is no nice Flow integration yet:** Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between mapStateToProps() and mapDispatchToProps()?**

mapStateToProps() is a utility which helps your component get updated state (which is updated by some other components):

const mapStateToProps = (state) => {

return {

todos: getVisibleTodos(state.todos, state.visibilityFilter)

}

}

mapDispatchToProps() is a utility which will help your component to fire an action event (dispatching action which may cause change of application state):

const mapDispatchToProps = (dispatch) => {

return {

onTodoClick: (id) => {

dispatch(toggleTodo(id))

}

}

}

Recommend always using the “object shorthand” form for the mapDispatchToProps

Redux wrap it in another function that looks like (…args) => dispatch(onTodoClick(…args)), and pass that wrapper function as a prop to your component.

const mapDispatchToProps = ({

onTodoClick

})

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can I dispatch an action in reducer?**

Dispatching an action within a reducer is an **anti-pattern**. Your reducer should be *without side effects*, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to access Redux store outside a component?**

You just need to export the store from the module where it created with createStore(). Also, it shouldn't pollute the global window object.

store = createStore(myReducer)

export default store

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the drawbacks of MVW pattern?**
   1. DOM manipulation is very expensive which causes applications to behave slow and inefficient.
   2. Due to circular dependencies, a complicated model was created around models and views.
   3. Lot of data changes happens for collaborative applications(like Google Docs).
   4. No way to do undo (travel back in time) easily without adding so much extra code.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Are there any similarities between Redux and RxJS?**

These libraries are very different for very different purposes, but there are some vague similarities.

Redux is a tool for managing state throughout the application. It is usually used as an architecture for UIs. Think of it as an alternative to (half of) Angular. RxJS is a reactive programming library. It is usually used as a tool to accomplish asynchronous tasks in JavaScript. Think of it as an alternative to Promises. Redux uses the Reactive paradigm because the Store is reactive. The Store observes actions from a distance, and changes itself. RxJS also uses the Reactive paradigm, but instead of being an architecture, it gives you basic building blocks, Observables, to accomplish this pattern.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to dispatch an action on load?**

You can dispatch an action in componentDidMount() method and in render() method you can verify the data.

class App extends Component {

componentDidMount() {

this.props.fetchData()

}

render() {

return this.props.isLoaded

? <div>{'Loaded'}</div>

: <div>{'Not Loaded'}</div>

}

}

const mapStateToProps = (state) => ({

isLoaded: state.isLoaded

})

const mapDispatchToProps = { fetchData }

export default connect(mapStateToProps, mapDispatchToProps)(App)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use connect() from React Redux?**

You need to follow two steps to use your store in your container:

* 1. **Use mapStateToProps():** It maps the state variables from your store to the props that you specify.
  2. **Connect the above props to your container:** The object returned by the mapStateToProps function is connected to the container. You can import connect() from react-redux.
  3. import React from 'react'
  4. import { connect } from 'react-redux'
  5. class App extends React.Component {
  6. render() {
  7. return <div>{this.props.containerData}</div>
  8. }
  9. }
  10. function mapStateToProps(state) {
  11. return { containerData: state.data }
  12. }

export default connect(mapStateToProps)(App)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to reset state in Redux?**

You need to write a *root reducer* in your application which delegate handling the action to the reducer generated by combineReducers().

For example, let us take rootReducer() to return the initial state after USER\_LOGOUT action. As we know, reducers are supposed to return the initial state when they are called with undefined as the first argument, no matter the action.

const appReducer = combineReducers({

/\* your app's top-level reducers \*/

})

const rootReducer = (state, action) => {

if (action.type === 'USER\_LOGOUT') {

state = undefined

}

return appReducer(state, action)

}

In case of using redux-persist, you may also need to clean your storage. redux-persist keeps a copy of your state in a storage engine. First, you need to import the appropriate storage engine and then, to parse the state before setting it to undefined and clean each storage state key.

const appReducer = combineReducers({

/\* your app's top-level reducers \*/

})

const rootReducer = (state, action) => {

if (action.type === 'USER\_LOGOUT') {

Object.keys(state).forEach(key => {

storage.removeItem(`persist:${key}`)

})

state = undefined

}

return appReducer(state, action)

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Whats the purpose of at symbol in the Redux connect decorator?**

The **@** symbol is in fact a JavaScript expression used to signify decorators. *Decorators* make it possible to annotate and modify classes and properties at design time.

Let's take an example setting up Redux without and with a decorator.

* + **Without decorator:**
  + import React from 'react'
  + import \* as actionCreators from './actionCreators'
  + import { bindActionCreators } from 'redux'
  + import { connect } from 'react-redux'
  + function mapStateToProps(state) {
  + return { todos: state.todos }
  + }
  + function mapDispatchToProps(dispatch) {
  + return { actions: bindActionCreators(actionCreators, dispatch) }
  + }
  + class MyApp extends React.Component {
  + // ...define your main app here
  + }

export default connect(mapStateToProps, mapDispatchToProps)(MyApp)

* + **With decorator:**
  + import React from 'react'
  + import \* as actionCreators from './actionCreators'
  + import { bindActionCreators } from 'redux'
  + import { connect } from 'react-redux'
  + function mapStateToProps(state) {
  + return { todos: state.todos }
  + }
  + function mapDispatchToProps(dispatch) {
  + return { actions: bindActionCreators(actionCreators, dispatch) }
  + }
  + @connect(mapStateToProps, mapDispatchToProps)
  + export default class MyApp extends React.Component {
  + // ...define your main app here

}

The above examples are almost similar except the usage of decorator. The decorator syntax isn't built into any JavaScript runtimes yet, and is still experimental and subject to change. You can use babel for the decorators support.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between React context and React Redux?**

You can use **Context** in your application directly and is going to be great for passing down data to deeply nested components which what it was designed for.

Whereas **Redux** is much more powerful and provides a large number of features that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't expose this fact in the public API.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why are Redux state functions called reducers?**

Reducers always return the accumulation of the state (based on all previous and current actions). Therefore, they act as a reducer of state. Each time a Redux reducer is called, the state and action are passed as parameters. This state is then reduced (or accumulated) based on the action, and then the next state is returned. You could *reduce* a collection of actions and an initial state (of the store) on which to perform these actions to get the resulting final state.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to make AJAX request in Redux?**

**You can use redux-thunk middleware which allows you to define async actions.**

Let's take an example of fetching specific account as an AJAX call using *fetch API*:

export function fetchAccount(id) {

return dispatch => {

dispatch(setLoadingAccountState()) // Show a loading spinner

fetch(`/account/${id}`, (response) => {

dispatch(doneFetchingAccount()) // Hide loading spinner

if (response.status === 200) {

dispatch(setAccount(response.json)) // Use a normal function to set the received state

} else {

dispatch(someError)

}

})

}

}

function setAccount(data) {

return { type: 'SET\_Account', data: data }

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Should I keep all component's state in Redux store?**

Keep your data in the Redux store, and the UI related state internally in the component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the proper way to access Redux store?**

The best way to access your store in a component is to use the connect() function, that creates a new component that wraps around your existing one. This pattern is called *Higher-Order Components*, and is generally the preferred way of extending a component's functionality in React. This allows you to map state and action creators to your component, and have them passed in automatically as your store updates.

Let's take an example of <FilterLink> component using connect:

import { connect } from 'react-redux'

import { setVisibilityFilter } from '../actions'

import Link from '../components/Link'

const mapStateToProps = (state, ownProps) => ({

active: ownProps.filter === state.visibilityFilter

})

const mapDispatchToProps = (dispatch, ownProps) => ({

onClick: () => dispatch(setVisibilityFilter(ownProps.filter))

})

const FilterLink = connect(

mapStateToProps,

mapDispatchToProps

)(Link)

export default FilterLink

Due to it having quite a few performance optimizations and generally being less likely to cause bugs, the Redux developers almost always recommend using connect() over accessing the store directly (using context API).

class MyComponent {

someMethod() {

doSomethingWith(this.context.store)

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between component and container in React Redux?**

**Component** is a class or function component that describes the presentational part of your application.

**Container** is an informal term for a component that is connected to a Redux store. Containers *subscribe* to Redux state updates and *dispatch* actions, and they usually don't render DOM elements; they delegate rendering to presentational child components.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of the constants in Redux?**

Constants allows you to easily find all usages of that specific functionality across the project when you use an IDE. It also prevents you from introducing silly bugs caused by typos – in which case, you will get a ReferenceError immediately.

Normally we will save them in a single file (constants.js or actionTypes.js).

export const ADD\_TODO = 'ADD\_TODO'

export const DELETE\_TODO = 'DELETE\_TODO'

export const EDIT\_TODO = 'EDIT\_TODO'

export const COMPLETE\_TODO = 'COMPLETE\_TODO'

export const COMPLETE\_ALL = 'COMPLETE\_ALL'

export const CLEAR\_COMPLETED = 'CLEAR\_COMPLETED'

In Redux, you use them in two places:

* 1. **During action creation:**

Let's take actions.js:

import { ADD\_TODO } from './actionTypes';

export function addTodo(text) {

return { type: ADD\_TODO, text }

}

* 1. **In reducers:**

Let's create reducer.js:

import { ADD\_TODO } from './actionTypes'

export default (state = [], action) => {

switch (action.type) {

case ADD\_TODO:

return [

...state,

{

text: action.text,

completed: false

}

];

default:

return state

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the different ways to write mapDispatchToProps()?**

There are a few ways of binding *action creators* to dispatch() in mapDispatchToProps().

Below are the possible options:

const mapDispatchToProps = (dispatch) => ({

action: () => dispatch(action())

})

const mapDispatchToProps = (dispatch) => ({

action: bindActionCreators(action, dispatch)

})

const mapDispatchToProps = { action }

The third option is just a shorthand for the first one.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the use of the ownProps parameter in mapStateToProps() and mapDispatchToProps()?**

If the ownProps parameter is specified, React Redux will pass the props that were passed to the component into your *connect* functions. So, if you use a connected component:

import ConnectedComponent from './containers/ConnectedComponent';

<ConnectedComponent user={'john'} />

The ownProps inside your mapStateToProps() and mapDispatchToProps() functions will be an object:

{ user: 'john' }

You can use this object to decide what to return from those functions.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to structure Redux top level directories?**

Most of the applications has several top-level directories as below:

* 1. **Components**: Used for *dumb* components unaware of Redux.
  2. **Containers**: Used for *smart* components connected to Redux.
  3. **Actions**: Used for all action creators, where file names correspond to part of the app.
  4. **Reducers**: Used for all reducers, where files name correspond to state key.
  5. **Store**: Used for store initialization.

This structure works well for small and medium size apps.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is redux-saga?**

redux-saga is a library that aims to make side effects (asynchronous things like data fetching and impure things like accessing the browser cache) in React/Redux applications easier and better.

It is available in NPM:

$ npm install --save redux-saga

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the mental model of redux-saga?**

*Saga* is like a separate thread in your application, that's solely responsible for side effects. redux-saga is a redux *middleware*, which means this thread can be started, paused and cancelled from the main application with normal Redux actions, it has access to the full Redux application state and it can dispatch Redux actions as well.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the differences between call() and put() in redux-saga?**

Both call() and put() are effect creator functions. call() function is used to create effect description, which instructs middleware to call the promise. put() function creates an effect, which instructs middleware to dispatch an action to the store.

Let's take example of how these effects work for fetching particular user data.

function\* fetchUserSaga(action) {

// `call` function accepts rest arguments, which will be passed to `api.fetchUser` function.

// Instructing middleware to call promise, it resolved value will be assigned to `userData` variable

const userData = yield call(api.fetchUser, action.userId)

// Instructing middleware to dispatch corresponding action.

yield put({

type: 'FETCH\_USER\_SUCCESS',

userData

})

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Redux Thunk?**

*Redux Thunk* middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods dispatch() and getState() as parameters.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the differences between redux-saga and redux-thunk?**

**Both *Redux Thunk* and *Redux Saga* take care of dealing with side effects. In most of the scenarios, Thunk uses *Promises* to deal with them, whereas Saga uses *Generators*. Thunk is simple to use and Promises are familiar to many developers, Sagas/Generators are more powerful but you will need to learn them. But both middleware can coexist, so you can start with Thunks and introduce Sagas when/if you need them.**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Redux DevTools?**

*Redux DevTools* is a live-editing time travel environment for Redux with hot reloading, action replay, and customizable UI. If you don't want to bother with installing Redux DevTools and integrating it into your project, consider using Redux DevTools Extension for Chrome and Firefox.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the features of Redux DevTools?**

Some of the main features of Redux DevTools are below,

* 1. Lets you inspect every state and action payload.
  2. Lets you go back in time by *cancelling* actions.
  3. If you change the reducer code, each *staged* action will be re-evaluated.
  4. If the reducers throw, you will see during which action this happened, and what the error was.
  5. With persistState() store enhancer, you can persist debug sessions across page reloads.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are Redux selectors and why to use them?**

*Selectors* are functions that take Redux state as an argument and return some data to pass to the component.

For example, to get user details from the state:

const getUserData = state => state.user.data

These selectors have two main benefits,

* 1. The selector can compute derived data, allowing Redux to store the minimal possible state
  2. The selector is not recomputed unless one of its arguments changes

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Redux Form?**

*Redux Form* works with React and Redux to enable a form in React to use Redux to store all of its state. Redux Form can be used with raw HTML5 inputs, but it also works very well with common UI frameworks like Material UI, React Widgets and React Bootstrap.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the main features of Redux Form?**

Some of the main features of Redux Form are:

* 1. Field values persistence via Redux store.
  2. Validation (sync/async) and submission.
  3. Formatting, parsing and normalization of field values.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to add multiple middlewares to Redux?**

You can use applyMiddleware().

For example, you can add redux-thunk and logger passing them as arguments to applyMiddleware():

import { createStore, applyMiddleware } from 'redux'

const createStoreWithMiddleware = applyMiddleware(ReduxThunk, logger)(createStore)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to set initial state in Redux?**

You need to pass initial state as second argument to createStore:

const rootReducer = combineReducers({

todos: todos,

visibilityFilter: visibilityFilter

})

const initialState = {

todos: [{ id: 123, name: 'example', completed: false }]

}

const store = createStore(

rootReducer,

initialState

)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How Relay is different from Redux?**

Relay is similar to Redux in that they both use a single store. The main difference is that relay only manages state originated from the server, and all access to the state is used via *GraphQL* queries (for reading data) and mutations (for changing data). Relay caches the data for you and optimizes data fetching for you, by fetching only changed data and nothing more.

1. **What is an action in Redux?**

*Actions* are plain JavaScript objects or payloads of information that send data from your application to your store. They are the only source of information for the store. Actions must have a type property that indicates the type of action being performed.

For example, let's take an action which represents adding a new todo item:

{

type: ADD\_TODO,

text: 'Add todo item'

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

**React Native**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between React Native and React?**

**React** is a JavaScript library, supporting both front end web and being run on the server, for building user interfaces and web applications.

**React Native** is a mobile framework that compiles to native app components, allowing you to build native mobile applications (iOS, Android, and Windows) in JavaScript that allows you to use React to build your components, and implements React under the hood.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to test React Native apps?**

React Native can be tested only in mobile simulators like iOS and Android. You can run the app in your mobile using expo app ([https://expo.io](https://expo.io/)) Where it syncs using QR code, your mobile and computer should be in same wireless network.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to do logging in React Native?**

You can use console.log, console.warn, etc. As of React Native v0.29 you can simply run the following to see logs in the console:

$ react-native log-ios

$ react-native log-android

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to debug your React Native?**

Follow the below steps to debug React Native app:

* 1. Run your application in the iOS simulator.
  2. Press Command + D and a webpage should open up at http://localhost:8081/debugger-ui.
  3. Enable *Pause On Caught Exceptions* for a better debugging experience.
  4. Press Command + Option + I to open the Chrome Developer tools, or open it via View -> Developer -> Developer Tools.
  5. You should now be able to debug as you normally would.

**React supported libraries & Integration**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is reselect and how it works?**

*Reselect* is a **selector library** (for Redux) which uses *memoization* concept. It was originally written to compute derived data from Redux-like applications state, but it can't be tied to any architecture or library.

Reselect keeps a copy of the last inputs/outputs of the last call, and recomputes the result only if one of the inputs changes. If the the same inputs are provided twice in a row, Reselect returns the cached output. It's memoization and cache are fully customizable.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Flow?**

*Flow* is a *static type checker* designed to find type errors in JavaScript. Flow types can express much more fine-grained distinctions than traditional type systems. For example, Flow helps you catch errors involving null, unlike most type systems.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between Flow and PropTypes?**

Flow is a *static analysis tool* (static checker) which uses a superset of the language, allowing you to add type annotations to all of your code and catch an entire class of bugs at compile time.

PropTypes is a *basic type checker* (runtime checker) which has been patched onto React. It can't check anything other than the types of the props being passed to a given component. If you want more flexible typechecking for your entire project Flow/TypeScript are appropriate choices.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use Font Awesome icons in React?**

The below steps followed to include Font Awesome in React:

* 1. Install font-awesome:

$ npm install --save font-awesome

* 1. Import font-awesome in your index.js file:

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

* 1. Add Font Awesome classes in className:
  2. render() {
  3. return <div><i className={'fa fa-spinner'} /></div>

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React Dev Tools?**

*React Developer Tools* let you inspect the component hierarchy, including component props and state. It exists both as a browser extension (for Chrome and Firefox), and as a standalone app (works with other environments including Safari, IE, and React Native).

The official extensions available for different browsers or environments.

* 1. **Chrome extension**
  2. **Firefox extension**
  3. **Standalone app** (Safari, React Native, etc)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why is DevTools not loading in Chrome for local files?**

If you opened a local HTML file in your browser (file://...) then you must first open *Chrome Extensions* and check Allow access to file URLs.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use Polymer in React?**

You need to follow below steps to use Polymer in React,

* 1. Create a Polymer element:
  2. <link rel='import' href='../../bower\_components/polymer/polymer.html' />
  3. Polymer({
  4. is: 'calender-element',
  5. ready: function() {
  6. this.textContent = 'I am a calender'
  7. }

})

* 1. Create the Polymer component HTML tag by importing it in a HTML document, e.g. import it in the index.html of your React application:

<link rel='import' href='./src/polymer-components/calender-element.html'>

* 1. Use that element in the JSX file:
  2. import React from 'react'
  3. class MyComponent extends React.Component {
  4. render() {
  5. return (
  6. <calender-element />
  7. )
  8. }
  9. }

export default MyComponent

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the advantages of React over Vue.js?**

React has the following advantages over Vue.js:

* 1. Gives more flexibility in large apps developing.
  2. Easier to test.
  3. Suitable for mobile apps creating.
  4. More information and solutions available.

**Note:** The above list of advantages are purely opinionated and it vary based on the professional experience. But they are helpful as base parameters.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between React and Angular?**

Let's see the difference between React and Angular in a table format.

| **React** | **Angular** |
| --- | --- |
| React is a library and has only the View layer | Angular is a framework and has complete MVC functionality |
| React handles rendering on the server side | AngularJS renders only on the client side but Angular 2 and above renders on the server side |
| React uses JSX that looks like HTML in JS which can be confusing | Angular follows the template approach for HTML, which makes code shorter and easy to understand |
| React Native, which is a React type to build mobile applications are faster and more stable | Ionic, Angular's mobile native app is relatively less stable and slower |
| In React, data flows only in one way and hence debugging is easy | In Angular, data flows both way i.e it has two-way data binding between children and parent and hence debugging is often difficult |

**Note:** The above list of differences are purely opinionated and it vary based on the professional experience. But they are helpful as base parameters.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why React tab is not showing up in DevTools?**

When the page loads, *React DevTools* sets a global named \_\_REACT\_DEVTOOLS\_GLOBAL\_HOOK\_\_, then React communicates with that hook during initialization. If the website is not using React or if React fails to communicate with DevTools then it won't show up the tab.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are Styled Components?**

styled-components is a JavaScript library for styling React applications. It removes the mapping between styles and components, and lets you write actual CSS augmented with JavaScript.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Give an example of Styled Components?**

Lets create <Title> and <Wrapper> components with specific styles for each.

import React from 'react'

import styled from 'styled-components'

// Create a <Title> component that renders an <h1> which is centered, red and sized at 1.5em

const Title = styled.h1`

font-size: 1.5em;

text-align: center;

color: palevioletred;

`

// Create a <Wrapper> component that renders a <section> with some padding and a papayawhip background

const Wrapper = styled.section`

padding: 4em;

background: papayawhip;

`

These two variables, Title and Wrapper, are now components that you can render just like any other react component.

<Wrapper>

<Title>{'Lets start first styled component!'}</Title>

</Wrapper>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Relay?**

**Relay is a JavaScript framework for providing a data layer and client-server communication to web applications** using the React view layer.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use TypeScript in create-react-app application?**

Starting from react-scripts@2.1.0 or higher, there is a built-in support for typescript. i.e, create-react-app now supports typescript natively. You can just pass --typescript option as below

npx create-react-app my-app --typescript

# or

yarn create react-app my-app --typescript

But for lower versions of react scripts, just supply --scripts-version option as react-scripts-ts while you create a new project. react-scripts-ts is a set of adjustments to take the standard create-react-app project pipeline and bring TypeScript into the mix.

Now the project layout should look like the following:

my-app/

├─ .gitignore

├─ images.d.ts

├─ node\_modules/

├─ public/

├─ src/

│ └─ ...

├─ package.json

├─ tsconfig.json

├─ tsconfig.prod.json

├─ tsconfig.test.json

└─ tslint.json

**Miscellaneous**

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the main features of Reselect library?**

Let's see the main features of Reselect library,

* 1. Selectors can compute derived data, allowing Redux to store the minimal possible state.
  2. Selectors are efficient. A selector is not recomputed unless one of its arguments changes.
  3. Selectors are composable. They can be used as input to other selectors.

1. **Give an example of Reselect usage?**

Let's take calculations and different amounts of a shipment order with the simplified usage of Reselect:

import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items

const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(

shopItemsSelector,

items => items.reduce((acc, item) => acc + item.value, 0)

)

const taxSelector = createSelector(

subtotalSelector,

taxPercentSelector,

(subtotal, taxPercent) => subtotal \* (taxPercent / 100)

)

export const totalSelector = createSelector(

subtotalSelector,

taxSelector,

(subtotal, tax) => ({ total: subtotal + tax })

)

let exampleState = {

shop: {

taxPercent: 8,

items: [

{ name: 'apple', value: 1.20 },

{ name: 'orange', value: 0.95 },

]

}

}

console.log(subtotalSelector(exampleState)) // 2.15

console.log(taxSelector(exampleState)) // 0.172

console.log(totalSelector(exampleState)) // { total: 2.322 }

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Does the statics object work with ES6 classes in React?**

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

someComponent= React.createClass({

statics: {

someMethod: function() {

// ..

}

}

})

But you can write statics inside ES6+ classes as below,

class Component extends React.Component {

static propTypes = {

// ...

}

static someMethod() {

// ...

}

}

or writing them outside class as below,

class Component extends React.Component {

....

}

Component.propTypes = {...}

Component.someMethod = function(){....}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can Redux only be used with React?**

Redux can be used as a data store for any UI layer. The most common usage is with React and React Native, but there are bindings available for Angular, Angular 2, Vue, Mithril, and more. Redux simply provides a subscription mechanism which can be used by any other code.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Do you need to have a particular build tool to use Redux?**

Redux is originally written in ES6 and transpiled for production into ES5 with Webpack and Babel. You should be able to use it regardless of your JavaScript build process. Redux also offers a UMD build that can be used directly without any build process at all.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How Redux Form initialValues get updated from state?**

You need to add enableReinitialize : true setting.

const InitializeFromStateForm = reduxForm({

form: 'initializeFromState',

enableReinitialize : true

})(UserEdit)

If your initialValues prop gets updated, your form will update too.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How React PropTypes allow different types for one prop?**

You can use oneOfType() method of PropTypes.

For example, the height property can be defined with either string or number type as below:

Component.PropTypes = {

size: PropTypes.oneOfType([

PropTypes.string,

PropTypes.number

])

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can I import an SVG file as react component?**

You can import SVG directly as component instead of loading it as a file. This feature is available with react-scripts@2.0.0 and higher.

import { ReactComponent as Logo } from './logo.svg'

const App = () => (

<div>

{/\* Logo is an actual react component \*/}

<Logo />

</div>

)

**Note**: Don't forget about the curly braces in the import.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why are inline ref callbacks or functions not recommended?**

If the ref callback is defined as an inline function, it will get called twice during updates, first with null and then again with the DOM element. This is because a new instance of the function is created with each render, so React needs to clear the old ref and set up the new one.

class UserForm extends Component {

handleSubmit = () => {

console.log("Input Value is: ", this.input.value)

}

render () {

return (

<form onSubmit={this.handleSubmit}>

<input

type='text'

ref={(input) => this.input = input} /> // Access DOM input in handle submit

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

</form>

)

}

}

But our expectation is for the ref callback to get called once, when the component mounts. One quick fix is to use the ES7 class property syntax to define the function

class UserForm extends Component {

handleSubmit = () => {

console.log("Input Value is: ", this.input.value)

}

setSearchInput = (input) => {

this.input = input

}

render () {

return (

<form onSubmit={this.handleSubmit}>

<input

type='text'

ref={this.setSearchInput} /> // Access DOM input in handle submit

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

</form>

)

}

}

\*\*Note:\*\* In React v16.3,

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is render hijacking in react?**

The concept of render hijacking is the ability to control what a component will output from another component. It actually means that you decorate your component by wrapping it into a Higher-Order component. By wrapping you can inject additional props or make other changes, which can cause changing logic of rendering. It does not actually enables hijacking, but by using HOC you make your component behave in different way.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are HOC factory implementations?**

There are two main ways of implementing HOCs in React.

* 1. Props Proxy (PP) and
  2. Inheritance Inversion (II).

But they follow different approaches for manipulating the *WrappedComponent*.

**Props Proxy**

In this approach, the render method of the HOC returns a React Element of the type of the WrappedComponent. We also pass through the props that the HOC receives, hence the name **Props Proxy**.

function ppHOC(WrappedComponent) {

return class PP extends React.Component {

render() {

return <WrappedComponent {...this.props}/>

}

}

}

**Inheritance Inversion**

In this approach, the returned HOC class (Enhancer) extends the WrappedComponent. It is called Inheritance Inversion because instead of the WrappedComponent extending some Enhancer class, it is passively extended by the Enhancer. In this way the relationship between them seems **inverse**.

function iiHOC(WrappedComponent) {

return class Enhancer extends WrappedComponent {

render() {

return super.render()

}

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to pass numbers to React component?**

You should be passing the numbers via curly braces({}) where as strings in quotes

React.render(<User age={30} department={"IT"} />, document.getElementById('container'));

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Do I need to keep all my state into Redux? Should I ever use react internal state?**

It is up to developer decision. i.e, It is developer job to determine what kinds of state make up your application, and where each piece of state should live. Some users prefer to keep every single piece of data in Redux, to maintain a fully serializable and controlled version of their application at all times. Others prefer to keep non-critical or UI state, such as “is this dropdown currently open”, inside a component's internal state.

Below are the thumb rules to determine what kind of data should be put into Redux

* 1. Do other parts of the application care about this data?
  2. Do you need to be able to create further derived data based on this original data?
  3. Is the same data being used to drive multiple components?
  4. Is there value to you in being able to restore this state to a given point in time (ie, time travel debugging)?
  5. Do you want to cache the data (i.e, use what's in state if it's already there instead of re-requesting it)?

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of registerServiceWorker in React?**

React creates a service worker for you without any configuration by default. The service worker is a web API that helps you cache your assets and other files so that when the user is offline or on slow network, he/she can still see results on the screen, as such, it helps you build a better user experience, that's what you should know about service worker's for now. It's all about adding offline capabilities to your site.

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import registerServiceWorker from './registerServiceWorker';

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

registerServiceWorker();

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React memo function?**

Class components can be restricted from rendering when their input props are the same using **PureComponent or shouldComponentUpdate**. Now you can do the same with function components by wrapping them in **React.memo**.

const MyComponent = React.memo(function MyComponent(props) {

/\* only rerenders if props change \*/

});

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is React lazy function?**

The React.lazy function lets you render an dynamic import as a regular component. It will automatically load the bundle containing the OtherComponent when the component gets rendered. This must return a Promise which resolves to a module with a default export containing a React component.

const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {

return (

<div>

<OtherComponent />

</div>

);

}

**Note:** React.lazy and Suspense is not yet available for server-side rendering. If you want to do code-splitting in a server rendered app, we still recommend React Loadable.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to prevent unnecessary updates using setState?**

You can compare current value of the state with an existing state value and decide whether to rerender the page or not. If the values are same then you need to return **null** to stop re-rendering otherwise return the latest state value.

For example, the user profile information is conditionally rendered as follows,

getUserProfile = user => {

const latestAddress = user.address;

this.setState(state => {

if (state.address === latestAddress) {

return null;

} else {

return { title: latestAddress };

}

});

};

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you render Array, Strings and Numbers in React 16 Version?**

**Arrays**: Unlike older releases, you don't need to make sure **render** method return a single element in React16. You are able to return multiple sibling elements without a wrapping element by returning an array.

For example, let us take the below list of developers,

const ReactJSDevs = () => {

return [

<li key="1">John</li>,

<li key="2">Jackie</li>,

<li key="3">Jordan</li>

];

}

You can also merge this array of items in another array component.

const JSDevs = () => {

return (

<ul>

<li>Brad</li>

<li>Brodge</li>

<ReactJSDevs/>

<li>Brandon</li>

</ul>

);

}

**Strings and Numbers:** You can also return string and number type from the render method.

render() {

return 'Welcome to ReactJS questions';

}

// Number

render() {

return 2018;

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to use class field declarations syntax in React classes?**

React Class Components can be made much more concise using the class field declarations. You can initialize local state without using the constructor and declare class methods by using arrow functions without the extra need to bind them.

Let's take a counter example to demonstrate class field declarations for state without using constructor and methods without binding,

class Counter extends Component {

state = { value: 0 };

handleIncrement = () => {

this.setState(prevState => ({

value: prevState.value + 1

}));

};

handleDecrement = () => {

this.setState(prevState => ({

value: prevState.value - 1

}));

};

render() {

return (

<div>

{this.state.value}

<button onClick={this.handleIncrement}>+</button>

<button onClick={this.handleDecrement}>-</button>

</div>

)

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are hooks?**

Hooks is a new feature(React 16.8) that lets you use state and other React features without writing a class.

Let's see an example of useState hook example,

import { useState } from 'react';

function Example() {

// Declare a new state variable, which we'll call "count"

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

return (

<div>

<p>You clicked {count} times</p>

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

Click me

</button>

</div>

);

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the rules needs to follow for hooks?**

You need to follow two rules in order to use hooks,

* 1. Call Hooks only at the top level of your react functions. i.e, You shouldn’t call Hooks inside loops, conditions, or nested functions. This will ensure that Hooks are called in the same order each time a component renders and it preserves the state of Hooks between multiple useState and useEffect calls.
  2. Call Hooks from React Functions only. i.e, You shouldn’t call Hooks from regular JavaScript functions.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to ensure hooks followed the rules in your project?**

React team released an ESLint plugin called **eslint-plugin-react-hooks** that enforces these two rules. You can add this plugin to your project using the below command,

npm install eslint-plugin-react-hooks@next

And apply the below config in your ESLint config file,

// Your ESLint configuration

{

"plugins": [

// ...

"react-hooks"

],

"rules": {

// ...

"react-hooks/rules-of-hooks": "error"

}

}

**Note:** This plugin is intended to use in Create React App by default.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the differences between Flux and Redux?**

Below are the major differences between Flux and Redux

| **Flux** | **Redux** |
| --- | --- |
| State is mutable | State is immutable |
| The Store contains both state and change logic | The Store and change logic are separate |
| There are multiple stores exist | There is only one store exist |
| All the stores are disconnected and flat | Single store with hierarchical reducers |
| It has a singleton dispatcher | There is no concept of dispatcher |
| React components subscribe to the store | Container components uses connect function |

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the benefits of React Router V4?**

Below are the main benefits of React Router V4 module,

* 1. In React Router v4(version 4), the API is completely about components. A router can be visualized as a single component(<BrowserRouter>) which wraps specific child router components(<Route>).
  2. You don't need to manually set history. The router module will take care history by wrapping routes with <BrowserRouter> component.
  3. The application size is reduced by adding only the specific router module(Web, core, or native)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can you describe about componentDidCatch lifecycle method signature?**

The **componentDidCatch** lifecycle method is invoked after an error has been thrown by a descendant component. The method receives two parameters,

* 1. error: - The error object which was thrown
  2. info: - An object with a componentStack key contains the information about which component threw the error.

The method structure would be as follows

componentDidCatch(error, info)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **In which scenarios error boundaries do not catch errors?**

Below are the cases in which error boundaries doesn't work,

* 1. Inside Event handlers
  2. Asynchronous code using **setTimeout or requestAnimationFrame** callbacks
  3. During Server side rendering
  4. When errors thrown in the error boundary code itself

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why do you not need error boundaries for event handlers?**

Error boundaries do not catch errors inside event handlers.

React doesn’t need error boundaries to recover from errors in event handlers. Unlike the render method and lifecycle methods, the event handlers don’t happen during rendering. So if they throw, React still knows what to display on the screen.

If you need to catch an error inside an event handler, use the regular JavaScript try / catch statement:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { error: null };

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

try {

// Do something that could throw

} catch (error) {

this.setState({ error });

}

}

render() {

if (this.state.error) {

return <h1>Caught an error.</h1>

}

return <button onClick={this.handleClick}>Click Me</button>

}

}

Note that the above example is demonstrating regular JavaScript behavior and doesn’t use error boundaries.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between try catch block and error boundaries?**

Try catch block works with imperative code whereas error boundaries are meant for declarative code to render on the screen.

For example, the try catch block used for below imperative code

try {

showButton();

} catch (error) {

// ...

}

Whereas error boundaries wrap declarative code as below,

<ErrorBoundary>

<MyComponent />

</ErrorBoundary>

So if an error occurs in a **componentDidUpdate** method caused by a **setState** somewhere deep in the tree, it will still correctly propagate to the closest error boundary.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the behavior of uncaught errors in react 16?**

In React 16, errors that were not caught by any error boundary will result in unmounting of the whole React component tree. The reason behind this decision is that it is worse to leave corrupted UI in place than to completely remove it. For example, it is worse for a payments app to display a wrong amount than to render nothing.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the proper placement for error boundaries?**

The granularity of error boundaries usage is up to the developer based on project needs. You can follow either of these approaches,

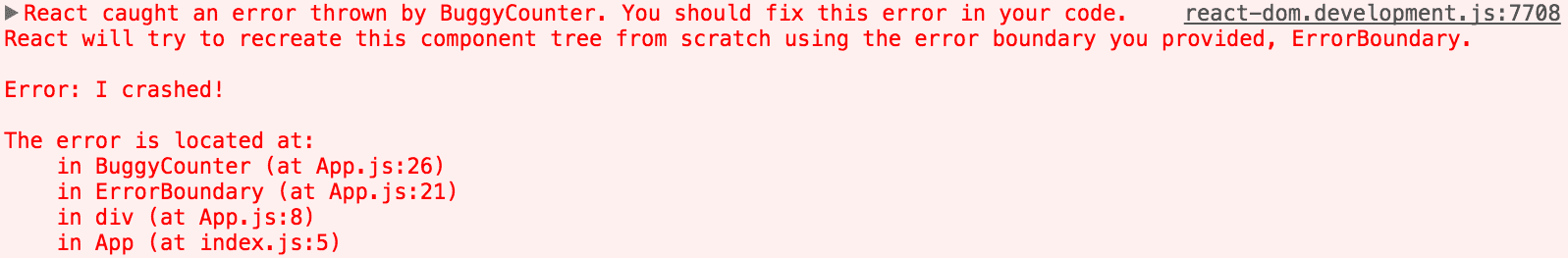
* 1. You can wrap top-level route components to display a generic error message for the entire application.
  2. You can also wrap individual components in an error boundary to protect them from crashing the rest of the application.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the benefit of component stack trace from error boundary?**

Apart from error messages and javascript stack, React16 will display the component stack trace with file names and line numbers using error boundary concept.

For example, BuggyCounter component displays the component stack trace as below,

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/error_boundary.png)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the required method to be defined for a class component?**

The render() method is the only required method in a class component. i.e, All methods other than render method are optional for a class component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the possible return types of render method?**

Below are the list of following types used and return from render method,

* 1. **React elements:** Elements that instruct React to render a DOM node. It includes html elements such as <div/> and user defined elements.
  2. **Arrays and fragments:** Return multiple elements to render as Arrays and Fragments to wrap multiple elements
  3. **Portals:** Render children into a different DOM subtree.
  4. **String and numbers:** Render both Strings and Numbers as text nodes in the DOM
  5. **Booleans or null:** Doesn't render anything but these types are used to conditionally render content.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the main purpose of constructor?**

The constructor is mainly used for two purposes,

* 1. To initialize local state by assigning object to this.state
  2. For binding event handler methods to the instance For example, the below code covers both the above cases,

constructor(props) {

super(props);

// Don't call this.setState() here!

this.state = { counter: 0 };

this.handleClick = this.handleClick.bind(this);

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it mandatory to define constructor for React component?**

No, it is not mandatory. i.e, If you don’t initialize state and you don’t bind methods, you don’t need to implement a constructor for your React component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are default props?**

The defaultProps are defined as a property on the component class to set the default props for the class. This is used for undefined props, but not for null props.

For example, let us create color default prop for the button component,

class MyButton extends React.Component {

// ...

}

MyButton.defaultProps = {

color: 'red'

};

If props.color is not provided then it will set the default value to 'red'. i.e, Whenever you try to access the color prop it uses default value

render() {

return <MyButton /> ; // props.color will be set to red

}

**Note:** If you provide null value then it remains null value.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why should not call setState in componentWillUnmount?**

You should not call setState() in componentWillUnmount() because once a component instance is unmounted, it will never be mounted again.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of getDerivedStateFromError?**

This lifecycle method is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter and should return a value to update state.

The signature of the lifecycle method is as follows,

static getDerivedStateFromError(error)

Let us take error boundary use case with the above lifecycle method for demonstration purpose,

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

// Update state so the next render will show the fallback UI.

return { hasError: true };

}

render() {

if (this.state.hasError) {

// You can render any custom fallback UI

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the methods order when component re-rendered?**

An update can be caused by changes to props or state. The below methods are called in the following order when a component is being re-rendered.

* 1. static getDerivedStateFromProps()
  2. shouldComponentUpdate()
  3. render()
  4. getSnapshotBeforeUpdate()
  5. componentDidUpdate()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the methods invoked during error handling?**

Below methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

* 1. static getDerivedStateFromError()
  2. componentDidCatch()

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of displayName class property?**

The displayName string is used in debugging messages. Usually, you don’t need to set it explicitly because it’s inferred from the name of the function or class that defines the component. You might want to set it explicitly if you want to display a different name for debugging purposes or when you create a higher-order component.

For example, To ease debugging, choose a display name that communicates that it’s the result of a withSubscription HOC.

function withSubscription(WrappedComponent) {

class WithSubscription extends React.Component {/\* ... \*/}

WithSubscription.displayName = `WithSubscription(${getDisplayName(WrappedComponent)})`;

return WithSubscription;

}

function getDisplayName(WrappedComponent) {

return WrappedComponent.displayName || WrappedComponent.name || 'Component';

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the browser support for react applications?**

React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10. If you use **es5-shim and es5-sham** polyfill then it even support old browsers that doesn't support ES5 methods.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of unmountComponentAtNode method?**

This method is available from react-dom package and it removes a mounted React component from the DOM and clean up its event handlers and state. If no component was mounted in the container, calling this function does nothing. Returns true if a component was unmounted and false if there was no component to unmount.

The method signature would be as follows,

ReactDOM.unmountComponentAtNode(container)

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is code-splitting?**

Code-Splitting is a feature supported by bundlers like Webpack and Browserify which can create multiple bundles that can be dynamically loaded at runtime. The react project supports code splitting via dynamic import() feature.

For example, in the below code snippets, it will make moduleA.js and all its unique dependencies as a separate chunk that only loads after the user clicks the 'Load' button. **moduleA.js**

const moduleA = 'Hello';

export { moduleA };

**App.js**

import React, { Component } from 'react';

class App extends Component {

handleClick = () => {

import('./moduleA')

.then(({ moduleA }) => {

// Use moduleA

})

.catch(err => {

// Handle failure

});

};

render() {

return (

<div>

<button onClick={this.handleClick}>Load</button>

</div>

);

}

}

export default App;

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the benefit of strict mode?**

The will be helpful in the below cases

* 1. Identifying components with **unsafe lifecycle methods**.
  2. Warning about **legacy string ref** API usage.
  3. Detecting unexpected **side effects**.
  4. Detecting **legacy context** API.
  5. Warning about deprecated findDOMNode usage

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are Keyed Fragments?**

The Fragments declared with the explicit <React.Fragment> syntax may have keys. The general use case is mapping a collection to an array of fragments as below,

function Glossary(props) {

return (

<dl>

{props.items.map(item => (

// Without the `key`, React will fire a key warning

<React.Fragment key={item.id}>

<dt>{item.term}</dt>

<dd>{item.description}</dd>

</React.Fragment>

))}

</dl>

);

}

**Note:** key is the only attribute that can be passed to Fragment. In the future, there might be a support for additional attributes, such as event handlers.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Does React support all HTML attributes?**

As of React 16, both standard or custom DOM attributes are fully supported. Since React components often take both custom and DOM-related props, React uses the camelCase convention just like the DOM APIs.

Let us take few props with respect to standard HTML attributes,

<div tabIndex="-1" /> // Just like node.tabIndex DOM API

<div className="Button" /> // Just like node.className DOM API

<input readOnly={true} /> // Just like node.readOnly DOM API

These props work similarly to the corresponding HTML attributes, with the exception of the special cases. It also support all SVG attributes.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the limitations with HOCs?**

Higher-order components come with a few caveats apart from its benefits. Below are the few listed in an order,

* 1. **Don’t use HOCs inside the render method:** It is not recommended to apply a HOC to a component within the render method of a component.
  2. render() {
  3. // A new version of EnhancedComponent is created on every render
  4. // EnhancedComponent1 !== EnhancedComponent2
  5. const EnhancedComponent = enhance(MyComponent);
  6. // That causes the entire subtree to unmount/remount each time!
  7. return <EnhancedComponent />;

}

The above code impact performance by remounting a component that causes the state of that component and all of its children to be lost. Instead, apply HOCs outside the component definition so that the resulting component is created only once.

* 1. **Static methods must be copied over:** When you apply a HOC to a component the new component does not have any of the static methods of the original component
  2. // Define a static method
  3. WrappedComponent.staticMethod = function() {/\*...\*/}
  4. // Now apply a HOC
  5. const EnhancedComponent = enhance(WrappedComponent);
  6. // The enhanced component has no static method

typeof EnhancedComponent.staticMethod === 'undefined' // true

You can overcome this by copying the methods onto the container before returning it,

function enhance(WrappedComponent) {

class Enhance extends React.Component {/\*...\*/}

// Must know exactly which method(s) to copy :(

Enhance.staticMethod = WrappedComponent.staticMethod;

return Enhance;

}

* 1. **Refs aren’t passed through:** For HOCs you need to pass through all props to the wrapped component but this does not work for refs. This is because ref is not really a prop similar to key. In this case you need to use the React.forwardRef API

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to debug forwardRefs in DevTools?**

**React.forwardRef** accepts a render function as parameter and DevTools uses this function to determine what to display for the ref forwarding component.

For example, If you don't name the render function or not using displayName property then it will appear as ”ForwardRef” in the DevTools,

const WrappedComponent = React.forwardRef((props, ref) => {

return <LogProps {...props} forwardedRef={ref} />;

});

But If you name the render function then it will appear as **”ForwardRef(myFunction)”**

const WrappedComponent = React.forwardRef(

function myFunction(props, ref) {

return <LogProps {...props} forwardedRef={ref} />;

}

);

As an alternative, You can also set displayName property for forwardRef function,

function logProps(Component) {

class LogProps extends React.Component {

// ...

}

function forwardRef(props, ref) {

return <LogProps {...props} forwardedRef={ref} />;

}

// Give this component a more helpful display name in DevTools.

// e.g. "ForwardRef(logProps(MyComponent))"

const name = Component.displayName || Component.name;

forwardRef.displayName = `logProps(${name})`;

return React.forwardRef(forwardRef);

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **When component props defaults to true?**

If you pass no value for a prop, it defaults to true. This behavior is available so that it matches the behavior of HTML.

For example, below expressions are equivalent,

<MyInput autocomplete />

<MyInput autocomplete={true} />

**Note:** It is not recommended to use this approach because it can be confused with the ES6 object shorthand (example, {name} which is short for {name: name})

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is NextJS and major features of it?**

Next.js is a popular and lightweight framework for static and server‑rendered applications built with React. It also provides styling and routing solutions. Below are the major features provided by NextJS,

* 1. Server-rendered by default
  2. Automatic code splitting for faster page loads
  3. Simple client-side routing (page based)
  4. Webpack-based dev environment which supports (HMR)
  5. Able to implement with Express or any other Node.js HTTP server
  6. Customizable with your own Babel and Webpack configurations

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you pass an event handler to a component?**

You can pass event handlers and other functions as props to child components. It can be used in child component as below,

<button onClick={this.handleClick}>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it good to use arrow functions in render methods?**

Yes, You can use. It is often the easiest way to pass parameters to callback functions. But you need to optimize the performance while using it.

class Foo extends Component {

handleClick() {

console.log('Click happened');

}

render() {

return <button onClick={() => this.handleClick()}>Click Me</button>;

}

}

**Note:** Using an arrow function in render method creates a new function each time the component renders, which may have performance implications

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to prevent a function from being called multiple times?**

If you use an event handler such as **onClick or onScroll** and want to prevent the callback from being fired too quickly, then you can limit the rate at which callback is executed. This can be achieved in the below possible ways,

* 1. **Throttling:** Changes based on a time based frequency. For example, it can be used using \_.throttle lodash function
  2. **Debouncing:** Publish changes after a period of inactivity. For example, it can be used using \_.debounce lodash function
  3. **RequestAnimationFrame throttling:** Changes based on requestAnimationFrame. For example, it can be used using raf-schd lodash function

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How JSX prevents Injection Attacks?**

React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered.

For example, you can embed user input as below,

const name = response.potentiallyMaliciousInput;

const element = <h1>{name}</h1>;

This way you can prevent XSS(Cross-site-scripting) attacks in the application.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you update rendered elements?**

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method.

For example, lets take a ticking clock example, where it updates the time by calling render method multiple times,

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

<h2>It is {new Date().toLocaleTimeString()}.</h2>

</div>

);

ReactDOM.render(element, document.getElementById('root'));

}

setInterval(tick, 1000);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you say that props are read only?**

When you declare a component as a function or a class, it must never modify its own props.

Let us take a below capital function,

function capital(amount, interest) {

return amount + interest;

}

The above function is called “pure” because it does not attempt to change their inputs, and always return the same result for the same inputs. Hence, React has a single rule saying "All React components must act like pure functions with respect to their props."

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you say that state updates are merged?**

When you call setState() in the component, React merges the object you provide into the current state.

For example, let us take a facebook user with posts and comments details as state variables,

constructor(props) {

super(props);

this.state = {

posts: [],

comments: []

};

}

Now you can update them independently with separate setState() calls as below,

componentDidMount() {

fetchPosts().then(response => {

this.setState({

posts: response.posts

});

});

fetchComments().then(response => {

this.setState({

comments: response.comments

});

});

}

As mentioned in the above code snippets, this.setState({comments}) updates only comments variable without modifying or replacing posts variable.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you pass arguments to an event handler?**

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or bind method.

Let us take an example of user details updated in a grid,

<button onClick={(e) => this.updateUser(userId, e)}>Update User details</button>

<button onClick={this.updateUser.bind(this, userId)}>Update User details</button>

In both the approaches, the synthetic argument e is passed as a second argument. You need to pass it explicitly for arrow functions and it forwarded automatically for bind method.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to prevent component from rendering?**

You can prevent component from rendering by returning null based on specific condition. This way it can conditionally render component.

function Greeting(props) {

if (!props.loggedIn) {

return null;

}

return (

<div className="greeting">

welcome, {props.name}

</div>

);

}

class User extends React.Component {

constructor(props) {

super(props);

this.state = {loggedIn: false, name: 'John'};

}

render() {

return (

<div>

//Prevent component render if it is not loggedIn

<Greeting loggedIn={this.state.loggedIn} />

<UserDetails name={this.state.name}>

</div>

);

}

In the above example, the greeting component skips its rendering section by applying condition and returning null value.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the conditions to safely use the index as a key?**

There are three conditions to make sure, it is safe use the index as a key.

* 1. The list and items are static– they are not computed and do not change
  2. The items in the list have no ids
  3. The list is never reordered or filtered.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it keys should be globally unique?**

Keys used within arrays should be unique among their siblings but they don’t need to be globally unique. i.e, You can use the same keys with two different arrays.

For example, the below book component uses two arrays with different arrays,

function Book(props) {

const index = (

<ul>

{props.pages.map((page) =>

<li key={page.id}>

{page.title}

</li>

)}

</ul>

);

const content = props.pages.map((page) =>

<div key={page.id}>

<h3>{page.title}</h3>

<p>{page.content}</p>

<p>{page.pageNumber}</p>

</div>

);

return (

<div>

{index}

<hr />

{content}

</div>

);

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the popular choice for form handling?**

Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission.

In detail, You can categorize them as follows,

* 1. Getting values in and out of form state
  2. Validation and error messages
  3. Handling form submission

It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the advantages of formik over redux form library?**

Below are the main reasons to recommend formik over redux form library,

* 1. The form state is inherently short-term and local, so tracking it in Redux (or any kind of Flux library) is unnecessary.
  2. Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases input latency for large apps.
  3. Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why do you not required to use inheritance?**

In React, it is recommend using composition instead of inheritance to reuse code between components. Both Props and composition give you all the flexibility you need to customize a component’s look and behavior in an explicit and safe way. Whereas, If you want to reuse non-UI functionality between components, it is suggested to extracting it into a separate JavaScript module. Later components import it and use that function, object, or a class, without extending it.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can I use web components in react application?**

Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components.

For example, let us use Vaadin date picker web component as below,

import React, { Component } from 'react';

import './App.css';

import '@vaadin/vaadin-date-picker';

class App extends Component {

render() {

return (

<div className="App">

<vaadin-date-picker label="When were you born?"></vaadin-date-picker>

</div>

);

}

}

export default App;

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is dynamic import?**

The dynamic import() syntax is a ECMAScript proposal not currently part of the language standard. It is expected to be accepted in the near future. You can achieve code-splitting into your app using dynamic import.

Let's take an example of addition,

* 1. **Normal Import**

import { add } from './math';

console.log(add(10, 20));

* 1. **Dynamic Import**

import("./math").then(math => {

console.log(math.add(10, 20));

});

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are loadable components?**

If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic import as a regular component.

Lets take an example,

import loadable from '@loadable/component'

const OtherComponent = loadable(() => import('./OtherComponent'))

function MyComponent() {

return (

<div>

<OtherComponent />

</div>

)

}

Now OtherComponent will be loaded in a separated bundle

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is suspense component?**

If the module containing the dynamic import is not yet loaded by the time parent component renders, you must show some fallback content while you’re waiting for it to load using a loading indicator. This can be done using **Suspense** component.

For example, the below code uses suspense component,

const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {

return (

<div>

<Suspense fallback={<div>Loading...</div>}>

<OtherComponent />

</Suspense>

</div>

);

}

As mentioned in the above code, Suspense is wrapped above the lazy component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is route based code splitting?**

One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed.

Let us take an example of route based website using libraries like React Router with React.lazy,

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

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

const Home = lazy(() => import('./routes/Home'));

const About = lazy(() => import('./routes/About'));

const App = () => (

<Router>

<Suspense fallback={<div>Loading...</div>}>

<Switch>

<Route exact path="/" component={Home}/>

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

</Switch>

</Suspense>

</Router>

);

In the above code, the code splitting will happen at each route level.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Give an example on How to use context?**

**Context** is designed to share data that can be considered **global** for a tree of React components.

For example, in the code below lets manually thread through a “theme” prop in order to style the Button component.

//Lets create a context with a default theme value "luna"

const ThemeContext = React.createContext('luna');

// Create App component where it uses provider to pass theme value in the tree

class App extends React.Component {

render() {

return (

<ThemeContext.Provider value="nova">

<Toolbar />

</ThemeContext.Provider>

);

}

}

// A middle component where you don't need to pass theme prop anymore

function Toolbar(props) {

return (

<div>

<ThemedButton />

</div>

);

}

// Lets read theme value in the button component to use

class ThemedButton extends React.Component {

static contextType = ThemeContext;

render() {

return <Button theme={this.context} />;

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of default value in context?**

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them.

Below code snippet provides default theme value as Luna.

const MyContext = React.createContext(defaultValue);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you use contextType?**

ContextType is used to consume the context object. The contextType property can be used in two ways,

* 1. **contextType as property of class:** The contextType property on a class can be assigned a Context object created by React.createContext(). After that, you can consume the nearest current value of that Context type using this.context in any of the lifecycle methods and render function.

Lets assign contextType property on MyClass as below,

class MyClass extends React.Component {

componentDidMount() {

let value = this.context;

/\* perform a side-effect at mount using the value of MyContext \*/

}

componentDidUpdate() {

let value = this.context;

/\* ... \*/

}

componentWillUnmount() {

let value = this.context;

/\* ... \*/

}

render() {

let value = this.context;

/\* render something based on the value of MyContext \*/

}

}

MyClass.contextType = MyContext;

* 1. **Static field** You can use a static class field to initialize your contextType using public class field syntax.
  2. class MyClass extends React.Component {
  3. static contextType = MyContext;
  4. render() {
  5. let value = this.context;
  6. /\* render something based on the value \*/
  7. }

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is a consumer?**

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.

Lets take a simple example,

<MyContext.Consumer>

{value => /\* render something based on the context value \*/}

</MyContext.Consumer>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you solve performance corner cases while using context?**

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider’s parent re-renders.

For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value.

class App extends React.Component {

render() {

return (

<Provider value={{something: 'something'}}>

<Toolbar />

</Provider>

);

}

}

This can be solved by lifting up the value to parent state,

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

value: {something: 'something'},

};

}

render() {

return (

<Provider value={this.state.value}>

<Toolbar />

</Provider>

);

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of forward ref in HOCs?**

Refs will not get passed through because ref is not a prop. It handled differently by React just like **key**. If you add a ref to a HOC, the ref will refer to the outermost container component, not the wrapped component. In this case, you can use Forward Ref API. For example, we can explicitly forward refs to the inner FancyButton component using the React.forwardRef API.

The below HOC logs all props,

function logProps(Component) {

class LogProps extends React.Component {

componentDidUpdate(prevProps) {

console.log('old props:', prevProps);

console.log('new props:', this.props);

}

render() {

const {forwardedRef, ...rest} = this.props;

// Assign the custom prop "forwardedRef" as a ref

return <Component ref={forwardedRef} {...rest} />;

}

}

return React.forwardRef((props, ref) => {

return <LogProps {...props} forwardedRef={ref} />;

});

}

Let's use this HOC to log all props that get passed to our “fancy button” component,

class FancyButton extends React.Component {

focus() {

// ...

}

// ...

}

export default logProps(FancyButton);

Now lets create a ref and pass it to FancyButton component. In this case, you can set focus to button element.

import FancyButton from './FancyButton';

const ref = React.createRef();

ref.current.focus();

<FancyButton

label="Click Me"

handleClick={handleClick}

ref={ref}

/>;

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it ref argument available for all functions or class components?**

Regular function or class components don’t receive the ref argument, and ref is not available in props either. The second ref argument only exists when you define a component with React.forwardRef call.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why do you need additional care for component libraries while using forward refs?**

When you start using forwardRef in a component library, you should treat it as a breaking change and release a new major version of your library. This is because your library likely has a different behavior such as what refs get assigned to, and what types are exported. These changes can break apps and other libraries that depend on the old behavior.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to create react class components without ES6?**

If you don’t use ES6 then you may need to use the create-react-class module instead. For default props, you need to define getDefaultProps() as a function on the passed object. Whereas for initial state, you have to provide a separate getInitialState method that returns the initial state.

var Greeting = createReactClass({

getDefaultProps: function() {

return {

name: 'Jhohn'

};

},

getInitialState: function() {

return {message: this.props.message};

},

handleClick: function() {

console.log(this.state.message);

},

render: function() {

return <h1>Hello, {this.props.name}</h1>;

}

});

**Note:** If you use createReactClass then auto binding is available for all methods. i.e, You don't need to use .bind(this) with in constructor for event handlers.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it possible to use react without JSX?**

Yes, JSX is not mandatory for using React. Actually it is convenient when you don’t want to set up compilation in your build environment. Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children).

For example, let us take a greeting example with JSX,

class Greeting extends React.Component {

render() {

return <div>Hello {this.props.message}</div>;

}

}

ReactDOM.render(

<Greeting message="World" />,

document.getElementById('root')

);

You can write the same code without JSX as below,

class Greeting extends React.Component {

render() {

return React.createElement('div', null, `Hello ${this.props.message}`);

}

}

ReactDOM.render(

React.createElement(Greeting, {message: 'World'}, null),

document.getElementById('root')

);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is diffing algorithm?**

React needs to use algorithms to find out how to efficiently update the UI to match the most recent tree. The diffing algorithms is generating the minimum number of operations to transform one tree into another. However, the algorithms have a complexity in the order of O(n3) where n is the number of elements in the tree.

In this case, for displaying 1000 elements would require in the order of one billion comparisons. This is far too expensive. Instead, React implements a heuristic O(n) algorithm based on two assumptions:

* 1. Two elements of different types will produce different trees.
  2. The developer can hint at which child elements may be stable across different renders with a key prop.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the rules covered by diffing algorithm?**

When diffing two trees, React first compares the two root elements. The behavior is different depending on the types of the root elements. It covers the below rules during reconciliation algorithm,

* 1. **Elements Of Different Types:** Whenever the root elements have different types, React will tear down the old tree and build the new tree from scratch. For example, elements to , or from to of different types lead a full rebuild.
  2. **DOM Elements Of The Same Type:** When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same underlying DOM node, and only updates the changed attributes. Lets take an example with same DOM elements except className attribute,
  3. <div className="show" title="ReactJS" />

<div className="hide" title="ReactJS" />

* 1. **Component Elements Of The Same Type:** When a component updates, the instance stays the same, so that state is maintained across renders. React updates the props of the underlying component instance to match the new element, and calls componentWillReceiveProps() and componentWillUpdate() on the underlying instance. After that, the render() method is called and the diff algorithm recurses on the previous result and the new result.
  2. **Recursing On Children:** when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there’s a difference. For example, when adding an element at the end of the children, converting between these two trees works well.
  3. <ul>
  4. <li>first</li>
  5. <li>second</li>
  6. </ul>
  7. <ul>
  8. <li>first</li>
  9. <li>second</li>
  10. <li>third</li>

</ul>

* 1. **Handling keys:** React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For example, adding a key can make the tree conversion efficient,

<ul>

<li key="2015">Duke</li>

<li key="2016">Villanova</li>

</ul>

<ul>

<li key="2014">Connecticut</li>

<li key="2015">Duke</li>

<li key="2016">Villanova</li>

</ul>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **When do you need to use refs?**

There are few use cases to go for refs,

* 1. Managing focus, text selection, or media playback.
  2. Triggering imperative animations.
  3. Integrating with third-party DOM libraries.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it prop must be named as render for render props?**

Even though the pattern named render props, you don’t have to use a prop named render to use this pattern. i.e, Any prop that is a function that a component uses to know what to render is technically a “render prop”. Lets take an example with the children prop for render props,

<Mouse children={mouse => (

<p>The mouse position is {mouse.x}, {mouse.y}</p>

)}/>

Actually children prop doesn’t need to be named in the list of “attributes” in JSX element. Instead, you can keep it directly inside element,

<Mouse>

{mouse => (

<p>The mouse position is {mouse.x}, {mouse.y}</p>

)}

</Mouse>

While using this above technique(without any name), explicitly state that children should be a function in your propTypes.

Mouse.propTypes = {

children: PropTypes.func.isRequired

};

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the problems of using render props with pure components?**

If you create a function inside a render method, it negates the purpose of pure component. Because the shallow prop comparison will always return false for new props, and each render in this case will generate a new value for the render prop. You can solve this issue by defining the render function as instance method.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you create HOC using render props?**

You can implement most higher-order components (HOC) using a regular component with a render prop. For example, if you would prefer to have a withMouse HOC instead of a component, you could easily create one using a regular with a render prop.

function withMouse(Component) {

return class extends React.Component {

render() {

return (

<Mouse render={mouse => (

<Component {...this.props} mouse={mouse} />

)}/>

);

}

}

}

This way render props gives the flexibility of using either pattern.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is windowing technique?**

Windowing is a technique that only renders a small subset of your rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created. If your application renders long lists of data then this technique is recommended. Both react-window and react-virtualized are popular windowing libraries which provides several reusable components for displaying lists, grids, and tabular data.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you print falsy values in JSX?**

The falsy values such as false, null, undefined, and true are valid children but they don't render anything. If you still want to display them then you need to convert it to string. Let's take an example on how to convert to a string,

<div>

My JavaScript variable is {String(myVariable)}.

</div>

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the typical use case of portals?**

React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context(z-index,position,opacity etc styles) and you need to visually “break out” of its container.

For example, dialogs, global message notifications, hovercards, and tooltips.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you set default value for uncontrolled component?**

In React, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you might want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a **defaultValue** attribute instead of **value**.

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

User Name:

<input

defaultValue="John"

type="text"

ref={this.input} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

The same applies for select and textArea inputs. But you need to use **defaultChecked** for checkbox and radio inputs.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is your favorite React stack?**

Even though the tech stack varies from developer to developer, the most popular stack is used in react boilerplate project code. It mainly uses Redux and redux-saga for state management and asynchronous side-effects, react-router for routing purpose, styled-components for styling react components, axios for invoking REST api, and other supported stack such as webpack, reselect, ESNext, Babel. You can clone the project <https://github.com/react-boilerplate/react-boilerplate> and start working on any new react project.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between Real DOM and Virtual DOM?**

Below are the main differences between Real DOM and Virtual DOM,

| **Real DOM** | **Virtual DOM** |
| --- | --- |
| Updates are slow | Updates are fast |
| DOM manipulation is very expensive. | DOM manipulation is very easy |
| You can update HTML directly. | You Can’t directly update HTML |
| It causes too much of memory wastage | There is no memory wastage |
| Creates a new DOM if element updates | It updates the JSX if element update |

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to add Bootstrap to a react application?**

Bootstrap can be added to your React app in a three possible ways,

* 1. Using the Bootstrap CDN: This is the easiest way to add bootstrap. Add both bootstrap CSS and JS resources in a head tag.
  2. Bootstrap as Dependency: If you are using a build tool or a module bundler such as Webpack, then this is the preferred option for adding Bootstrap to your React application

npm install bootstrap

* 1. React Bootstrap Package: In this case, you can add Bootstrap to our React app is by using a package that has rebuilt Bootstrap components to work particularly as React components. Below packages are popular in this category,
     1. react-bootstrap
     2. reactstrap

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can you list down top websites or applications using react as front end framework?**

Below are the top 10 websites using React as their front-end framework,

* 1. Facebook
  2. Uber
  3. Instagram
  4. WhatsApp
  5. Khan Academy
  6. Airbnb
  7. Dropbox
  8. Flipboard
  9. Netflix
  10. PayPal

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is it recommended to use CSS In JS technique in React?**

React does not have any opinion about how styles are defined but if you are a beginner then good starting point is to define your styles in a separate \*.css file as usual and refer to them using className. This functionality is not part of React but came from third-party libraries. But If you want to try a different approach(CSS-In-JS) then styled-components library is a good option.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Do I need to rewrite all my class components with hooks?**

No. But you can try Hooks in a few components(or new components) without rewriting any existing code. Because there are no plans to remove classes in ReactJS.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How to fetch data with React Hooks?**

The effect hook called useEffect is used to fetch the data with axios from the API and to set the data in the local state of the component with the state hook’s update function.

Let's take an example in which it fetches list of react articles from the API

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

import axios from 'axios';

function App() {

const [data, setData] = useState({ hits: [] });

useEffect(async () => {

const result = await axios(

'http://hn.algolia.com/api/v1/search?query=react',

);

setData(result.data);

}, []);

return (

<ul>

{data.hits.map(item => (

<li key={item.objectID}>

<a href={item.url}>{item.title}</a>

</li>

))}

</ul>

);

}

export default App;

Remember we provided an empty array as second argument to the effect hook to avoid activating it on component updates but only for the mounting of the component. i.e, It fetches only for component mount.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Is Hooks cover all use cases for classes?**

Hooks doesn't cover all use cases of classes but there is a plan to add them soon. Currently there are no Hook equivalents to the uncommon **getSnapshotBeforeUpdate** and **componentDidCatch** lifecycles yet.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the stable release for hooks support?**

React includes a stable implementation of React Hooks in 16.8 release for below packages

* 1. React DOM
  2. React DOM Server
  3. React Test Renderer
  4. React Shallow Renderer

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Why do we use array destructuring (square brackets notation) in useState?**

When we declare a state variable with useState, it returns a pair — an array with two items. The first item is the current value, and the second is a function that updates the value. Using [0] and [1] to access them is a bit confusing because they have a specific meaning. This is why we use array destructuring instead.

For example, the array index access would look as follows:

var userStateVariable = useState('userProfile'); // Returns an array pair

var user = userStateVariable[0]; // Access first item

var setUser = userStateVariable[1]; // Access second item

Whereas with array destructuring the variables can be accessed as follows:

const [user, setUser] = useState('userProfile');

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the sources used for introducing hooks?**

Hooks got the ideas from several different sources. Below are some of them,

* 1. Previous experiments with functional APIs in the react-future repository
  2. Community experiments with render prop APIs such as Reactions Component
  3. State variables and state cells in DisplayScript.
  4. Subscriptions in Rx.
  5. Reducer components in ReasonReact.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you access imperative API of web components?**

Web Components often expose an imperative API to implement its functions. You will need to use a **ref** to interact with the DOM node directly if you want to access imperative API of a web component. But if you are using third-party Web Components, the best solution is to write a React component that behaves as a **wrapper** for your Web Component.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is formik?**

Formik is a small react form library that helps you with the three major problems,

* 1. Getting values in and out of form state
  2. Validation and error messages
  3. Handling form submission

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are typical middleware choices for handling asynchronous calls in Redux?**

Some of the popular middleware choices for handling asynchronous calls in Redux eco system are Redux Thunk, Redux Promise, Redux Saga.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Do browsers understand JSX code?**

No, browsers can't understand JSX code. You need a transpiler to convert your JSX to regular Javascript that browsers can understand. The most widely used transpiler right now is Babel.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Describe about data flow in react?**

React implements one-way reactive data flow using props which reduce boilerplate and is easier to understand than traditional two-way data binding.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is react scripts?**

The react-scripts package is a set of scripts from the create-react-app starter pack which helps you kick off projects without configuring. The react-scripts start command sets up the development environment and starts a server, as well as hot module reloading.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the features of create react app?**

Below are the list of some of the features provided by create react app.

* 1. React, JSX, ES6, Typescript and Flow syntax support.
  2. Autoprefixed CSS
  3. CSS Reset/Normalize
  4. A live development server
  5. A fast interactive unit test runner with built-in support for coverage reporting
  6. A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps
  7. An offline-first service worker and a web app manifest, meeting all the Progressive Web App criteria.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of renderToNodeStream method?**

The ReactDOMServer#renderToNodeStream method is used to generate HTML on the server and send the markup down on the initial request for faster page loads. It also helps search engines to crawl your pages easily for SEO purposes. **Note:** Remember this method is not available in the browser but only server.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is MobX?**

MobX is a simple, scalable and battle tested state management solution for applying functional reactive programming (TFRP). For reactJs application, you need to install below packages,

npm install mobx --save

npm install mobx-react --save

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the differences between Redux and MobX?**

Below are the main differences between Redux and MobX,

| **Topic** | **Redux** | **MobX** |
| --- | --- | --- |
| Definition | It is a javascript library for managing the application state | It is a library for reactively managing the state of your applications |
| Programming | It is mainly written in ES6 | It is written in JavaScript(ES5) |
| Data Store | There is only one large store exist for data storage | There is more than one store for storage |
| Usage | Mainly used for large and complex applications | Used for simple applications |
| Performance | Need to be improved | Provides better performance |
| How it stores | Uses JS Object to store | Uses observable to store the data |

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Should I learn ES6 before learning ReactJS?**

No, you don’t have to learn es2015/es6 to learn react. But you may find many resources or React ecosystem uses ES6 extensively. Let's see some of the frequently used ES6 features,

* 1. **Destructuring:** To get props and use them in a component
  2. // in es 5
  3. var someData = this.props.someData
  4. var dispatch = this.props.dispatch
  5. // in es6

const { someData, dispatch } = this.props

* 1. Spread operator: Helps in passing props down into a component
  2. // in es 5
  3. <SomeComponent someData={this.props.someData} dispatch={this.props.dispatch} />
  4. // in es6

<SomeComponent {...this.props} />

* 1. Arrow functions: Makes compact syntax
  2. // es 5
  3. var users = usersList.map(function (user) {
  4. return <li>{user.name}</li>
  5. })
  6. // es 6

const users = usersList.map(user => <li>{user.name}</li>);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is Concurrent Rendering?**

The Concurrent rendering makes React apps to be more responsive by rendering component trees without blocking the main UI thread. It allows React to interrupt a long-running render to handle a high-priority event. i.e, When you enabled concurrent Mode, React will keep an eye on other tasks that need to be done, and if there's something with a higher priority it will pause what it is currently rendering and let the other task finish first. You can enable this in two ways,

// 1. Part of an app by wrapping with ConcurrentMode

<React.unstable\_ConcurrentMode>

<Something />

</React.unstable\_ConcurrentMode>

// 2. Whole app using createRoot

ReactDOM.unstable\_createRoot(domNode).render(<App />);

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between async mode and concurrent mode?**

Both refers the same thing. Previously concurrent Mode being referred to as "Async Mode" by React team. The name has been changed to highlight React’s ability to perform work on different priority levels. So it avoids the confusion from other approaches to Async Rendering.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **Can I use javascript urls in react16.9?**

Yes, you can use javascript: URLs but it will log a warning in the console. Because URLs starting with javascript: are dangerous by including unsanitized output in a tag like <a href> and create a security hole.

const companyProfile = {

website: "javascript: alert('Your website is hacked')",

};

// It will log a warning

<a href={companyProfile.website}>More details</a>

Remember that the future versions will throw an error for javascript URLs.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the purpose of eslint plugin for hooks?**

The ESLint plugin enforces rules of Hooks to avoid bugs. It assumes that any function starting with ”use” and a capital letter right after it is a Hook. In particular, the rule enforces that,

* 1. Calls to Hooks are either inside a PascalCase function (assumed to be a component) or another useSomething function (assumed to be a custom Hook).
  2. Hooks are called in the same order on every render.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What is the difference between Imperative and Declarative in React?**

Imagine a simple UI component, such as a "Like" button. When you tap it, it turns blue if it was previously grey, and grey if it was previously blue.

The imperative way of doing this would be:

if( user.likes() ) {

if( hasBlue() ) {

removeBlue();

addGrey();

} else {

removeGrey();

addBlue();

}

}

Basically, you have to check what is currently on the screen and handle all the changes necessary to redraw it with the current state, including undoing the changes from the previous state. You can imagine how complex this could be in a real-world scenario.

In contrast, the declarative approach would be:

if( this.state.liked ) {

return <blueLike />;

} else {

return <greyLike />;

}

Because the declarative approach separates concerns, this part of it only needs to handle how the UI should look in a sepecific state, and is therefore much simpler to understand.

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the benefits of using typescript with reactjs?**

Below are some of the benefits of using typescript with Reactjs,

* 1. It is possible to use latest JavaScript features
  2. Use of interfaces for complex type definitions
  3. IDEs such as VS Code was made for TypeScript
  4. Avoid bugs with the ease of readability and Validation

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **How do you make sure that user remains authenticated on page refresh while using Context API State Management?**

When a user logs in and reload, to persist the state generally we add the load user action in the useEffect hooks in the main App.js. While using Redux, loadUser action can be easily accessed.

**App.js**

import {loadUser} from '../actions/auth';

store.dispatch(loadUser());

* But while using **Context API**, to access context in App.js, wrap the AuthState in index.js so that App.js can access the auth context. Now whenever the page reloads, no matter what route you are on, the user will be authenticated as **loadUser** action will be triggered on each re-render.

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import AuthState from './context/auth/AuthState'

ReactDOM.render(

<React.StrictMode>

<AuthState>

<App />

</AuthState>

</React.StrictMode>,

document.getElementById('root')

);

**App.js**

const authContext = useContext(AuthContext);

const { loadUser } = authContext;

useEffect(() => {

loadUser();

},[])

**loadUser**

const loadUser = async () => {

const token = sessionStorage.getItem('token');

if(!token){

dispatch({

type: ERROR

})

}

setAuthToken(token);

try {

const res = await axios('/api/auth');

dispatch({

type: USER\_LOADED,

payload: res.data.data

})

} catch (err) {

console.error(err);

}

}

[**⬆ Back to Top**](https://github.com/sudheerj/reactjs-interview-questions#table-of-contents)

1. **What are the benefits of new JSX transform?**

There are three major benefits of new JSX transform,

* 1. It is possible to use JSX without importing React packages
  2. The compiled output might improve the bundle size in a small amount
  3. The future improvements provides the flexibility to reduce the number of concepts to learn React.

1. **How does new JSX transform different from old transform?**

The new JSX transform doesn’t require React to be in scope. i.e, You don't need to import React package for simple scenarios.

Let's take an example to look at the main differences between the old and the new transform,

**Old Transform:**

import React from 'react';

function App() {

return <h1>Good morning!!</h1>;

}

Now JSX transform convert the above code into regular JavaScript as below,

import React from 'react';

function App() {

return React.createElement('h1', null, 'Good morning!!');

}

**New Transform:**

The new JSX transform doesn't require any React imports

function App() {

return <h1>Good morning!!</h1>;

}

Under the hood JSX transform compiles to below code

import {jsx as \_jsx} from 'react/jsx-runtime';

function App() {

return \_jsx('h1', { children: 'Good morning!!' });

}

**Note:** You still need to import React to use Hooks.

1. **How do you get redux scaffolding using create-react-app?**

Redux team has provided official redux+js or redux+typescript templates for create-react-app project. The generated project setup includes,

* 1. Redux Toolkit and React-Redux dependencies
  2. Create and configure Redux store
  3. React-Redux <Provider> passing the store to React components
  4. Small "counter" example to demo how to add redux logic and React-Redux hooks API to interact with the store from components

The below commands need to be executed along with template option as below,

* 1. **Javascript template:**

npx create-react-app my-app --template redux

* 1. **Typescript template:**

npx create-react-app my-app --template redux-typescript

1. **What are React Server components?**

React Server Component is a way to write React component that gets rendered in the server-side with the purpose of improving React app performance. These components allow us to load components from the backend.

**Note:** React Server Components is still under development and not recommended for production yet.

1. **What is prop drilling?**

Prop Drilling is the process by which you pass data from one component of the React Component tree to another by going through other components that do not need the data but only help in passing it around.

1. **What are the different ways to prevent state mutation?**

**1.  Differentiate between Real DOM and Virtual DOM.**

|  |  |
| --- | --- |
| Real DOM vs Virtual DOM | |
| **Real DOM** | **Virtual  DOM** |
| 1. It updates slow. | 1. It updates faster. |
| 2. Can directly update HTML. | 2. Can’t directly update HTML. |
| 3. Creates a new DOM if element updates. | 3. Updates the JSX if element updates. |
| 4. DOM manipulation is very expensive. | 4. DOM manipulation is very easy. |
| 5. Too much of memory wastage. | 5. No memory wastage. |

**2. What is React?**

* React is a front-end JavaScript library developed by Facebook in 2011.
* It follows the component based approach which helps in building reusable UI components.
* It is used for developing complex and interactive web and mobile UI.
* Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

**3. What are the features of React?**

Major features of React are listed below:

1. It uses the **virtual DOM** instead of the real DOM.
2. It uses **server-side rendering**.
3. It follows **uni-directional data flow** or data binding.

**4. List some of the major advantages of React.**

Some of the major advantages of React are:

1. It increases the application’s performance
2. It can be conveniently used on the client as well as server side
3. Because of JSX, code’s readability increases
4. React is easy to integrate with other frameworks like Meteor, Angular, etc
5. Using React, writing UI test cases become extremely easy

**5. What are the limitations of React?**

Limitations of React are listed below:

1. React is just a library, not a full-blown framework
2. Its library is very large and takes time to understand
3. It can be little difficult for the novice programmers to understand
4. Coding gets complex as it uses inline templating and JSX

**React Interview Questions and Answers 2021 | Edureka**



**6. What is JSX?**

JSX is a shorthand for JavaScript XML. This is a type of file used by React which utilizes the expressiveness of JavaScript along with HTML like template syntax. This makes the HTML file really easy to understand. This file makes applications robust and boosts its performance. Below is an example of JSX:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | render(){      return(    <div>    <h1> Hello World from Edureka!!</h1>             </div>        );  } |

**7. What do you understand by Virtual DOM? Explain its working.**

A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system.  
This Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed. 

**8. Why can’t browsers read JSX?**

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

[[](https://www.edureka.co/reactjs-redux-certification-training)](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

**[React with Redux Certification Training](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)**

* *[Instructor-led Sessions](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Assessments](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Assignments](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Lifetime Access](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*

[Explore Curriculum](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

**9. How different is React’s ES6 syntax when compared to ES5?**

Syntax has changed from ES5 to ES6 in following aspects:

1. require vs import

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  var React = require('react');    // ES6  import React from 'react'; |

1. export vs exports

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  module.exports = Component;    // ES6  export default Component; |

1. component and function

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // ES5  var MyComponent = React.createClass({      render: function() {          return    <h3>Hello Edureka!</h3>  ;      }  });    // ES6  class MyComponent extends React.Component {      render() {          return    <h3>Hello Edureka!</h3>  ;      }  } |

1. props

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // ES5  var App = React.createClass({      propTypes: { name: React.PropTypes.string },      render: function() {          return    <h3>Hello, {this.props.name}!</h3>  ;      }  });    // ES6  class App extends React.Component {      render() {          return    <h3>Hello, {this.props.name}!</h3>  ;      }  } |

1. state

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | // ES5  var App = React.createClass({      getInitialState: function() {          return { name: 'world' };      },      render: function() {          return    <h3>Hello, {this.state.name}!</h3>  ;      }  });    // ES6  class App extends React.Component {      constructor() {          super();          this.state = { name: 'world' };      }      render() {          return    <h3>Hello, {this.state.name}!</h3>  ;      }  } |

**10. How is React different from Angular?**

|  |  |  |
| --- | --- | --- |
| React vs Angular | | |
| **TOPIC** | **REACT** | **ANGULAR** |
| *1. ARCHITECTURE* | Only the View of MVC | Complete MVC |
| *2. RENDERING* | Server-side rendering | Client-side rendering |
| *3. DOM* | Uses virtual DOM | Uses real DOM |
| *4. DATA BINDING* | One-way data binding | Two-way data binding |
| *5. DEBUGGING* | Compile time debugging | Runtime debugging |
| *6. AUTHOR* | Facebook | Google |

**React Components – React Interview Questions**

**11. “In React, everything is a component.” Explain.**

Components are the building blocks of a React application’s UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

**12. What is the purpose of render() in React.**

Each React component must have a **render()**mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>,<div>** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

**13. How can you embed two or more components into one?**

We can embed components into one in the following way:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | class MyComponent extends React.Component{      render(){          return(    <div>    <h1>Hello</h1>                    <Header/>              </div>            );      }  }  class Header extends React.Component{      render(){          return    <h1>Header Component</h1>       };  }  ReactDOM.render(      <MyComponent/>, document.getElementById('content')  ); |

**14. What is Props?**

Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data.

**15. What is a state in React and how is it used?**

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are mutable unlike the props and create dynamic and interactive components. They are accessed via **this.state().**

**16. Differentiate between states and props.**

|  |  |  |
| --- | --- | --- |
| States vs Props | | |
| **Conditions** | **State** | **Props** |
| 1. Receive initial value from parent component | Yes | Yes |
| 2. Parent component can change value | No | Yes |
| 3. Set default values inside component | Yes | Yes |
| 4. Changes inside component | Yes | No |
| 5. Set initial value for child components | Yes | Yes |
| 6. Changes inside child components | No | Yes |

**17. How can you update the state of a component?**

State of a component can be updated using this.setState().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | class MyComponent extends React.Component {      constructor() {          super();          this.state = {              name: 'Maxx',              id: '101'          }      }      render()          {              setTimeout(()=>{this.setState({name:'Jaeha', id:'222'})},2000)              return (    <div>    <h1>Hello {this.state.name}</h1>    <h2>Your Id is {this.state.id}</h2>                       </div>                );          }      }  ReactDOM.render(      <MyComponent/>, document.getElementById('content')  ); |

**18. What is arrow function in React? How is it used?**

Arrow functions are more of brief syntax for writing the function expression. They are also called *‘fat arrow*‘ (**=>**) the functions. These functions allow to bind the context of the components properly since in ES6 auto binding is not available by default. Arrow functions are mostly useful while working with the higher order functions.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //General way  render() {      return(          <MyInput onChange={this.handleChange.bind(this) } />      );  }  //With Arrow Function  render() {      return(          <MyInput onChange={ (e) => this.handleOnChange(e) } />      );  } |

**19. Differentiate between stateful and stateless components.**

|  |  |
| --- | --- |
| Stateful vs Stateless | |
| **Stateful Component** | **Stateless Component** |
| 1. Stores info about component’s state change in memory | 1. Calculates the internal state of the components |
| 2. Have authority to change state | 2. Do not have the authority to change state |
| 3. Contains the knowledge of past, current and possible future changes in state | 3. Contains no knowledge of past, current and possible future state changes |
| 4. Stateless components notify them about the requirement of the state change, then they send down the props to them. | 4. They receive the props from the Stateful components and treat them as callback functions. |

**20. What are the different phases of React component’s lifecycle?**

There are three different phases of React component’s lifecycle:

1. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
2. *Updating Phase:*Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this phase.
3. *Unmounting Phase:*This is the final phase of a component’s life cycle in which the component is destroyed and removed from the DOM.

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

**React Interview Questions**

**21. Explain the lifecycle methods of React components in detail.**

Some of the most important lifecycle methods are:

1. ***componentWillMount()***–Executed just before rendering takes place both on the client as well as server-side.
2. ***componentDidMount()***–Executed on the client side only after the first render.
3. ***componentWillReceiveProps()***– Invoked as soon as the props are received from the parent class and before another render is called.
4. ***shouldComponentUpdate()***–Returns true or false value based on certain conditions. If you want your component to update, return **true** else return **false**. By default, it returns false.
5. ***componentWillUpdate()***– Called just before rendering takes place in the DOM.
6. ***componentDidUpdate()***–Called immediately after rendering takes place.
7. ***componentWillUnmount()***– Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

**22. What is an event in React?**

In React, events are the triggered reactions to specific actions like mouse hover, mouse click, key press, etc. Handling these events are similar to handling events in DOM elements. But there are some syntactical differences like:

1. Events are named using camel case instead of just using the lowercase.
2. Events are passed as functions instead of strings.

The event argument contains a set of properties, which are specific to an event. Each event type contains its own properties and behavior which can be accessed via its event handler only.

**23. How do you create an event in React?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Display extends React.Component({      show(evt) {          // code      },      render() {          // Render the div with an onClick prop (value is a function)          return (    <div onClick={this.show}>Click Me!</div>            );      }  }); |

**24. What are synthetic events in React?**

Synthetic events are the objects which act as a cross-browser wrapper around the browser’s native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

**25. What do you understand by refs in React?**

Refs is the short hand for References in React. It is an attribute which helps to store a reference to a particular React element or component, which will be returned by the components render configuration function. It is used to return references to a particular element or component returned by render(). They come in handy when we need DOM measurements or to add methods to the components.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class ReferenceDemo extends React.Component{       display() {           const name = this.inputDemo.value;           document.getElementById('disp').innerHTML = name;       }  render() {      return(    <div>              Name: <input type="text" ref={input => this.inputDemo = input} />              <button name="Click" onClick={this.display}>Click</button>    <h2>Hello <span id="disp"></span> !!!</h2>          </div>      );     }   } |

**26. List some of the cases when you should use Refs.**

Following are the cases when refs should be used:

* When you need to manage focus, select text or media playback
* To trigger imperative animations
* Integrate with third-party DOM libraries

**27. How do you modularize code in React?**

We can modularize code by using the export and import properties. They help in writing the components separately in different files.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //ChildComponent.jsx  export default class ChildComponent extends React.Component {      render() {          return(    <div>    <h1>This is a child component</h1>               </div>            );      }  }    //ParentComponent.jsx  import ChildComponent from './childcomponent.js';  class ParentComponent extends React.Component {      render() {          return(    <div>                  <App />              </div>            );      }  } |

**28. How** **are forms created in React?**

React forms are similar to HTML forms. But in React, the state is contained in the state property of the component and is only updated via setState(). Thus the elements can’t directly update their state and their submission is handled by a JavaScript function. This function has full access to the data that is entered by the user into a form.

**Front End Web Development Training**

**[WEB DEVELOPMENT CERTIFICATION TRAINING](https://www.edureka.co/complete-web-developer" \o "Web Development Certification Training " \t "_blank)**

**[Web Development Certification Training](https://www.edureka.co/complete-web-developer" \o "Web Development Certification Training " \t "_blank)**

*[Reviews](https://www.edureka.co/complete-web-developer" \o "Web Development Certification Training " \t "_blank)*

**[5](https://www.edureka.co/complete-web-developer" \o "Web Development Certification Training " \t "_blank)**[(6692)](https://www.edureka.co/complete-web-developer" \o "Web Development Certification Training " \t "_blank)

**[ANGULAR CERTIFICATION TRAINING](https://www.edureka.co/angular-training" \o "Angular Certification Training" \t "_blank)**

**[Angular Certification Training](https://www.edureka.co/angular-training" \o "Angular Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/angular-training" \o "Angular Certification Training" \t "_blank)*

**[5](https://www.edureka.co/angular-training" \o "Angular Certification Training" \t "_blank)**[(16217)](https://www.edureka.co/angular-training" \o "Angular Certification Training" \t "_blank)

**[REACT WITH REDUX CERTIFICATION TRAINING](https://www.edureka.co/reactjs-redux-certification-training" \o "React with Redux Certification Training " \t "_blank)**

**[React with Redux Certification Training](https://www.edureka.co/reactjs-redux-certification-training" \o "React with Redux Certification Training " \t "_blank)**

*[Reviews](https://www.edureka.co/reactjs-redux-certification-training" \o "React with Redux Certification Training " \t "_blank)*

**[5](https://www.edureka.co/reactjs-redux-certification-training" \o "React with Redux Certification Training " \t "_blank)**[(5430)](https://www.edureka.co/reactjs-redux-certification-training" \o "React with Redux Certification Training " \t "_blank)

**[JAVASCRIPT AND JQUERY ESSENTIALS TRAINING AND CERTIFICATION](https://www.edureka.co/javascript-jquery-training" \o "JavaScript and JQuery Essentials Training and Certification" \t "_blank)**

**[JavaScript and JQuery Essentials Training and Certification](https://www.edureka.co/javascript-jquery-training" \o "JavaScript and JQuery Essentials Training and Certification" \t "_blank)**

*[Reviews](https://www.edureka.co/javascript-jquery-training" \o "JavaScript and JQuery Essentials Training and Certification" \t "_blank)*

**[5](https://www.edureka.co/javascript-jquery-training" \o "JavaScript and JQuery Essentials Training and Certification" \t "_blank)**[(2621)](https://www.edureka.co/javascript-jquery-training" \o "JavaScript and JQuery Essentials Training and Certification" \t "_blank)

**[JQUERY UI DEVELOPMENT](https://www.edureka.co/jquery-ui-development" \o "jQuery UI Development" \t "_blank)**

**[jQuery UI Development](https://www.edureka.co/jquery-ui-development" \o "jQuery UI Development" \t "_blank)**

*[Reviews](https://www.edureka.co/jquery-ui-development" \o "jQuery UI Development" \t "_blank)*

**[5](https://www.edureka.co/jquery-ui-development" \o "jQuery UI Development" \t "_blank)**[(1008)](https://www.edureka.co/jquery-ui-development" \o "jQuery UI Development" \t "_blank)

Next

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | handleSubmit(event) {      alert('A name was submitted: ' + this.state.value);      event.preventDefault();  }    render() {      return (    <form onSubmit={this.handleSubmit}>              <label>                  Name:                  <input type="text" value={this.state.value} onChange={this.handleSubmit} />              </label>              <input type="submit" value="Submit" />          </form>        );  } |

**29. What do you know about controlled and uncontrolled components?**

|  |  |
| --- | --- |
| Controlled vs Uncontrolled Components | |
| **Controlled Components** | **Uncontrolled Components** |
| 1. They do not maintain their own state | 1. They maintain their own state |
| 2. Data is controlled by the parent component | 2. Data is controlled by the DOM |
| 3. They take in the current values through props and then notify the changes via callbacks | 3. Refs are used to get their current values |

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

**React Interview Questions**

**30. What are Higher Order Components(HOC)?**

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

**31. What can you do with HOC?**

HOC can be used for many tasks like:

* Code reuse, logic and bootstrap abstraction
* Render High jacking
* State abstraction and manipulation
* Props manipulation

**32. What are Pure Components?**

*Pure*components are the simplest and fastest components which can be written. They can replace any component which only has a **render().**These components enhance the simplicity of the code and performance of the application.

**33. What is the significance of keys in React?**

Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application’s performance.

**React Redux – React Interview Questions**

**34. What were the major problems with MVC framework?**

Following are some of the major problems with MVC framework:

* DOM manipulation was very expensive
* Applications were slow and inefficient
* There was huge memory wastage
* Because of circular dependencies, a complicated model was created around models and views

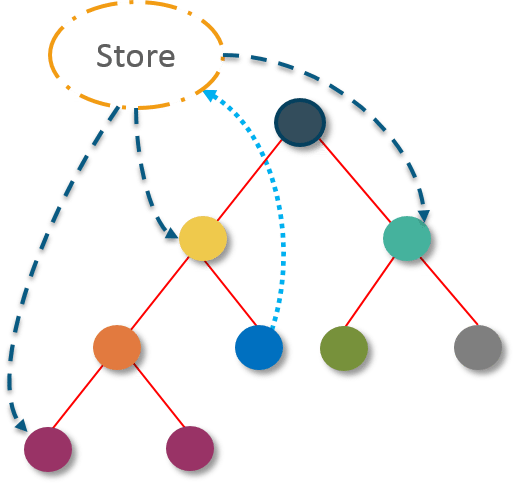
**35. Explain Flux.**

**Flux is an architectural pattern which enforces the uni-directional data flow.** It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

**36.** **What is Redux?**

Redux is one of the most trending libraries for front-end development in today’s marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behavior.

**37. What are the three principles that Redux follows?**

1. ***Single source of truth:***The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
2. ***State is read-only:***The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.
3. ***Changes are made with pure functions:*** In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

**38. What do you understand by “Single source of truth”?**

Redux uses ‘Store’ for storing the application’s entire state at one place. So all the component’s state are stored in the Store and they receive updates from the Store itself. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

**39. List down the components of Redux.**

**Redux is composed of the following components:**

1. **Action – It’s an object that describes what happened.**
2. **Reducer –  It is a place to determine how the state will change.**
3. **Store – State/ Object tree of the entire application is saved in the Store.**
4. **View – Simply displays the data provided by the Store**.

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

**40. Show how the data flows through Redux?**



**41. How are Actions defined in Redux?**

Actions in React must have a type property that indicates the type of ACTION being performed. They must be defined as a String constant and you can add more properties to it as well. In Redux, actions are created using the functions called Action Creators. Below is an example of Action and Action Creator:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function addTodo(text) {         return {                  type: ADD\_TODO,                   text      }  } |

**42. Explain the role of Reducer.**

Reducers are pure functions which specify how the application’s state changes in response to an ACTION. Reducers work by taking in the previous state and action, and then it returns a new state. It determines what sort of update needs to be done based on the type of the action, and then returns new values. It returns the previous state as it is, if no work needs to be done.

**43. What is the significance of Store in Redux?**

A store is a JavaScript object which can hold the application’s state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable. We can pass middleware to the store to handle the processing of data as well as to keep a log of various actions that change the state of stores. All the actions return a new state via reducers.

**44. How is Redux different from Flux?**

|  |  |
| --- | --- |
| Flux vs Redux | |
| **Flux** | **Redux** |
| 1. The Store contains state and change logic | 1. Store and change logic are separate |
| 2. There are multiple stores | 2. There is only one store |
| 3. All the stores are disconnected and flat | 3. Single store with hierarchical reducers |
| 4. Has singleton dispatcher | 4. No concept of dispatcher |
| 5. React components subscribe to the store | 5. Container components utilize connect |
| 6. State is mutable | 6. State is immutable |

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

**React Interview Questions**

**45. What are the advantages of Redux?**

Advantages of Redux are listed below:

* **Predictability of outcome –**Since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
* **Maintainability –**The code becomes easier to maintain with a predictable outcome and strict structure.
* **Server-side rendering –** You just need to pass the store created on the server, to the client side. This is very useful for initial render and provides a better user experience as it optimizes the application performance.
* **Developer tools –**From actions to state changes, developers can track everything going on in the application in real time.
* **Community and ecosystem –**Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contribute to the betterment of the library and develop various applications with it.
* **Ease of testing –**Redux’s code is mostly functions which are small, pure and isolated. This makes the code testable and independent.
* **Organization –**Redux is precise about how code should be organized, this makes the code more consistent and easier when a team works with it.

**React Router – React Interview Questions**

**46. What is React Router?**

React Router is a powerful routing library built on top of React, which helps in adding new screens and flows to the application. This keeps the URL in sync with data that’s being displayed on the web page. It maintains a standardized structure and behavior and is used for developing single page web applications. React Router has a simple API.

**47. Why** **is switch keyword used in React Router v4?**

Although a **<div>** is used to encapsulate multiple routes inside the Router. The ‘switch’ keyword is used when you want to display only a single route to be rendered amongst the several defined routes. The **<switch>**tag when in use matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes.

**48. Why do we need a Router in React?**

A Router is used to define multiple routes and when a user types a specific URL, if this URL matches the path of any ‘route’ defined inside the router, then the user is redirected to that particular route. So basically, we need to add a Router library to our app that allows creating multiple routes with each leading to us a unique view.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <switch>      <route exact path=’/’ component={Home}/>      <route path=’/posts/:id’ component={Newpost}/>      <route path=’/posts’   component={Post}/>  </switch> |

**49. List down the advantages of React Router.**

Few advantages are:

[[](https://www.edureka.co/reactjs-redux-certification-training)](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

**[React with Redux Certification Training](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

1. Just like how React is based on components, in React Router v4, the API is *‘All About Components’*. A Router can be visualized as a single root component (**<BrowserRouter>**) in which we enclose the specific child routes (**<route>**).
2. No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the **<BrowserRouter>** component.
3. The packages are split: Three packages one each for Web, Native and Core. This supports the compact size of our application. It is easy to switch over based on a similar coding style.
4. **How is React Router different from conventional routing?**

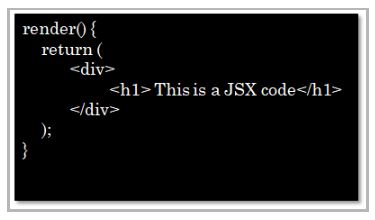
|  |
| --- |
| Conventional Routing vs React Routing |
|  |

### **1. What are the features of React?**

|  |  |
| --- | --- |
| JSX react | **JSX:** JSX is a syntax extension to JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code. |
| Components | **Components:**Components are the building blocks of any React application, and a single app usually consists of multiple components. It splits the user interface into independent, reusable parts that can be processed separately. |
| Virtual DOM | **Virtual DOM:**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, virtual DOM changes only that object in the real DOM, rather than updating all the objects. |
| Data binding | **One-way data-binding:**React’s one-way data binding keeps everything modular and fast. A unidirectional data flow means that when designing a React app, you often nest child components within parent components. |
| High-per | **High performance:**React updates only those components that have changed, rather than updating all the components at once. This results in much faster web applications. |

### **2. What is JSX?**

JSX is a syntax extension of JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code.



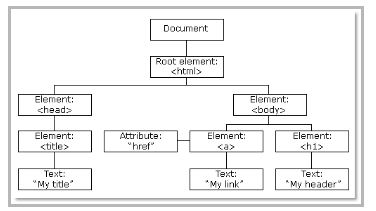
### **3. Can web browsers read JSX directly?**

* 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

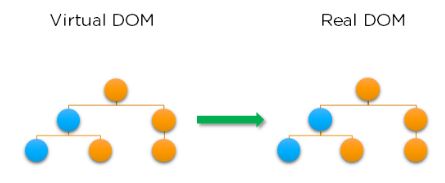


### **4. What is the virtual DOM?**

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.

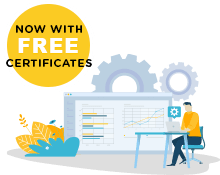


### **5. Why use React instead of other frameworks, like Angular?**

|  |  |
| --- | --- |
| Dynamic | **Easy creation of dynamic applications:** React makes it easier to create dynamic web applications because it provides less coding and provides more functionality, whereas, with JavaScript applications, code tends to get complex very quickly. |
| Improved | **Improved performance:**React uses virtual DOM, which makes web applications perform faster. Virtual DOM compares its previous state and updates only those components in the real DOM, whose states have changed, rather than updating all the components — like conventional web applications. |
| Reusable | **Reusable components:**Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their own logic and controls, and they can be reused through the application, which, in turn, dramatically reduces the development time of an application. |
| Data flow | **Unidirectional data flow:**React follows a unidirectional data flow. This means that when designing a React app, we often nest child components within parent components. And since the data flows in a single direction, it becomes easier to debug errors and know where the problem occurs in an application at the moment. |
| Dedicated | **Dedicated tools for easy debugging:**Facebook has released a chrome extension that we can use to debug React applications. This makes the process of debugging React to web applications faster and easier. |

#### Free ReactJS for Beginners Course

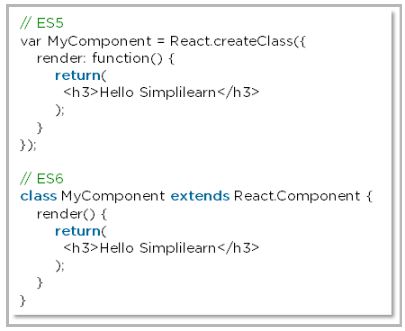
Master the Basics of ReactJS[START LEARNING](https://www.simplilearn.com/learn-react-js-basics-free-course-skillup?utm_source=frs&utm_medium=skillup-course-banner&utm_campaign=frs-skillup-course-promotion)



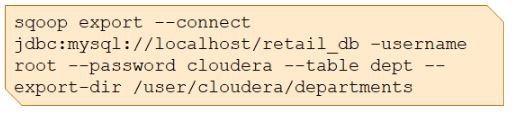
### 6. What is the difference between the ES6 and ES5 standards?

These are the few instances where ES6 syntax has changed from ES5 syntax:

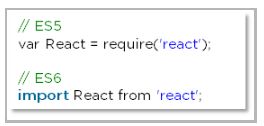
* **Components and Function**



* **exports vs export**



* **require vs import**



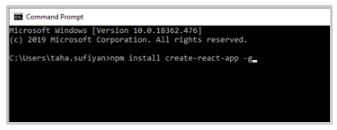
### **7. How do you create a React app?**

These are the steps for creating a React app:

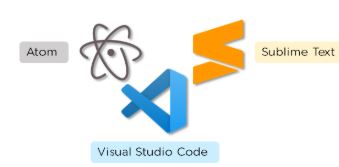
* Install NodeJS on the computer because we need npm to install the React library. Npm is the node package manager that contains many JavaScript libraries, including React.



* Install the **create-react-app** package using the command prompt or terminal.



* Install a text editor of your choice, like VS Code or Sublime Text.



### **8. What is an event in React?**

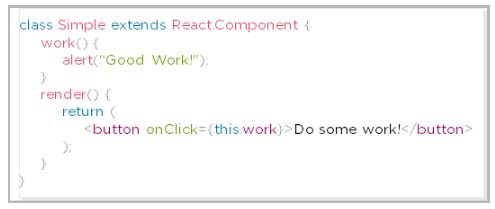
An event is an action that a user or system may trigger, such as pressing a key, a mouse click, etc.

* React events are named using camelCase, rather than lowercase in HTML.
* With JSX, you pass a function as the event handler, rather than a string in HTML.

|  |
| --- |
| <Button onPress={lightItUp} /> |

### **9. How do you create an event in React?**

A React event can be created by doing the following:



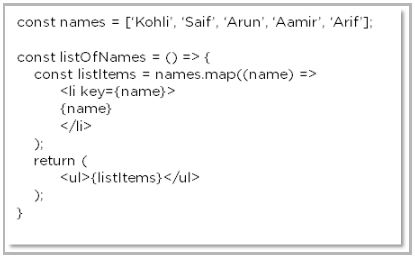
### **10. What are synthetic events in React?**

* Synthetic events combine the response of different browser's native events into one API, ensuring that the events are consistent across different browsers.
* The application is consistent regardless of the browser it is running in. Here, **preventDefault**is a synthetic event.



### **11. Explain how lists work in React**

* We create lists in React as we do in regular JavaScript. Lists display data in an ordered format
* The traversal of lists is done using the map() function



### **12. Why is there a need for using keys in Lists?**

Keys are very important in lists for the following reasons:

* A key is a unique identifier and it is used to identify which items have changed, been updated or deleted from the lists
* It also helps to determine which components need to be re-rendered instead of re-rendering all the components every time. Therefore, it increases performance, as only the updated components are re-rendered

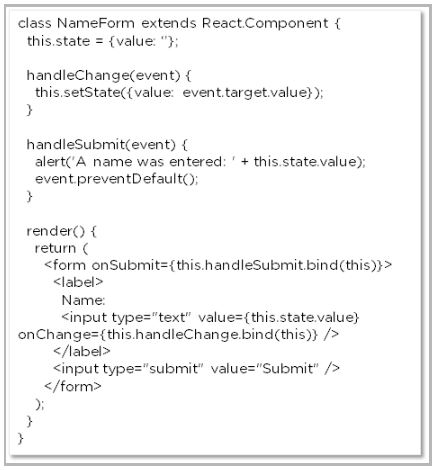
### **13. What are forms in React?**

React employs forms to enable users to interact with web applications.

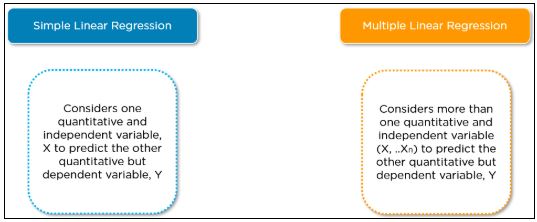
* Using forms, users can interact with the application and enter the required information whenever needed. Form contain certain elements, such as text fields, buttons, checkboxes, radio buttons, etc
* Forms are used for many different tasks such as user authentication, searching, filtering, indexing, etc

### **14. How do you create forms in React?**

We create forms in React by doing the following:



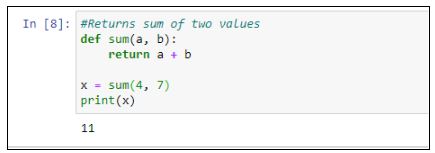
The above code will yield an input field with the label **Name**and a submit button. It will also alert the user when the submit button is pressed.



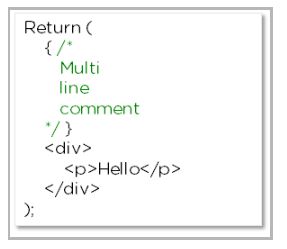
### **15. How do you write comments in React?**

There are basically two ways in which we can write comments:

* Single-line comments

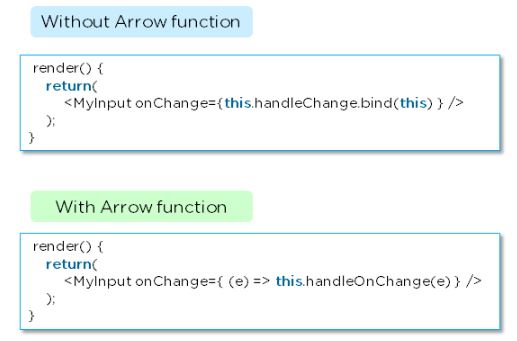


* Multi-line comments



### **16. What is an arrow function and how is it used in React?**

* An arrow function is a short way of writing a function to React.
* It is unnecessary to bind **‘this’**inside the constructor when using an arrow function. This prevents bugs caused by the use of **‘this’**in React callbacks.



### **17. How is React different from React Native?**

|  |  |  |
| --- | --- | --- |
|  | **React** | **React Native** |
| Release | 2013 | 2015 |
| Platform | Web | Mobile – Android, iOS |
| HTML | Yes | No |
| CSS | Yes | No |
| Prerequisites | JavaScript, HTML, CSS | React.js |

### **18. How is React different from Angular?**

|  |  |  |
| --- | --- | --- |
|  | **Angular** | **React** |
| Author | Google | Facebook |
| Architecture | Complete MVC | View layer of MVC |
| DOM | Real DOM | Virtual DOM |
| Data-Binding | Bi-directional | Uni-directional |
| Rendering | Client-Side | Server-Side |
| Performance | Comparatively slow | Faster due to Virtual DOM |

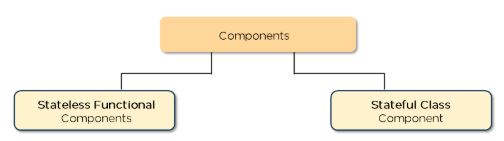
## **ReactJS Component Questions**

Here are some ReactJS Interview Questions on components in React.

### **19. What are the components in React?**

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:



* **Functional Components:**These types of components have no state of their own and only contain render methods, and therefore are also called **stateless components**. They may derive data from other components as props (properties).

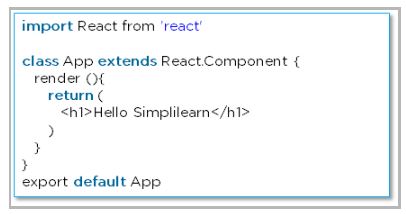
|  |
| --- |
| function Greeting(props) {    return <h1>Welcome to {props.name}</h1>;  } |

* **Class Components:**These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

|  |
| --- |
| class Greeting extends React.Component {    render() {      return <h1>Welcome to {this.props.name}</h1>;    }  } |

### **20. What is the use of render() in React?**

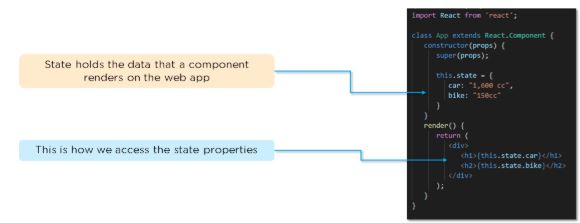
* It is required for each component to have a render() function. This function returns the HTML, which is to be displayed in the component.
* If you need to render more than one element, all of the elements must be inside one parent tag like <div>, <form>.



### **21. What is a state in React?**

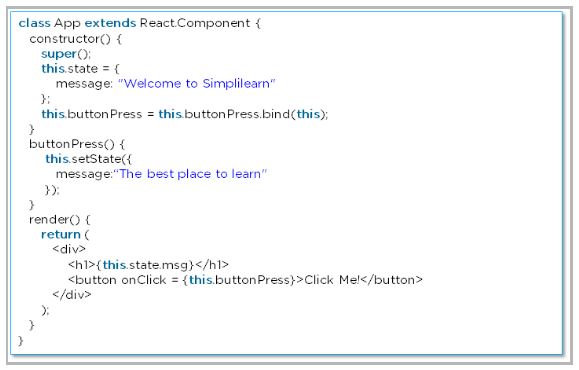
* The state is a built-in React object that is used to contain data or information about the component. The state in a component can change over time, and whenever it changes, the component re-renders.
* The change in state can happen as a response to user action or system-generated events. It determines the behavior of the component and how it will render.

### **22. How do you implement state in React?**



### **23. How do you update the state of a component?**

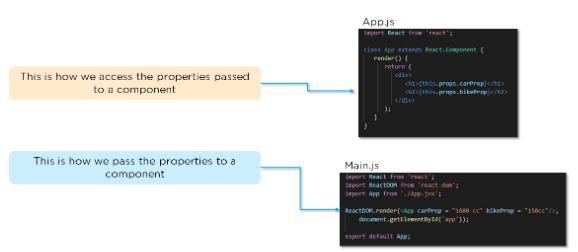
We can update the state of a component by using the built-in **‘setState()’**method:



### **24. What are props in React?**

* [Props](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-props) are short for Properties. It is a React built-in object that stores the value of attributes of a tag and works similarly to HTML attributes.
* Props provide a way to pass data from one component to another component. Props are passed to the component in the same way as arguments are passed in a function.

### **25. How do you pass props between components?**



### **26. What are the differences between state and props?**

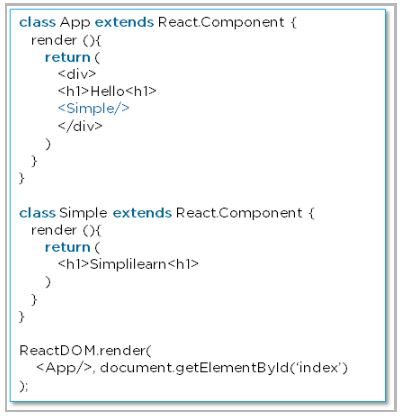
|  |  |  |
| --- | --- | --- |
|  | **State** | **Props** |
| Use | Holds information about the components | Allows to pass data from one component to other components as an argument |
| Mutability | Is mutable | Are immutable |
| Read-Only | Can be changed | Are read-only |
| Child components | Child components cannot access | Child component can access |
| Stateless components | Cannot have state | Can have props |

### **27. What is a higher-order component in React?**

A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic.

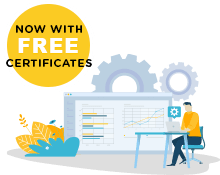
### **28. How can you embed two or more components into one?**

We can embed two or more components into one using this method:



#### FREE Java Certification Training

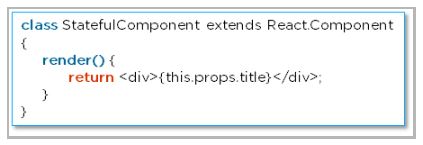
Learn A-Z of Java like never before[ENROLL NOW](https://www.simplilearn.com/learn-java-basics-skillup?utm_source=frs&utm_medium=skillup-course-banner&utm_campaign=frs-tutorial-skillup-course-promotion)



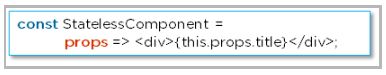
### **29. What are the differences between class and functional components?**

|  |  |  |
| --- | --- | --- |
|  | **Class Components** | **Functional Components** |
| State | Can hold or manage state | Cannot hold or manage state |
| Simplicity | Complex as compared to the stateless component | Simple and easy to understand |
| Lifecycle methods | Can work with all lifecycle methods | Does not work with any lifecycle method |
| Reusability | Can be reused | Cannot be reused |

* **Class components example:**



* **Functional components example:**



### **30. Explain the lifecycle methods of components.**

* **getInitialState():** This is executed before the creation of the component.
* **componentDidMount():** Is executed when the component gets rendered and placed on the DOM.
* **shouldComponentUpdate():** Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.
* **componentDidUpdate():** Is invoked immediately after rendering takes place.
* **componentWillUnmount():** Is invoked immediately before a component is destroyed and unmounted permanently.

## **ReactJS Redux Interview Questions**

Here are some ReactJS Interview Questions on the ReactJS Redux concept.

### **31. What is Redux?**

[Redux](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-with-redux) is an open-source, JavaScript library used to manage the application state. React uses Redux to build the user interface. It is a predictable state container for JavaScript applications and is used for the entire application’s state management.

### **32. What are the components of Redux?**

* **Store:** Holds the state of the application.
* **Action:** The source information for the store.
* **Reducer:** Specifies how the application's state changes in response to actions sent to the store.



### **33. What is the Flux?**

* **Flux is the application architecture that Facebook uses for building web applications.** It is a method of handling complex data inside a client-side application and manages how data flows in a React application.



* There is a single source of data (the store) and triggering certain actions is the only way way to update them.The actions call the dispatcher, and then the store is triggered and updated with their own data accordingly.



* When a dispatch has been triggered, and the store updates, it will emit a change event that the views can rerender accordingly.



### **34. How is Redux different from Flux?**

|  |  |  |
| --- | --- | --- |
| **SN** | **Redux** | **Flux** |
| 1. | Redux is an open-source JavaScript library used to manage application State | Flux is an architecture and not a framework or library |
| 2. | Store’s state is immutable | Store’s state is mutable |
| 3. | Can only have a single-store | Can have multiple stores |
| 4. | Uses the concept of reducer | Uses the concept of the dispatcher |

## **ReactJS Router Questions**

Here are some ReactJS Interview Questions on React Router concepts.

### **35. What is React Router?**

React Router is a routing library built on top of React, which is used to create routes in a React application.

### **36. Why do we need to React Router?**

* It maintains consistent structure and behavior and is used to develop single-page web applications.
* Enables multiple views in a single application by defining multiple routes in the React application.

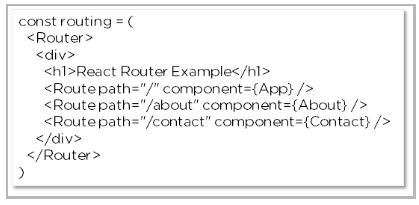
### **37. How is React routing different from conventional routing?**

|  |  |  |
| --- | --- | --- |
| **SN** | **React Routing** | **Conventional routing** |
| 1. | Single HTML page | Each view is a new HTML file |
| 2. | The user navigates multiple views in the same file | The user navigates multiple files for each view |
| 3. | The page does not refresh since it is a single file | The page refreshes every time user navigates |
| 4. | Improved performance | Slower performance |

### **38. How do you implement React routing?**

We can implement routing in our React application using this method:

Considering we have the components **App**, **About**, and **Contact** in our application:



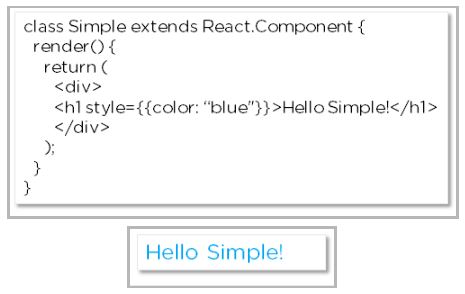
## **ReactJS Styling Questions**

Here are some ReactJS Interview Questions on Styling concept ReactJS.

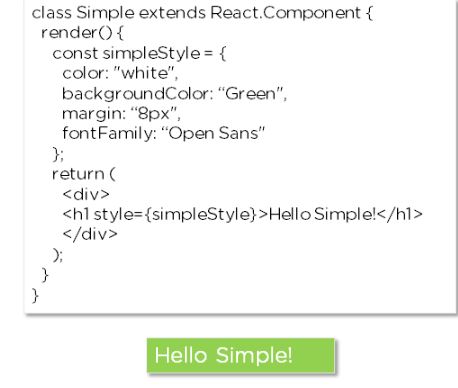
### **39. How do you style React components?**

There are several ways in which we can style React components:

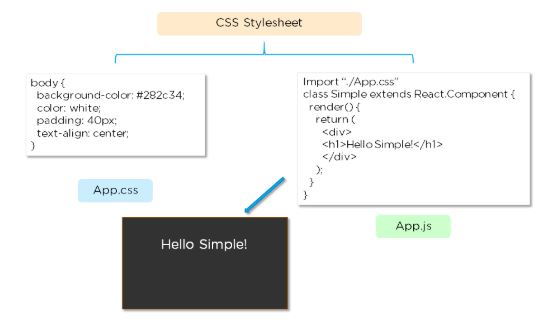
* **Inline Styling**



* **JavaScript Object**



* **CSS Stylesheet**



### **40. Explain the use of CSS modules in React.**

* The CSS module file is created with the **.module.css** extension
* The CSS inside a module file is available only for the component that imported it, so there are no naming conflicts while styling the components.



### 1) What is React?

React is a declarative, efficient, flexible open source front-end JavaScript library developed by Facebook in 2011. It follows the component-based approach for building reusable UI components, especially for single page application. It is used for developing interactive view layer of web and mobile apps. It was created by Jordan Walke, a software engineer at Facebook. It was initially deployed on Facebook's News Feed section in 2011 and later used in its products like WhatsApp & Instagram.

For More Information, [**Click here**](https://www.javatpoint.com/react-introduction).

### 2) What are the features of React?

React framework gaining quick popularity as the best framework among web developers. The main features of React are:

* JSX
* Components
* One-way Data Binding
* Virtual DOM
* Simplicity
* Performance

For More Information, [**Click here**](https://www.javatpoint.com/react-features).

### 3) What are the advantages of React?

The advantages of React are:

* Easy to Learn and USe
* Creating Dynamic Web Applications Becomes Easier
* Reusable Components
* Performance Enhancement
* The Support of Handy Tools
* Known to be SEO Friendly
* The Benefit of Having JavaScript Library
* Scope for Testing the Codes

For More Information, [**Click here**](https://www.javatpoint.com/pros-and-cons-of-react).

### 4) What are the limitations of React?

The limitations of React are:

* The high pace of development
* Poor Documentation
* View Part
* JSX as a barrier

For More Information, [**Click here**](https://www.javatpoint.com/pros-and-cons-of-react).

### 5) What is JSX?

JSX stands for JavaScript XML. It is a React extension which allows writing JavaScript code that looks similar to HTML. It makes HTML file easy to understand. The JSX file makes the React application robust and boosts its performance. JSX provides you to write XML-like syntax in the same file where you write JavaScript code, and then preprocessor (i.e., transpilers like Babel) transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

**Example**

1. **class** App **extends** React.Component {
2. render() {
3. **return**(
4. <div>
5. <h1>Hello JavaTpoint</h1>
6. </div>
7. )
8. }
9. }

In the above example, text inside <h1> tag return as JavaScript function to the render function. After compilation, the JSX expression becomes a normal JavaScript function, as shown below.

1. React.createElement("h1", **null**, "Hello JavaTpoint");

For More Information, [**Click here**](https://www.javatpoint.com/react-jsx).

### 6) Why can't browsers read JSX?

Browsers cannot read JSX directly because they can only understand JavaScript objects, and JSX is not a regular JavaScript object. Thus, we need to transform the JSX file into a JavaScript object using transpilers like Babel and then pass it to the browser.

### 7) Why we use JSX?

* It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
* Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both.
* t is type-safe, and most of the errors can be found at compilation time.
* It makes easier to create templates.

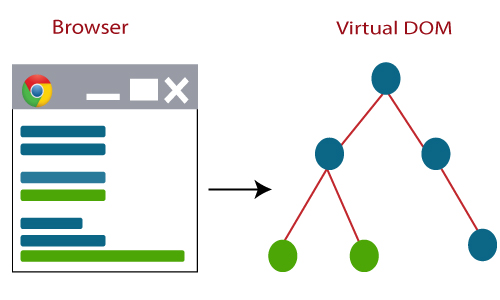
### 8) What do you understand by Virtual DOM?

A Virtual DOM is a lightweight JavaScript object which is an in-memory representation of real DOM. It is an intermediary step between the render function being called and the displaying of elements on the screen. It is similar to a node tree which lists the elements, their attributes, and content as objects and their properties. The render function creates a node tree of the React components and then updates this node tree in response to the mutations in the data model caused by various actions done by the user or by the system.

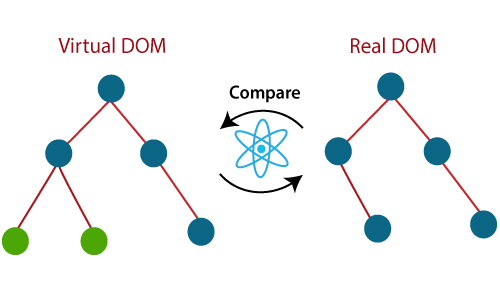
### 9) Explain the working of Virtual DOM.

Virtual DOM works in three steps:

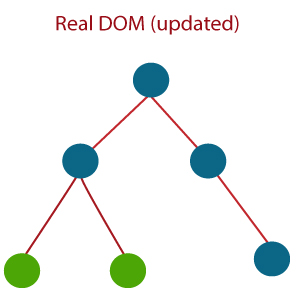
1. Whenever any data changes in the React App, the entire UI is re-rendered in Virtual DOM representation.



2. Now, the difference between the previous DOM representation and the new DOM is calculated.



3. Once the calculations are completed, the real DOM updated with only those things which are changed.



### 10) How is React different from Angular?

The React is different from Angular in the following ways.

|  |  |  |
| --- | --- | --- |
|  | **Angular** | **React** |
| **Author** | Google | Facebook Community |
| **Developer** | Misko Hevery | Jordan Walke |
| **Initial Release** | October 2010 | March 2013 |
| **Language** | JavaScript, HTML | JSX |
| **Type** | Open Source MVC Framework | Open Source JS Framework |
| **Rendering** | Client-Side | Server-Side |
| **Data-Binding** | Bi-directional | Uni-directional |
| **DOM** | Regular DOM | Virtual DOM |
| **Testing** | Unit and Integration Testing | Unit Testing |
| **App Architecture** | MVC | Flux |
| **Performance** | Slow | Fast, due to virtual DOM. |

For More Information, [**Click here**](https://www.javatpoint.com/reactjs-vs-angularjs).

### 11) How React's ES6 syntax is different from ES5 syntax?

The React's ES6 syntax has changed from ES5 syntax in the following aspects.

**require vs. Import**

1. // ES5
2. var React = require('react');
4. // ES6
5. **import** React from 'react';

**exports vs. export**

1. // ES5
2. module.exports = Component;
4. // ES6
5. export **default** Component;

**component and function**

1. // ES5
2. var MyComponent = React.createClass({
3. render: function() {
4. **return**(
5. <h3>Hello JavaTpoint</h3>
6. );
7. }
8. });
10. // ES6
11. **class** MyComponent **extends** React.Component {
12. render() {
13. **return**(
14. <h3>Hello Javatpoint</h3>
15. );
16. }
17. }

**props**

1. // ES5
2. var App = React.createClass({
3. propTypes: { name: React.PropTypes.string },
4. render: function() {
5. **return**(
6. <h3>Hello, {**this**.props.name}!</h3>
7. );
8. }
9. });
11. // ES6
12. **class** App **extends** React.Component {
13. render() {
14. **return**(
15. <h3>Hello, {**this**.props.name}!</h3>
16. );
17. }
18. }

**state**

1. var App = React.createClass({
2. getInitialState: function() {
3. **return** { name: 'world' };
4. },
5. render: function() {
6. **return**(
7. <h3>Hello, {**this**.state.name}!</h3>
8. );
9. }
10. });
12. // ES6
13. **class** App **extends** React.Component {
14. constructor() {
15. **super**();
16. **this**.state = { name: 'world' };
17. }
18. render() {
19. **return**(
20. <h3>Hello, {**this**.state.name}!</h3>
21. );
22. }
23. }

### 12) What is the difference between ReactJS and React Native?

The main differences between ReactJS and React Native are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **ReactJS** | **React Native** |
| **1.** | Initial release in 2013. | Initial release in 2015. |
| **2.** | It is used for developing web applications. | It is used for developing mobile applications. |
| **3.** | It can be executed on all platforms. | It is not platform independent. It takes more effort to be executed on all platforms. |
| **4.** | It uses a JavaScript library and CSS for animations. | It comes with built-in animation libraries. |
| **5.** | It uses React-router for navigating web pages. | It has built-in Navigator library for navigating mobile applications. |
| **6.** | It uses HTML tags. | It does not use HTML tags. |
| **7.** | In this, the Virtual DOM renders the browser code. | In this, Native uses its API to render code for mobile applications. |

For More Information, [**Click here**](https://www.javatpoint.com/reactjs-vs-reactnative).

## React Component Interview Questions

### 13) What do you understand from "In React, everything is a component."

In React, components are the building blocks of React applications. These components divide the entire React application's UI into small, independent, and reusable pieces of code. React renders each of these components independently without affecting the rest of the application UI. Hence, we can say that, in React, everything is a component.

### 14) Explain the purpose of render() in React.

It is mandatory for each React component to have a render() function. Render function is used to return the HTML which you want to display in a component. If you need to rendered more than one HTML element, you need to grouped together inside single enclosing tag (parent tag) such as <div>, <form>, <group> etc. This function returns the same result each time it is invoked.

**Example:** If you need to display a heading, you can do this as below.

1. **import** React from 'react'
3. **class** App **extends** React.Component {
4. render (){
5. **return** (
6. <h1>Hello World</h1>
7. )
8. }
9. }
10. export **default** App

**Points to Note:**

* Each render() function contains a return statement.
* The return statement can have only one parent HTML tag.

### 15) How can you embed two or more components into one?

You can embed two or more components into the following way:

1. **import** React from 'react'
3. **class** App **extends** React.Component {
4. render (){
5. **return** (
6. <h1>Hello World</h1>
7. )
8. }
9. }
11. **class** Example **extends** React.Component {
12. render (){
13. **return** (
14. <h1>Hello JavaTpoint</h1>
15. )
16. }
17. }
18. export **default** App

### 16) What is Props?

Props stand for "Properties" in React. They are read-only inputs to components. Props are an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from the parent to the child components throughout the application.

It is similar to function arguments and passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

For More Information, [**Click here**](https://www.javatpoint.com/react-props).

### 17) What is a State in React?

The State is an updatable structure which holds the data and information about the component. It may be changed over the lifetime of the component in response to user action or system event. It is the heart of the react component which determines the behavior of the component and how it will render. It must be kept as simple as possible.

Let's create a "User" component with "message state."

1. **import** React from 'react'
3. **class** User **extends** React.Component {
4. constructor(props) {
5. **super**(props)
7. **this**.state = {
8. message: 'Welcome to JavaTpoint'
9. }
10. }
12. render() {
13. **return** (
14. <div>
15. <h1>{**this**.state.message}</h1>
16. </div>
17. )
18. }
19. }
20. export **default** User

For More Information, [**Click here**](https://www.javatpoint.com/react-state).

### 18) Differentiate between States and Props.

The major differences between States and Props are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **Props** | **State** |
| **1.** | Props are read-only. | State changes can be asynchronous. |
| **2.** | Props are immutable. | State is mutable. |
| **3.** | Props allow you to pass data from one component to other components as an argument. | State holds information about the components. |
| **4.** | Props can be accessed by the child component. | State cannot be accessed by child components. |
| **5.** | Props are used to communicate between components. | States can be used for rendering dynamic changes with the component. |
| **6.** | The stateless component can have Props. | The stateless components cannot have State. |
| **7.** | Props make components reusable. | The State cannot make components reusable. |
| **8.** | Props are external and controlled by whatever renders the component. | The State is internal and controlled by the component itself. |

For More Information, [**Click here**](https://www.javatpoint.com/react-state-vs-props).

### 19) How can you update the State of a component?

We can update the State of a component using this.setState() method. This method does not always replace the State immediately. Instead, it only adds changes to the original State. It is a primary method which is used to update the user interface(UI) in response to event handlers and server responses.

**Example**

1. **import** React, { Component } from 'react';
2. **import** PropTypes from 'prop-types';
4. **class** App **extends** React.Component {
5. constructor() {
6. **super**();
7. **this**.state = {
8. msg: "Welcome to JavaTpoint"
9. };
10. **this**.updateSetState = **this**.updateSetState.bind(**this**);
11. }
12. updateSetState() {
13. **this**.setState({
14. msg:"Its a best ReactJS tutorial"
15. });
16. }
17. render() {
18. **return** (
19. <div>
20. <h1>{**this**.state.msg}</h1>
21. <button onClick = {**this**.updateSetState}>SET STATE</button>
22. </div>
23. );
24. }
25. }
26. export **default** App;

For More Information, [**Click here**](https://www.javatpoint.com/react-component-api).

### 20) Differentiate between stateless and stateful components.

The difference between stateless and stateful components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Stateless Component** | **Stateful Component** |
| **1.** | The stateless components do not hold or manage state. | The stateful components can hold or manage state. |
| **2.** | It does not contain the knowledge of past, current, and possible future state changes. | It can contain the knowledge of past, current, and possible future changes in state. |
| **3.** | It is also known as a functional component. | It is also known as a class component. |
| **4.** | It is simple and easy to understand. | It is complex as compared to the stateless component. |
| **5.** | It does not work with any lifecycle method of React. | It can work with all lifecycle method of React. |
| **6.** | The stateless components cannot be reused. | The stateful components can be reused. |

### 21) What is arrow function in React? How is it used?

The Arrow function is the new feature of the ES6 standard. If you need to use arrow functions, it is not necessary to bind any event to 'this.' Here, the scope of 'this' is global and not limited to any calling function. So If you are using Arrow Function, there is no need to bind 'this' inside the constructor. It is also called 'fat arrow '(=>) functions.

1. //General way
2. render() {
3. **return**(
4. <MyInput onChange={**this**.handleChange.bind(**this**) } />
5. );
6. }
7. //With Arrow Function
8. render() {
9. **return**(
10. <MyInput onChange={ (e) => **this**.handleOnChange(e) } />
11. );
12. }

### 22) What is an event in React?

An event is an action which triggers as a result of the user action or system generated event like a mouse click, loading of a web page, pressing a key, window resizes, etc. In React, the event handling system is very similar to handling events in DOM elements. The React event handling system is known as Synthetic Event, which is a cross-browser wrapper of the browser's native event.

Handling events with React have some syntactical differences, which are:

* React events are named as camelCase instead of lowercase.
* With JSX, a function is passed as the event handler instead of a string.

For More Information, [**Click here**](https://www.javatpoint.com/react-events).

### 23) How do you create an event in React?

We can create an event as follows.

1. **class** Display **extends** React.Component({
2. show(msgEvent) {
3. // code
4. },
5. render() {
6. // Here, we render the div with an onClick prop
7. **return** (
8. <div onClick={**this**.show}>Click Me</div>
9. );
10. }
11. });

**Example**

1. **import** React, { Component } from 'react';
2. **class** App **extends** React.Component {
3. constructor(props) {
4. **super**(props);
5. **this**.state = {
6. companyName: ''
7. };
8. }
9. changeText(event) {
10. **this**.setState({
11. companyName: event.target.value
12. });
13. }
14. render() {
15. **return** (
16. <div>
17. <h2>Simple Event Example</h2>
18. <label htmlFor="name">Enter company name: </label>
19. <input type="text" id="companyName" onChange={**this**.changeText.bind(**this**)}/>
20. <h4>You entered: { **this**.state.companyName }</h4>
21. </div>
22. );
23. }
24. }
25. export **default** App;

For More Information, [**Click here**](https://www.javatpoint.com/react-events).

### 24) What are synthetic events in React?

**A synthetic event is an object which acts as a cross-browser wrapper around the browser's native event.** It combines the behavior of different browser's native event into one API, including stopPropagation() and preventDefault().

In the given example, e is a Synthetic event.

1. function ActionLink() {
2. function handleClick(e) {
3. e.preventDefault();
4. console.log('You had clicked a Link.');
5. }
6. **return** (
7. <a href="#" onClick={handleClick}>
8. Click\_Me
9. </a>
10. );
11. }

### 25) what is the difference between controlled and uncontrolled components?

The difference between controlled and uncontrolled components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Controlled** | **Uncontrolled** |
| **1.** | It does not maintain its internal state. | It maintains its internal states. |
| **2.** | Here, data is controlled by the parent component. | Here, data is controlled by the DOM itself. |
| **3.** | It accepts its current value as a prop. | It uses a ref for their current values. |
| **4.** | It allows validation control. | It does not allow validation control. |
| **5.** | It has better control over the form elements and data. | It has limited control over the form elements and data. |

For More Information, [**Click here**](https://www.javatpoint.com/react-controlled-vs-uncontrolled-component).

### 26) Explain the Lists in React.

Lists are used to display data in an ordered format. In React, Lists can be created in a similar way as we create it in JavaScript. We can traverse the elements of the list using the map() function.

**Example**

1. **import** React from 'react';
2. **import** ReactDOM from 'react-dom';
4. function NameList(props) {
5. **const** myLists = props.myLists;
6. **const** listItems = myLists.map((myList) =>
7. <li>{myList}</li>
8. );
9. **return** (
10. <div>
11. <h2>Rendering Lists inside component</h2>
12. <ul>{listItems}</ul>
13. </div>
14. );
15. }
16. **const** myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
17. ReactDOM.render(
18. <NameList myLists={myLists} />,
19. document.getElementById('app')
20. );
21. export **default** App;

For More Information, [**Click here**](https://www.javatpoint.com/react-lists).

### 27) What is the significance of keys in React?

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time. It increases application performance.

For More Information, [**Click here**](https://www.javatpoint.com/react-keys).

### 28) How are forms created in React?

Forms allow the users to interact with the application as well as gather information from the users. Forms can perform many tasks such as user authentication, adding user, searching, filtering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

React offers a stateful, reactive approach to build a form. The forms in React are similar to HTML forms. But in React, the state property of the component is only updated via setState(), and a JavaScript function handles their submission. This function has full access to the data which is entered by the user into a form.

1. **import** React, { Component } from 'react';
3. **class** App **extends** React.Component {
4. constructor(props) {
5. **super**(props);
6. **this**.state = {value: ''};
7. **this**.handleChange = **this**.handleChange.bind(**this**);
8. **this**.handleSubmit = **this**.handleSubmit.bind(**this**);
9. }
10. handleChange(event) {
11. **this**.setState({value: event.target.value});
12. }
13. handleSubmit(event) {
14. alert('You have submitted the input successfully: ' + **this**.state.value);
15. event.preventDefault();
16. }
17. render() {
18. **return** (
19. <form onSubmit={**this**.handleSubmit}>
20. <h1>Controlled Form Example</h1>
21. <label>
22. Name:
23. <input type="text" value={**this**.state.value} onChange={**this**.handleChange} />
24. </label>
25. <input type="submit" value="Submit" />
26. </form>
27. );
28. }
29. }
30. export **default** App;

For More Information, [**Click here**](https://www.javatpoint.com/react-forms).

### 29) What are the different phases of React component's lifecycle?

The different phases of React component's lifecycle are:

**Initial Phase:** It is the birth phase of the React lifecycle when the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component.

**Mounting Phase:** In this phase, the instance of a component is created and added into the DOM.

**Updating Phase:** It is the next phase of the React lifecycle. In this phase, we get new Props and change State. This phase can potentially update and re-render only when a prop or state change occurs. The main aim of this phase is to ensure that the component is displaying the latest version of itself. This phase repeats again and again.

**Unmounting Phase:** It is the final phase of the React lifecycle, where the component instance is destroyed and unmounted(removed) from the DOM.

For More Information, [**Click here**](https://www.javatpoint.com/react-component-life-cycle).

### 30) Explain the lifecycle methods of React components in detail.

The important React lifecycle methods are:

* **getInitialState():** It is used to specify the default value of this.state. It is executed before the creation of the component.
* **componentWillMount():** It is executed before a component gets rendered into the DOM.
* **componentDidMount():** It is executed when the component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
* **componentWillReceiveProps():** It is invoked when a component receives new props from the parent class and before another render is called. If you want to update the State in response to prop changes, you should compare this.props and nextProps to perform State transition by using this.setState() method.
* **shouldComponentUpdate():** It is invoked when a component decides any changes/updation to the DOM and returns true or false value based on certain conditions. If this method returns true, the component will update. Otherwise, the component will skip the updating.
* **componentWillUpdate():** It is invoked before rendering takes place in the DOM. Here, you can't change the component State by invoking this.setState() method. It will not be called, if shouldComponentUpdate() returns false.
* **componentDidUpdate():** It is invoked immediately after rendering takes place. In this method, you can put any code inside this which you want to execute once the updating occurs.
* **componentWillUnmount():** It is invoked immediately before a component is destroyed and unmounted permanently. It is used to clear up the memory spaces such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

For More Information, [**Click here**](https://www.javatpoint.com/react-component-life-cycle).

### 31) What are Pure Components?

Pure components introduced in React 15.3 version. The React.Component and React.PureComponent differ in the shouldComponentUpdate() React lifecycle method. This method decides the re-rendering of the component by returning a boolean value (true or false). In React.Component, shouldComponentUpdate() method returns true by default. But in React.PureComponent, it compares the changes in state or props to re-render the component. The pure component enhances the simplicity of the code and performance of the application.

### 32) What are Higher Order Components(HOC)?

In React, Higher Order Component is an advanced technique for reusing component logic. **It is a function that takes a component and returns a new component**. In other words, it is a function which accepts another function as an argument. According to the official website, it is not the feature(part) in React API, but a pattern that emerges from React's compositional nature.

For More Information, [**Click here**](https://www.javatpoint.com/react-higher-order-components).

### 33) What can you do with HOC?

You can do many tasks with HOC, some of them are given below:

* Code Reusability
* Props manipulation
* State manipulation
* Render highjacking

### 34) What is the difference between Element and Component?

The main differences between Elements and Components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Element** | **Component** |
| **1.** | An element is a plain JavaScript object which describes the component state and DOM node, and its desired properties. | A component is the core building block of React application. It is a class or function which accepts an input and returns a React element. |
| **2.** | It only holds information about the component type, its properties, and any child elements inside it. | It can contain state and props and has access to the React lifecycle methods. |
| **3.** | It is immutable. | It is mutable. |
| **4.** | We cannot apply any methods on elements. | We can apply methods on components. |
| **5.** | **Example:** const element = React.createElement( 'div', {id: 'login-btn'}, 'Login' ) | **Example:** function Button ({ onLogin }) { return React.createElement( 'div', {id: 'login-btn', onClick: onLogin}, 'Login' ) } |

### 35) How to write comments in React?

In React, we can write comments as we write comments in JavaScript. It can be in two ways:

**1. Single Line Comments:** We can write comments as /\* Block Comments \*/ with curly braces:

1. {/\* Single Line comment \*/}

**2. Multiline Comments:** If we want to comment more that one line, we can do this as

1. { /\*
2. Multi
3. line
4. comment
5. \*/ }

### 36) Why is it necessary to start component names with a capital letter?

In React, it is necessary to start component names with a capital letter. If we start the component name with lower case, it will throw an error as an unrecognized tag. It is because, in JSX, lower case tag names are considered as HTML tags.

### 37) What are fragments?

In was introduced in React 16.2 version. In React, Fragments are used for components to return multiple elements. It allows you to group a list of multiple children without adding an extra node to the DOM.

**Example**

1. render() {
2. **return** (
3. <React.Fragment>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </React.Fragment>
8. )
9. }

There is also a shorthand syntax exists for declaring Fragments, but it's not supported in many tools:

1. render() {
2. **return** (
3. <>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </>
8. )
9. }

For More Information, [**Click here**](https://www.javatpoint.com/react-fragments).

### 38) Why are fragments better than container divs?

* Fragments are faster and consume less memory because it did not create an extra DOM node.
* Some CSS styling like CSS Grid and Flexbox have a special parent-child relationship and add <div> tags in the middle, which makes it hard to keep the desired layout.
* The DOM Inspector is less cluttered.

### 39) How to apply validation on props in React?

Props validation is a tool which helps the developers to avoid future bugs and problems. It makes your code more readable. React components used special property PropTypes that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes.

We can apply validation on props using App.propTypes in React component. When some of the props are passed with an invalid type, you will get the warnings on JavaScript console. After specifying the validation patterns, you need to set the App.defaultProps.

1. **class** App **extends** React.Component {
2. render() {}
3. }
4. Component.propTypes = { /\*Definition \*/};

For More Information, [**Click here**](https://www.javatpoint.com/react-props-validation).

### 40) What is create-react-app?

Create React App is a tool introduced by Facebook to build React applications. It provides you to create single-page React applications. The create-react-app are preconfigured, which saves you from time-consuming setup and configuration like Webpack or Babel. You need to run a single command to start the React project, which is given below.

1. $ npx create-react-app my-app

This command includes everything which we need to build a React app. Some of them are given below:

* It includes React, JSX, ES6, and Flow syntax support.
* It includes Autoprefixed CSS, so you don't need -webkit- or other prefixes.
* It includes a fast, interactive unit test runner with built-in support for coverage reporting.
* It includes a live development server that warns about common mistakes.
* It includes a build script to bundle JS, CSS, and images for production, with hashes and source maps.

For More Information, [**Click here**](https://www.javatpoint.com/react-installation).

## React Refs Interview Questions

### 41) What do you understand by refs in React?

Refs is the shorthand used for references in React. It is an attribute which helps to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

For More Information, [**Click here**](https://www.javatpoint.com/react-refs).

### 42) How to create refs?

Refs can be created by using React.createRef() and attached to React elements via the ref attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

1. **class** MyComponent **extends** React.Component {
2. constructor(props) {
3. **super**(props);
4. **this**.callRef = React.createRef();
5. }
6. render() {
7. **return** <div ref={**this**.callRef} />;
8. }
9. }

### 43) What are Forward Refs?

Ref forwarding is a feature which is used for passing a ref through a component to one of its child components. It can be performed by making use of the React.forwardRef() method. It is particularly useful with higher-order components and specially used in reusable component libraries.

**Example**

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **const** TextInput = React.forwardRef((props, ref) => (
5. <input type="text" placeholder="Hello World" ref={ref} />
6. ));
8. **const** inputRef = React.createRef();
10. **class** CustomTextInput **extends** React.Component {
11. handleSubmit = e => {
12. e.preventDefault();
13. console.log(inputRef.current.value);
14. };
15. render() {
16. **return** (
17. <div>
18. <form onSubmit={e => **this**.handleSubmit(e)}>
19. <TextInput ref={inputRef} />
20. <button>Submit</button>
21. </form>
22. </div>
23. );
24. }
25. }
26. export **default** App;

For More Information, [**Click here**](https://www.javatpoint.com/react-refs).

### 44) Which is the preferred option callback refs or findDOMNode()?

The preferred option is to use callback refs over findDOMNode() API. Because callback refs give better control when the refs are set and unset whereas findDOMNode() prevents certain improvements in React in the future.

1. **class** MyComponent **extends** Component {
2. componentDidMount() {
3. findDOMNode(**this**).scrollIntoView()
4. }
5. render() {
6. **return** <div />
7. }
8. }

The recommended approach is:

1. **class** MyComponent **extends** Component {
2. componentDidMount() {
3. **this**.node.scrollIntoView()
4. }
5. render() {
6. **return** <div ref={node => **this**.node = node} />
7. }
8. }
9. **class** MyComponent **extends** Component {
10. componentDidMount() {
11. **this**.node.scrollIntoView()
12. }
13. render() {
14. **return** <div ref={node => **this**.node = node} />
15. }
16. }

### 45) What is the use of Refs?

The Ref in React is used in the following cases:

* It is used to return a reference to the element.
* It is used when we need DOM measurements such as managing focus, text selection, or media playback.
* It is used in triggering imperative animations.
* It is used when integrating with third-party DOM libraries.
* It can also use as in callbacks.

For More Information, [**Click here**](https://www.javatpoint.com/react-refs).

## React Router Interview Questions

### 46) What is React Router?

React Router is a standard routing library system built on top of the React. It is used to create Routing in the React application using React Router Package. It helps you to define multiple routes in the app. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

For More Information, [**Click here**](https://www.javatpoint.com/react-router).

### 47) Why do we need a Router in React?

React Router plays an important role to display multiple views in a single page application. It is used to define multiple routes in the app. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular Route. So, we need to add a Router library to the React app, which allows creating multiple routes with each leading to us a unique view.

1. <**switch**>
2. <h1>React Router Example</h1>
3. <Route path="/" component={Home} />
4. <Route path="/about" component={About} />
5. <Route path="/contact" component={Contact} />
6. </**switch**>

### 48) List down the advantages of React Router.

The important advantages of React Router are given below:

* In this, it is not necessary to set the browser history manually.
* Link uses to navigate the internal links in the application. It is similar to the anchor tag.
* It uses Switch feature for rendering.
* The Router needs only a Single Child element.
* In this, every component is specified in <Route>.
* The packages are split into three packages, which are Web, Native, and Core. It supports the compact size of the React application.

### 49) How is React Router different from Conventional Routing?

The difference between React Routing and Conventional Routing are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Conventional Routing** | **React Routing** |
| **1.** | In Conventional Routing, each view contains a new file. | In React Routing, there is only a single HTML page involved. |
| **2.** | The HTTP request is sent to a server to receive the corresponding HTML page. | Only the History attribute <BrowserRouter> is changed. |
| **3.** | In this, the user navigates across different pages for each view. | In this, the user is thinking he is navigating across different pages, but its an illusion only. |

### 50) Why you get "Router may have only one child element" warning?

It is because you have not to wrap your Route's in a <Switch> block or <div> block which renders a route exclusively.

**Example**

1. render((
2. <Router>
3. <Route {/\* ... \*/} />
4. <Route {/\* ... \*/} />
5. </Router>
6. )

should be

1. render(
2. <Router>
3. <Switch>
4. <Route {/\* ... \*/} />
5. <Route {/\* ... \*/} />
6. </Switch>
7. </Router>
8. )

### 51) Why switch keyword used in React Router v4?

The 'switch' keyword is used to display only a single Route to rendered amongst the several defined Routes. The <Switch> component is used to render components only when the path will be matched. Otherwise, it returns to the not found component.

## React Styling Interview Questions

### 52) How to use styles in React?

We can use style attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in camelCased properties rather than a CSS string. The style attribute is consistent with accessing the properties on DOM nodes in JavaScript.

**Example**

1. **const** divStyle = {
2. color: 'blue',
3. backgroundImage: 'url(' + imgUrl + ')'
4. };
6. function HelloWorldComponent() {
7. **return** <div style={divStyle}>Hello World!</div>
8. }

### 53) How many ways can we style the React Component?

We can style React Component in mainly four ways, which are given below:

* Inline Styling
* CSS Stylesheet
* CSS Module
* Styled Components

For More Information, [**Click here**](https://www.javatpoint.com/react-css).

### 54) Explain CSS Module styling in React.

CSS Module is a CSS file where all class names and animation names are scoped locally by default. It is available only for the component which imports it, and without your permission, it cannot be applied to any other Components. You can create CSS Module file with the .module.css extension.

For More Information, [**Click here**](https://www.javatpoint.com/react-css).

### 55) What are Styled Components?

Styled-Components is a library for React. It is the successor of CSS Modules. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS. It is scoped to a single component and cannot leak to any other element in the page.

The styled-components provides:

* Automatic critical CSS
* No class name bugs
* Easier deletion of CSS
* Simple dynamic styling
* Painless maintenance

For More Information, [**Click here**](https://www.javatpoint.com/react-css).

## React Redux Interview Questions

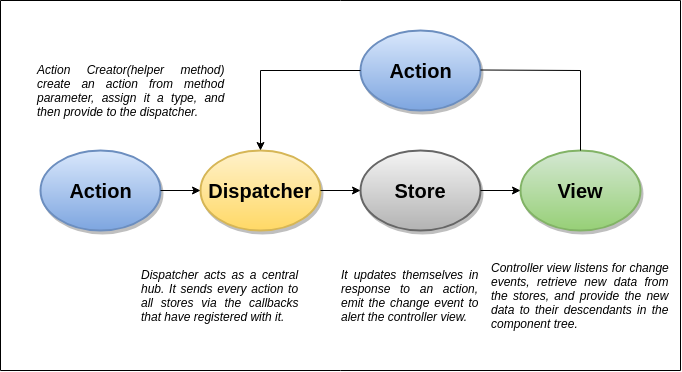
### 56) What were the major problems with MVC framework?

The major problems with the MVC framework are:

* DOM manipulation was very expensive.
* It makes the application slow and inefficient.
* There was a huge memory wastage.
* It makes the application debugging hard.

### 57) Explain the Flux concept.

Flux is an application architecture that Facebook uses internally for building the client-side web application with React. **It is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model**. It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.



For More Information, [**Click here**](https://www.javatpoint.com/react-flux-concept).

### 58) What is Redux?

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. The Redux application is easy to test and can run in different environments showing consistent behavior. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

For More Information, [**Click here**](https://www.javatpoint.com/react-redux).

### 59) What are the three principles that Redux follows?

The three principles that redux follows are:

1. **Single source of truth:** The State of your entire application is stored in an object/state tree inside a single Store. The single State tree makes it easier to keep changes over time. It also makes it easier to debug or inspect the application.
2. **The State is read-only:** There is only one way to change the State is to emit an action, an object describing what happened. This principle ensures that neither the views nor the network callbacks can write directly to the State.
3. **Changes are made with pure functions:** To specify how actions transform the state tree, you need to write reducers (pure functions). Pure functions take the previous State and Action as a parameter and return a new State.

### 60) List down the components of Redux.

The components of Redux are given below.

* **STORE:** A Store is a place where the entire State of your application lists. It is like a brain responsible for all moving parts in Redux.
* **ACTION:** It is an object which describes what happened.
* **REDUCER:** It determines how the State will change.

For More Information, [**Click here**](https://www.javatpoint.com/react-redux).

### 61) Explain the role of Reducer.

Reducers read the payloads from the actions and then updates the Store via the State accordingly. It is a pure function which returns a new state from the initial State. It returns the previous State as it is if no work needs to be done.

### 62) What is the significance of Store in Redux?

A Store is an object which holds the application's State and provides methods to access the State, dispatch Actions and register listeners via subscribe(listener). The entire State tree of an application is saved in a single Store which makes the Redux simple and predictable. We can pass middleware to the Store which handles the processing of data as well as keep a log of various actions that change the Store's State. All the Actions return a new state via reducers.

### 63) How is Redux different from Flux?

The Redux is different from Flux in the following manner.

|  |  |  |
| --- | --- | --- |
| **SN** | **Redux** | **Flux** |
| **1.** | Redux is an open-source JavaScript library used to manage application State. | Flux is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model. |
| **2.** | Store's State is immutable. | Store's State is mutable. |
| **3.** | In this, Store and change logic are separate. | In this, the Store contains State and change logic. |
| **4.** | It has only a single Store. | It can have multiple Store. |
| **5.** | Redux does not have Dispatcher concept. | It has single Dispatcher, and all actions pass through that Dispatcher. |

### 64) What are the advantages of Redux?

The main advantages of React Redux are:

* React Redux is the official UI bindings for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.
* It encourages good 'React' architecture.
* It implements many performance optimizations internally, which allows to components re-render only when it actually needs.
* It makes the code maintenance easy.
* Redux's code written as functions which are small, pure, and isolated, which makes the code testable and independent.

### 65) How to access the Redux store outside a component?

You need to export the Store from the module where it created with createStore() method. Also, you need to assure that it will not pollute the global window space.

1. store = createStore(myReducer)
2. export **default** store

-\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question #1: What is React? How is it different from other JS frameworks?

Although this sounds like a relatively simple question, it’s really asking the candidate to state an informed opinion about React, as well as any competing alternatives. In short, this question is designed to test a candidate's knowledge about the JavaScript ecosystem at large while also pressing for specifics on what makes React unique.

Let’s look at each part of the answer separately.

### ****What is React?****

React is an open-source JavaScript library created by Facebook for building complex, interactive UIs in web and mobile applications.

The key point in this answer is that React’s core purpose is to build UI components; it is often referred to as just the “V” (View) in an “MVC” architecture. Therefore it has no opinions on the other pieces of your technology stack and can be [seamlessly integrated into any application](https://www.codementor.io/javascript/tutorial/should-you-build-your-web-application-with-javascript-mvc-frameworks).

### ****How is React different?****

The answer to this question will likely vary depending on the candidate's personal experiences. The important thing is to listen for real-life examples provided and opinions on whether or not the candidate prefers React and why.

Because React is a small library focused on building UI components, it is necessarily different than a lot of other JavaScript frameworks.

For example, AngularJS (1.x) approaches building an application by extending HTML markup and injecting various constructs (e.g. Directives, Controllers, Services) at runtime. As a result, AngularJS is very opinionated about the greater architecture of your application — these abstractions are certainly useful in some cases, but in many situations, they come at the cost of flexibility.

By contrast, React focuses exclusively on the creation of components, and has few (if any) opinions about an application’s architecture. This allows a developer an incredible amount of flexibility in choosing the architecture they deem “best” — though it also places the responsibility of choosing (or building) those parts on the developer.

I recently migrated an application originally written in AngularJS to React, and one of the things I loved most was…

By comparing and contrasting React with another library, not only can the candidate demonstrate a deep understanding of React, but also position themself as a potentially strong candidate.

Be prepared to ask some follow-up questions as well, such as:

* Under what circumstances would you choose React over another technology? For example, [*React vs Angular*](https://www.codementor.io/codementorteam/react-vs-angular-2-comparison-beginners-guide-lvz5710ha) or [*React vs Vue*](https://www.codementor.io/vuejsdevelopers/react-or-vue-which-javascript-ui-library-should-you-be-using-6hri3num4).
* If React only focuses on a small part of building UI components, can you explain some pitfalls one might encounter when developing a large application?
* If you were rewriting an AngularJS application in React, how much code could you expect to re-use?

## Question #2: What happens during the lifecycle of a React component?

One of the most valuable parts of React is its [component lifecycle](https://facebook.github.io/react/docs/component-specs.html) — so understanding exactly how components function over time is instrumental in building a maintainable application.

### ****High-Level Component Lifecycle****

At the highest level, React components have lifecycle events that fall into three general categories:

1. Initialization
2. State/Property Updates
3. Destruction

Every React component defines these events as a mechanism for managing its properties, state, and rendered output. Some of these events only happen once, others happen more frequently; understanding these three general categories should help you clearly visualize when certain logic needs to be applied.

For example, a component may need to add event listeners to the DOM when it first mounts. However, it should probably remove those event listeners when the component unmounts from the DOM so that irrelevant processing does not occur.

**class** **MyComponent** **extends** **React**.**Component** {

*// when the component is added to the DOM...*

componentDidMount() {

window.addEventListener('resize', **this**.onResizeHandler);

}

*// when the component is removed from the DOM...*

componentWillUnmount() {

window.removeEventListener('resize', **this**.onResizeHandler);

}

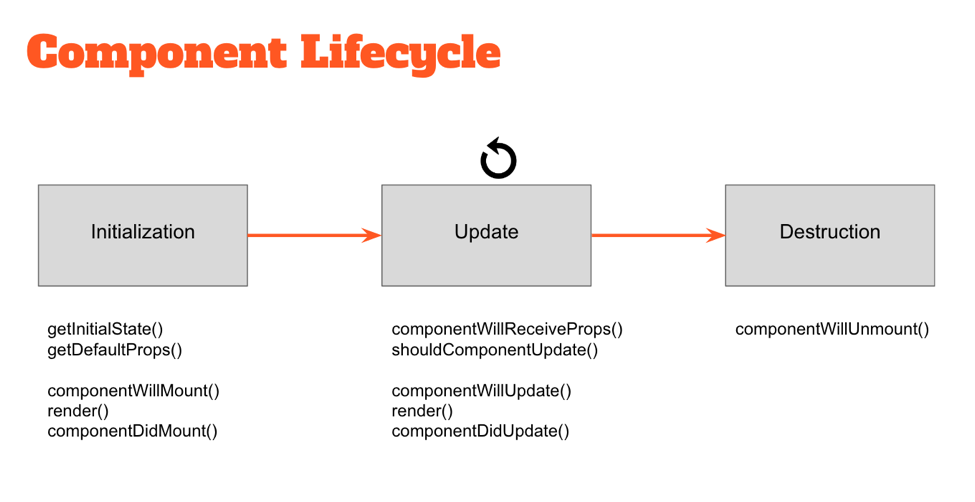
onResizeHandler() {

console.log('The window has been resized!');

}

}

### ****Low-Level Component Lifecycle****



Within these three general buckets exist a number of specific lifecycle hooks — essentially abstract methods — that can be utilized by any React component to more accurately manage updates. Understanding how and when these hooks fire is key to building stable components and will enable you to control the rendering process (improving performance).

Take a look at the diagram above. The events under “Initialization” only happen when a component is first initialized or added to the DOM. Similarly, the events under “Destruction” only happen once (when the component is removed from the DOM). However, the events under “Update” happen every time the properties or state of the component change.

For example, components will automatically re-render themselves any time their properties or state change. However, in some cases a component might not need to update — so preventing the component from re-rendering might improve the performance of our application.

**class** **MyComponent** **extends** **React**.**Component** {

*// only re-render if the ID has changed!*

shouldComponentUpdate(nextProps, nextState) {

**return** nextProps.id === **this**.props.id;

}

}

[](https://www.codementor.io/reactjs-developers)

#### Find top React developers today.

CodementorX will find you the best engineers for your project.

## Question #3: What can you tell me about JSX?

When Facebook first released React to the world, they also introduced a new dialect of JavaScript called JSX that embeds raw HTML templates inside JavaScript code. JSX code by itself cannot be read by the browser; it must be transpiled into traditional JavaScript using tools like Babel and webpack. While many developers understandably have initial knee-jerk reactions against it, JSX (in tandem with ES2015) has become the defacto method of defining React components.

**class** **MyComponent** **extends** **React**.**Component** {

render() {

**let** props = **this**.props;

**return** (

<div className="my-component">

<a href={props.url}>{props.name}</a>

</div>

);

}

}

Asking questions about JSX tests whether or not the candidate can state an informed opinion towards JSX and defend it based on personal experience. Let’s cover some of the basic talking points.

### ****Key Talking Points****

Developers do not have to use JSX (and ES2015) to write an application in React.

This is certainly true. Having said that, many React developers prefer to use JSX as its syntax is far more declarative and reduces overall code complexity. Facebook certainly encourages it in all of their documentation!

Adopting JSX allows the developer to simultaneously adopt ES2015 — giving immediate access to some wonderful syntactic sugar.

ES2015 introduced a variety of new features to JavaScript that makes writing large applications far easier than ever before: [classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes), block scoping via [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let), and the new [spread](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator) operator are just a small portion of the additions.

**import** AnotherClass **from** './AnotherClass';

**class** **MyComponent** **extends** **React**.**Component** {

render() {

**let** props = **this**.props;

**return** (

<div className="my-component">

<AnotherClass {...props} />

</div>

);

}

}

But while ES2015 is becoming more and more widespread, it still is far from widely supported by the major browsers — so tools like Babel or webpack are needed to convert everything into legacy ES5 code.

Candidates that have built a React application using JSX and ES2015 can speak about some specific pros or cons encountered, such as:

Although it took me some time to get used to the JSX and ES2015 syntax, I discovered how much I really enjoyed using it. Specifically, I’m a big fan of…

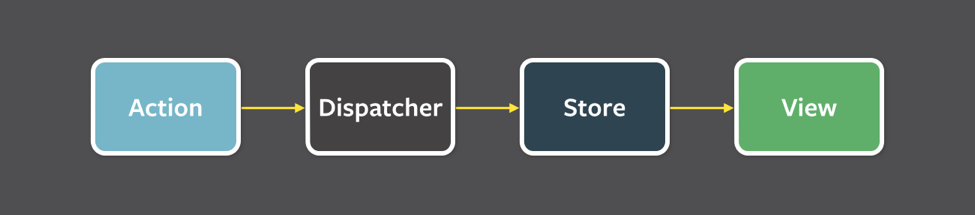
On the other hand, I could do without the hassle of configuring webpack and Babel. Our team ran into issues with…

The React docs on [JSX Gotchas](https://facebook.github.io/react/docs/jsx-gotchas.html) may be good to know/review.

## Question #4: Are you familiar with Flux?

Flux is an architectural pattern that enforces unidirectional data flow — its core purpose is to control derived data so that multiple components can interact with that data without risking pollution.

The Flux pattern is generic; it’s not specific to React applications, nor is it required to build a React app. However, Flux is commonly used by React developers because React components are declarative — the rendered UI (View) is simply a function of state (Store data).



Flux is relatively simple in concept, but in a technical interview, it's important that the developer demonstrates a deep understanding of its implementation. Let’s cover of the important few discussion points.

### ****Description of Flux****

In the Flux pattern, the Store is the central authority for all data; any mutations to the data must occur within the store. Changes to the Store data are subsequently broadcast to subscribing Views via events. Views then update themselves based on the new state of received data.

To request changes to any Store data, Actions may be fired. These Actions are controlled by a central Dispatcher; Actions may not occur simultaneously, ensuring that a Store only mutates data once per Action.

The strict unidirectional flow of this Flux pattern enforces data stability, reducing data-related runtime errors throughout an application.

### ****Flux vs MVC****

Traditional MVC patterns have worked well for separating the concerns of data (Model), UI (View) and logic (Controller) — but many web developers have discovered limitations with that approach as applications grow in size. Specifically, MVC architectures frequently encounter two main problems:

* **Poorly defined data flow:** The cascading updates which occur across views often lead to a tangled web of events which is difficult to debug.
* **Lack of data integrity:** Model data can be mutated from anywhere, yielding unpredictable results across the UI.

With the Flux pattern complex UIs no longer suffer from cascading updates; any given React component will be able to reconstruct its state based on the data provided by the store. The flux pattern also enforces data integrity by restricting direct access to the shared data.

During a technical interview, one should discuss the differences between the Flux and MVC design patterns within the context of a specific example:

For example, imagine we have a “master/detail” UI in which the user can select a record from a list (master view) and edit it using an auto-populated form (detail view).

With an MVC architecture, the data contained within the Model is shared between both the master and detail Views. Each of these views might have its own Controller delegating updates between the Model and the View. At any point the data contained within the Model might be updated — and it’s difficult to know where exactly that change occurred. Did it happen in one of the Views sharing that Model, or in one of the Controllers? Because the Model’s data can be mutated by any actor in the application, the risk of data pollution in complex UIs is greater than we’d like.

With a Flux architecture, the Store data is similarly shared between multiple Views. However this data can’t be directly mutated — all of the requests to update the data must pass through the Action > Dispatcher chain first, eliminating the risk of random data pollution. When updates are made to the data, it’s now much easier to locate the code requesting those changes.

### ****Difference with AngularJS (1.x)****

UI components in AngularJS typically rely on some internal $scope to store their data. This data can be directly mutated from within the UI component or anything given access to $scope — a risky situation for any part of the component or greater application which relies on that data.

By contrast, the Flux pattern encourages the use of immutable data. Because the store is the central authority on all data, any mutations to that data must occur within the store. The risk of data pollution is greatly reduced.

### ****Testing****

One of the most valuable aspects of applications built on Flux is that their components become incredibly easy to test. Developers can recreate and test the state of any React component by simply updating the store — direct interactions with the UI (with tools like [Selenium](http://www.seleniumhq.org/projects/webdriver/)) are no longer necessary in many cases.

### ****Popular Flux Libraries****

While Flux is a general pattern for enforcing data flow through an application, there exist many implementations from which to choose from. There are nuances between each implementation, as well as specific pros and cons to consider. The candidate should provide examples of real-world experience with using Flux.

For example, the candidate might discuss:

* [Redux](http://redux.js.org/): perhaps the most popular Flux library today.
* [Alt.js](http://alt.js.org/): another popular library for managing data in React applications.

## Question #5: What are stateless components?

If React components are essentially state machines that generate UI markup, then what are stateless components?

Stateless components (a flavor of “reusable” components) are nothing more than pure functions that render DOM based solely on the properties provided to them.

**const** StatelessCmp = props => {

**return** (

<div className="my-stateless-component">

{props.name}: {props.birthday}

</div>

);

};

*// ---*

ReactDOM.render(

<StatelessCmp name="Art" birthday="10/01/1980" />,

document.getElementById('main')

);

This component has no need for any internal state — let alone a constructor or lifecycle handlers. The output of the component is purely a function of the properties provided to it.

## Bonus Question: Explain this Code

As I mentioned at the beginning of this article, [technical interviews](http://skillcrush.com/2016/03/29/rock-your-next-whiteboard-test/) may also include time where the developer is asked to look at (and probably write) some code. Take a look at the code below:

**class** **MyComponent** **extends** **React**.**Component** {

**constructor**(props) {

*// set the default internal state*

**this**.state = {

clicks: 0

};

}

componentDidMount() {

**this**.refs.myComponentDiv.addEventListener('click', **this**.clickHandler);

}

componentWillUnmount() {

**this**.refs.myComponentDiv.removeEventListener('click', **this**.clickHandler);

}

clickHandler() {

**this**.setState({

clicks: **this**.clicks + 1

});

}

render() {

**let** children = **this**.props.children;

**return** (

<div className="my-component" ref="myComponentDiv">

<h2>My Component ({this.state.clicks} clicks})</h2>

<h3>{this.props.headerText}</h3>

{children}

</div>

);

}

}

**Given the code defined above, can you identify two problems?**

1. The constructor does not pass its props to the super class. It should include the following line:

**constructor**(props) {

**super**(props);

*// ...*

}

1. The event listener (when assigned via addEventListener()) is not properly scoped because [ES2015 doesn’t provide autobinding](https://facebook.github.io/react/docs/reusable-components.html#no-autobinding). Therefore the developer can re-assign clickHandler in the constructor to include the correct binding to this:

**constructor**(props) {

**super**(props);

**this**.clickHandler = **this**.clickHandler.bind(**this**);

*// ...*

}

**Can you explain what the output of this class actually does? How would you use it in an application?**

This class creates a <div /> element and attaches a click listener to it. The content of this component includes a <h2 /> element that updates every time the user clicks on the parent <div />, as well as an <h3 /> element containing a provided title and whatever child elements were passed to it.

To use this class, the candidate should import it into another class and use it like this:

<MyComponent headerText="A list of paragraph tags">

<p>First child.</p>

<p>Any other <span>number</span> of children...</p>

</MyComponent>

#### 1. How React Works? How Virtual DOM works in React?

React works on Virtual DOM. When a state changes in a component, it runs a diffing algorithm. This identifies that what has been changed in the virtual DOM. The next step is reconciliation which updates the DOM with diff result.

HTML DOM has a tree structure shape that is allowed by the structure of HTML document. DOM trees are big due to the size of large apps. Nowadays, we are looking more towards dynamic apps (Single page applications) so it is important to change the DOM tree constantly a lot. This is a real performance and pain in development.

Virtual DOM is the abstraction of HTML DOM. Virtual DOM is lightweight, and it’s detached from browser-specific implementation details. Virtual DOM is not developed by React, but it uses it and provides it for free.  ReactElement lives inside the virtual DOM. It makes the basic node here. ReactElements are rendered into the “real” DOM once you define the elements.

Once a state is changed in React diff algorithm identifies what has been changed. Then DOM gets updated with the result of diff. Virtual DOM is faster than the regular DOM.

#### 2. What is JSX?

JSX stands for JavaScript XML. It is a syntax extension of JavaScript, and it comes with the full power of JavaScript. JSX create React “elements”. You will see the syntax of JSX wrapped in curly braces. After compilation JSX expressions become the regular JavaScript objects which means you are allowed to use JSX inside if statements and for loops, assign it to a variable, accept it as an argument and return it from functions.

Look at the example given below to understand the syntax of JSX in React..

* Javascript

|  |
| --- |
| const element = (  <h1 className="greeting">  Hello World!  </h1>  ); |

The equivalent of it using createElement is given below…

* Javascript

|  |
| --- |
| const element = React.createElement(  'h1',    {"className":"greeting"},    'Hello World!'  ); |

#### ****3. What is ReactDOM, and what is the Difference Between ReactDOM and React?****

Earlier ReactDOM was part of React but later React and ReactDOM were split into two different libraries. Basically, ReactDOM works like glue between React and the DOM. We can use it for one single thing: mounting with ReactDOM.

ReactDOM.findDOMNode() which is another useful feature of ReactDOM can be used to access the DOM element. For the rest of the things React is there. React is used to define and create the elements, for lifecycle hooks, etc.

#### 4. Difference Between a Class Component and a Functional Component?

In the class component, you can use additional features such as local state and lifecycle hooks. Adding more into it, to enable your component and to have direct access to our store and thus to holds state.

When the component just receives props and renders them to the page. It Is called a ‘stateless component’ for which pure function can be used.

Below is an example of a functional component that is stateless…

* Javascript

|  |
| --- |
| import React from 'react'  const Booklist = books => (    <ul>       {books.map(({ title, author}) =>           {title}--{author}       )}    </ul>  ) |

#### 5. What is the Difference Between State and Props?

When a component mounts in a React app, the state data structure starts with the default value. Across time, it is mutated, mostly as a result of user events. Props which is the shorthand for properties are a Component configuration. Basically, props work like an argument that defines how components talk to each other. Props are immutable and as far as components are receiving them is concerned.

A component can not change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data- callback functions may get passed in as props.

We can also have default props so that props are set even if a parent component doesn’t pass props down. Props and state do the same thing, but both are used in different ways. The majority of components are going to be stateless. Props are used to pass data from parent to child or between the components itself.

Props are immutable and can not be changed. On the other hand, the state is mutable or data that will change. This is specifically useful for user input.

#### 6. What is the Higher-Order Component?

It’s an advanced technique in React for reusing component logic. HOCs are not part of the React API. These are patterns that emerge from React’s compositional nature.

Basically, higher-order components are function that takes a component and returns a new component. HOC’s allows you to reuse the code, logic, and bootstrap abstraction. In third-party React libraries, HOC’s are common.

The most common is Redux’s connect function. Instead of simply sharing utility libraries and simple composition, HOC’s are the best way to share behavior between React Components. You can take advantage of HOC in code refactoring. When you are repeating the same code again and again at different places then use reusable HOC to refactor the code.

#### 7. What is Redux?

Redux is a great way to store the entire application’s state in a single store. When your application is small, you wouldn’t be facing issues in handling the state. But when it starts growing you will find that state in various components is becoming unmanageable. Here Redux solves your problem.

Redux mainly works on three components:

* **Action**: Actions are payloads of information that send data from the application to the store. Actions are the only source of information for the store. We send them to the store using the store.dispatch().
* **Reducer**: Reducer specifies how the applications’ state changes in response to actions sent to the store. Actions describe what happened, but it doesn’t describe how the application’s state changes. Basically, a reducer determines how the state will change to action.
* **Store**: Store objects bring the action and reducer together. You can access the state via **getState()**; It allows the state to be updated via **dispatch (action);**

Store contains javascript objects. You can change the state by firing actions from your application. After that, you can write reducers for these actions and modify the state. The whole transition is kept inside the reducer, and it should not have any side effects.

In Redux there should be only a single source of truth for your application state. It can be a UI state such as which state is active or Data state like the user profile details. These data are retained by Redux in a closure that redux calls a store.

**Note**that you’re only allowed to create a single store in a Redux.

* Javascript

|  |
| --- |
| {    first\_name: 'John',    last\_name: 'Doe',    age: 28  } |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q1. How React works? How Virtual-DOM works in React?**

React creates a virtual DOM. When state changes in a component it firstly runs a “diffing” algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

The HTML DOM is always tree-structured — which is allowed by the structure of HTML document. The DOM trees are huge nowadays because of large apps. Since we are more and more pushed towards dynamic web apps (Single Page Applications — SPAs), we need to modify the DOM tree incessantly and a lot. And this is a real performance and development pain.

The Virtual DOM is an abstraction of the HTML DOM. It is lightweight and detached from the browser-specific implementation details. It is not invented by React but it uses it and provides it for free. ReactElements lives in the virtual DOM. They make the basic nodes here. Once we defined the elements, ReactElements can be render into the "real" DOM.

Whenever a ReactComponent is changing the state, diff algorithm in React runs and identifies what has changed. And then it updates the DOM with the results of diff. The point is - it’s done faster than it would be in the regular DOM.

**Q2. What is JSX?**

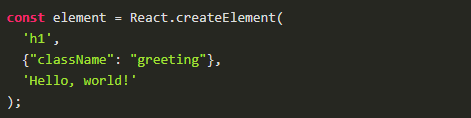
JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React “elements”. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions. Eventhough React does not require JSX, it is the recommended way of describing our UI in React app.

For example, below is the syntax for a basic element in React with JSX and its equivalent without it.

https://miro.medium.com/max/60/1*jdz_xhFqhqiAQaPZGaRjRw.png?q=20

Equivalent of the above using React.createElement

https://miro.medium.com/max/60/1*CdjJk4B7TL5LbZFDLBX8sQ.png?q=20

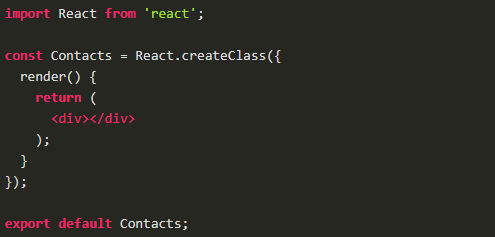


**Q3. What is React.createClass?**

React.createClass allows us to generate component "classes." But with ES6, React allows us to implement component classes that use ES6 JavaScript classes. The end result is the same -- we have a component class. But the style is different. And one is using a "custom" JavaScript class system (createClass) while the other is using a "native" JavaScript class system.

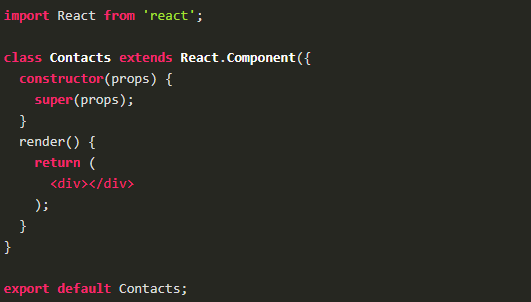
When using React’s createClass() method, we pass in an object as an argument. So we can write a component using createClass that looks like this:

https://miro.medium.com/max/60/1*q58HA3tryv9hpw2dxtpxmg.png?q=20



Using an ES6 class to write the same component is a little different. Instead of using a method from the react library, we extend an ES6 class that the library defines, Component.

https://miro.medium.com/max/60/1*zteV_2CbkJcRO1oHxL-lHQ.png?q=20



constructor() is a special function in a JavaScript class. JavaScript invokes constructor() whenever an object is created via a class.

**Q4. What is ReactDOM and what is the difference between ReactDOM and React?**

Prior to v0.14, all ReactDOM functionality was part of React. But later, React and ReactDOM were split into two different libraries.

As the name implies, ReactDOM is the glue between React and the DOM. Often, we will only use it for one single thing: mounting with ReactDOM. Another useful feature of ReactDOM is ReactDOM.findDOMNode() which we can use to gain direct access to a DOM element.

For everything else, there’s React. We use React to define and create our elements, for lifecycle hooks, etc. i.e. the guts of a React application.

**Q5. What are the differences between a class component and functional component?**

Class components allows us to use additional features such as local state and lifecycle hooks. Also, to enable our component to have direct access to our store and thus holds state.

When our component just receives props and renders them to the page, this is a ‘stateless component’, for which a pure function can be used. These are also called dumb components or presentational components.

From the previous question, we can say that our Booklist component is functional components and are stateless.

https://miro.medium.com/max/60/1*2lHksigcP_3MFjKqCi7NBg.png?q=20



On the other hand, the BookListContainer component is a class component.

**Q6. What is the difference between state and props?**

The state is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component’s configuration. Props are how components talk to each other. They are received from above component and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data — callback functions may be passed in as props.

There is also the case that we can have default props so that props are set even if a parent component doesn’t pass props down.

https://miro.medium.com/max/60/1*a3zDRW805YMtAGVPuFZLMQ.png?q=20



Props and State do similar things but are used in different ways. The majority of our components will probably be stateless. Props are used to pass data from parent to child or by the component itself. They are immutable and thus will not be changed. State is used for mutable data, or data that will change. This is particularly useful for user input.

**Q7. What are controlled components?**

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. A form element whose value is controlled by React in this way is called a "controlled component".

With a controlled component, every state mutation will have an associated handler function. This makes it straightforward to modify or validate user input.

**Q8. What is a higher order component?**

A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API. They are a pattern that emerges from React’s compositional nature.

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

HOC’s allow you to reuse code, logic and bootstrap abstraction. HOCs are common in third-party React libraries. The most common is probably Redux’s connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

**Q9. What is create-react-app?**

create-react-app is the official CLI (Command Line Interface) for React to create React apps with no build configuration.

We don’t need to install or configure tools like Webpack or Babel. They are preconfigured and hidden so that we can focus on the code. We can install easily just like any other node modules. Then it is just one command to start the React project.

https://miro.medium.com/max/60/1*2jC_2JPgxKTEN5bvp3SrUw.png?q=20

https://miro.medium.com/max/651/1*2jC_2JPgxKTEN5bvp3SrUw.png

It includes everything we need to build a React app:

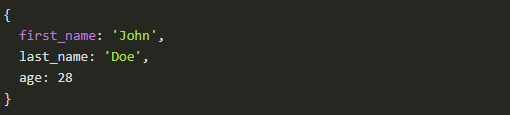
* React, JSX, ES6, and Flow syntax support.
* Language extras beyond ES6 like the object spread operator.
* Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
* A fast interactive unit test runner with built-in support for coverage reporting.
* A live development server that warns about common mistakes.
* A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

**Q10. What is Redux?**

The basic idea of Redux is that the entire application state is kept in a single store. The store is simply a javascript object. The only way to change the state is by firing actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.

Redux is based on the idea that there should be only a single source of truth for your application state, be it UI state like which tab is active or Data state like the user profile details.

https://miro.medium.com/max/60/1*UixQj8JaMzLmXOsHocWYeA.png?q=20

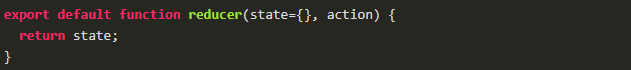


All of these data is retained by redux in a closure that redux calls a store . It also provides us a recipe of creating the said store, namely createStore(x).

The createStore function accepts another function, x as an argument. The passed in function is responsible for returning the state of the application at that point in time, which is then persisted in the store. This passed in function is known as the reducer.

This is a valid example reducer function:

https://miro.medium.com/max/60/1*ejuTaJ-mXMWnPVn8ItwFPA.png?q=20



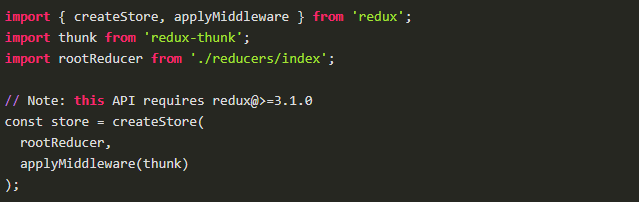
This store can only be updated by dispatching an action. Our App dispatches an action, it is passed into reducer; the reducer returns a fresh instance of the state; the store notifies our App and it can begin it's re render as required.

**Q11. What is Redux Thunk used for?**

Redux thunk is middleware that allows us to write action creators that return a function instead of an action. The thunk can then be used to delay the dispatch of an action if a certain condition is met. This allows us to handle the asyncronous dispatching of actions. The inner function receives the store methods dispatch and getState as parameters.

To enable Redux Thunk, we need to use applyMiddleware() as below

https://miro.medium.com/max/60/1*cCuCnVMum5Y453iFjYQKwQ.png?q=20



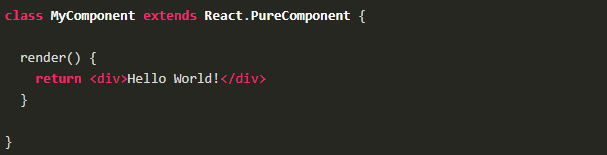
**Q12. What is PureComponent? When to use PureComponent over Component?**

PureComponent is exactly the same as Component except that it handles the shouldComponentUpdate method for us. When props or state changes, PureComponent will do a shallow comparison on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

When comparing previous props and state to next, a shallow comparison will check that primitives have the same value (eg, 1 equals 1 or that true equals true) and that the references are the same between more complex javascript values like objects and arrays.

It is good to prefer PureComponent over Component whenever we never mutate our objects.

https://miro.medium.com/max/60/1*iisL6VZlGGpO2W5wAKTb_g.png?q=20



**Q13. How Virtual-DOM is more efficient than Dirty checking?**

In React, each of our components have a state. This state is like an observable. Essentially, React knows when to re-render the scene because it is able to observe when this data changes. Dirty checking is slower than observables because we must poll the data at a regular interval and check all of the values in the data structure recursively. By comparison, setting a value on the state will signal to a listener that some state has changed, so React can simply listen for change events on the state and queue up re-rendering.

The virtual DOM is used for efficient re-rendering of the DOM. This isn’t really related to dirty checking your data. We could re-render using a virtual DOM with or without dirty checking. In fact, the diff algorithm is a dirty checker itself.

We aim to re-render the virtual tree only when the state changes. So using an observable to check if the state has changed is an efficient way to prevent unnecessary re-renders, which would cause lots of unnecessary tree diffs. If nothing has changed, we do nothing.

**Q14. Is setState() is async? Why is setState() in React Async instead of Sync?**

setState() actions are asynchronous and are batched for performance gains. This is explained in documentation as below.

setState() does not immediately mutate this.state but creates a pending state transition. Accessing this.state after calling this method can potentially return the existing value. There is no guarantee of synchronous operation of calls to setState and calls may be batched for performance gains.

This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. Thus the setState calls are asynchronous as well as batched for better UI experience and performance.

**Q15. What is render() in React? And explain its purpose?**

Each React component must have a render() mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as <form>, <group>, <div> etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

**Q16. What are controlled and uncontrolled components in React?**

This relates to stateful DOM components (form elements) and the difference:

* A **Controlled Component** is one that takes its current value through props and notifies changes through callbacks like onChange. A parent component “controls” it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a “dumb component”.
* A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

In most (or all) cases we should use controlled components.

**Q17. Explain the components of Redux.**

Redux is composed of the following components:

* **Action** — Actions are payloads of information that send data from our application to our store. They are the only source of information for the store. We send them to the store using store.dispatch(). Primarly, they are just an object describes what happened in our app.
* **Reducer** — Reducers specify how the application’s state changes in response to actions sent to the store. Remember that actions only describe what happened, but don’t describe how the application’s state changes. So this place determines how state will change to an action.
* **Store** — The Store is the object that brings Action and Reducer together. The store has the following responsibilities: Holds application state; Allows access to state via getState(); Allows state to be updated via dispatch(action); Registers listeners via subscribe(listener); Handles unregistering of listeners via the function returned by subscribe(listener).

It’s important to note that we’ll only have a single store in a Redux application. When we want to split your data handling logic, we’ll use reducer composition instead of many stores.

**Q18. What is React.cloneElement? And the difference with this.props.children?**

React.cloneElement clone and return a new React element using using the passed element as the starting point. The resulting element will have the original element's props with the new props merged in shallowly. New children will replace existing children. key and ref from the original element will be preserved.

React.cloneElement only works if our child is a single React element. For almost everything {this.props.children} is the better solution. Cloning is useful in some more advanced scenarios, where a parent send in an element and the child component needs to change some props on that element or add things like ref for accessing the actual DOM element.

**Q19. What is the second argument that can optionally be passed to setState and what is its purpose?**

A callback function which will be invoked when setState has finished and the component is re-rendered.

Since the setState is asynchronous, which is why it takes in a second callback function. With this function, we can do what we want immediately after state has been updated.

**Q20. What is the difference between React Native and React?**

React is a JavaScript library, supporting both front end web and being run on the server, for building user interfaces and web applications.

On the other hand, React Native is a mobile framework that compiles to native app components, allowing us to build native mobile applications (iOS, Android, and Windows) in JavaScript that allows us to use ReactJS to build our components, and implements ReactJS under the hood.

With React Native it is possible to mimic the behavior of the native app in JavaScript and at the end, we will get platform specific code as the output. We may even mix the native code with the JavaScript if we need to optimize our application further.

-

#### ****Question****: ****What is React? What are some of its standouts?****

**Answer**: React is a front-end JavaScript library. It was developed by Facebook in 2011. It enhances application performance while allowing for working on both client-side and server-side.

Writing UI test cases is simple with React, which is also easy to integrate with Angular, Meteor, and other popular JS frameworks. Here are some of the major standouts of React:

* Excellent for developing complex and interactive web and mobile UI
* Follows the component-based approach and helps in building reusable UI components
* Features one of the largest community support
* Makes use of the virtual DOM instead of the real DOM
* Relies on server-side rendering
* Supports unidirectional data flow or data binding

#### ****Question****: ****React has something called a state. What is it and how it is used?****

**Answer**: States are the source of data for React components. In other words, they are objects responsible for determining components behavior and rendering. As such, they must be kept as simple as possible.

Accessible by means of this.state(), state is mutable and creates dynamic and interactive components. Use of a state can be visualized by the following code snippet:

**class** **App** **extends** **React**.**Component** {

**constructor**() {

**super**();

**this**.state={

foo: 'bar'

}

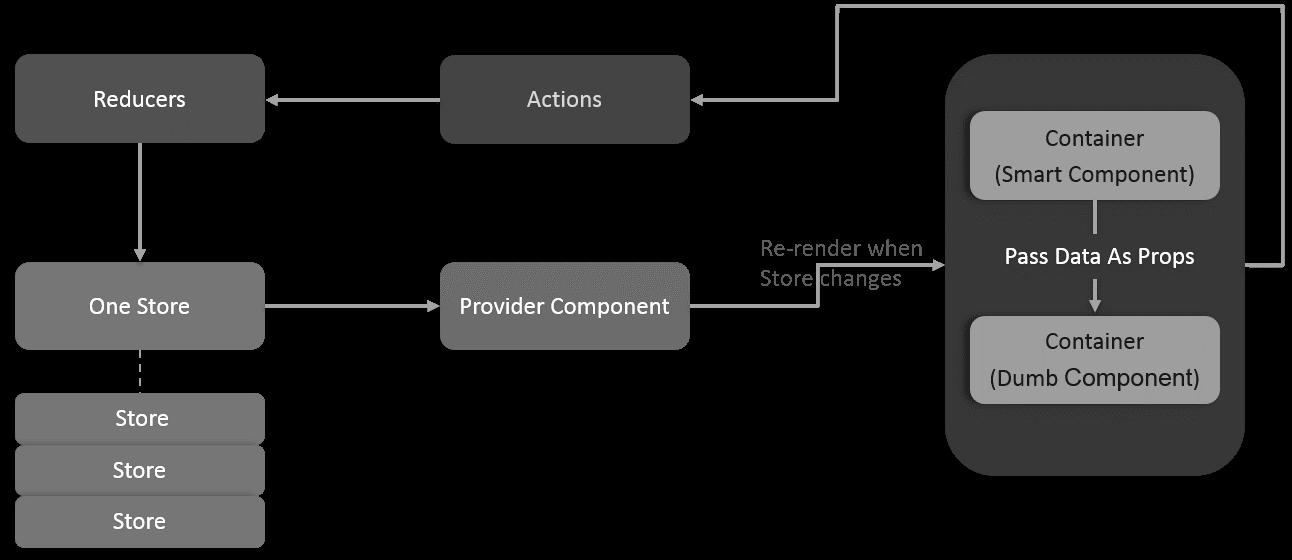
}

#### ****Question****: ****Why do we use render() in React?****

**Answer**: In React, each component must have a render() function. It returns a single React element, which is, in fact, the representation of the native DOM component.

When there is a need for more than one HTML element to be rendered, we group them in one enclosing tag, which can be,or some other tag. There is a need for the render() function to return the same result each time it is invoked i.e. it needs to be kept pure.

#### ****Question****: ****Draw a diagram showing how data flows through Redux.****

**Answer**:

#### ****Question****: ****Explain major differences between the ES5 and ES6 syntax with relevant examples.****

**Answer**: The syntax has witnessed a great change from ES5 to ES6. Important differences between the two releases of ECMAScript are:

* **Require vs. Import – The require used in ES5 is now replaced with import.**var React = require('react'); //is now replaced with  
  import React from 'react'; //in ES6
* **Export vs. Exports –** Instead of exports, now export is used.export default Component; // replaces  
  module.exports = Component; // in ES6
* **Component and Function –** The use of component and function has also changed from ES5 to ES6.

**In ES5:**

**var** MyComponent = React.createClass({

render: **function**() {

return  
Hello World!  
;

}

});

**In ES6:**

**class** **MyComponent** **extends** **React**.**Component** {

render() {

return  
Hello World!  
;

}

}

* **Props – Rules for using props has also changed from ES5 to ES6**

**In ES5:**

**var** App = React.createClass({

propTypes: { name: React.PropTypes.string },

render: **function**() {

return  
Hello, !  
;

}

});

**In ES6:**

**class** **App** **extends** **React**.**Component** {

render() {

return  
Hello, !  
;

}

}

* **State – Using state has also been tweaked for ES6.**

**In ES5:**

**var** App = React.createClass({

getInitialState: **function**() {

**return** { name: 'world' };

},

render: **function**() {

return  
Hello, !  
;

}

});

**In ES6:**

**class** **App** **extends** **React**.**Component** {

**constructor**() {

**super**();

**this**.state = { name: 'world' };

}

render() {

return  
Hello, !  
;

}

}

#### ****Question****: ****Explain the Virtual DOM and its working.****

**Answer**: A virtual DOM is a lightweight JS object. It is simply a copy of the real DOM. A virtual DOM is a node tree that lists various elements, their attributes, and content as objects and their properties.

The render() function in React is responsible for creating a node tree from the React components. This tree is then updated in response to the mutations resulting in the data model due to various actions made by the user or the system.

Virtual DOM operates in three simple steps:

* **Step 1 –** The entire UI is re-rendered in Virtual DOM representation as soon as there are some underlying data changes.
* **Step 2 –** Now, the difference between the previous DOM representation and the new one (resulted from underlying data changes) is calculated.
* **Step 3 –** After the calculations are successfully carried out, the real DOM is updated in line with only the things that actually underwent changes.

#### ****Question****: ****How does the Real DOM differ from the Virtual DOM?****

**Answer**:

* **DOM Manipulation –** Real DOM supports a very expensive DOM manipulation. Virtual DOM, on the contrary, has an inexpensive DOM manipulation.
* **Element Update –** Real DOM creates a new DOM when an element updates. Virtual DOM doesn’t do so in such a case. Instead, it updates the JSX.
* **Memory Wastage –** Real DOM causes a lot of memory wastage while there is no memory wastage for Virtual DOM.
* **Update Speed –** Real DOM updates slowly. On the other end, the virtual DOM updates faster.
* **Updating HTML –** Real DOM can directly update HTML, while virtual DOM can’t update HTML directly.

#### ****Question: Explain various lifecycle methods of React components.****

**Answer**:

* **componentDidMount() –** Executes on the client side after the first render
* **componentDidUpdate() –** Called immediately after rendering takes place in the DOM
* **componentWillMount() –** Executes immediately before rendering starts on both the client-side and the server-side
* **componentWillReceiveProps() –** Invokes when props are received from the parent class and before another render is called
* **componentWillUnmount() –** Used to clear up the memory space. Called right after the component is unmounted from the DOM
* **componentWillUpdate() –** Called immediately before rendering takes place in the DOM
* **shouldComponentUpdate() –** Returns either true or false. Though false by default, needs to be set to return true if the component needs to be updated

#### ****Question****: ****Explain JSX with a code example. Why can’t browsers read it?****

**Answer**: JSX is a contraction of the JavaScript XML. It uses the expressiveness of JavaScript for making the HTML code easily understandable. JSX files make applications robust while boosting their performance. A code example of JSX is:

render(){

**return**(  
React learning made better **by** Hackr.io!!  
);

}

JSX isn’t a regular JS object. The inability of browsers in reading JSX is due to the fact that browsers can only read regular JS objects.

In order to enable a web browser for reading the JSX file, it needs to be transformed into a regular JavaScript object. For this, JSX transformers, like [Babel](https://babeljs.io/), are used.

#### ****Question****: ****Give a code example to demonstrate embedding two or more components into one.****

**Answer**:

**class** **MyComponent** **extends** **React**.**Component**{

render(){

**return**(  
Hello  
);

}

}

**class** **Header** **extends** **React**.**Component**{

render(){

return  
Header Component  
};

}

ReactDOM.render(

, document.getElementById('content')

);

#### ****Question****: ****Give a code example to modularize code in React.****

**Answer**: In order to modularize code in React, export and import properties are used. They assist in writing the components distinctly in different files:

**export** **default** **class** **ChildComponent** **extends** **React**.**Component** {

render() {

**return**(  
This **is** a child component  
);

}

}

**import** ChildComponent **from** './childcomponent.js';

**class** **ParentComponent** **extends** **React**.**Component** {

render() {

**return**(  
);

}

}

#### ****Question****: ****How does the React Router differ from conventional routing?****

**Answer**:

* **Changes in the URL –** A HTTP request is sent to a server for receiving a corresponding HTML page in conventional routing. React routing necessitates only for a change in the History attribute.
* **Navigation –** In conventional routing, the user actually navigates across different web pages for each individual view. In React routing, however, the users feel like they are navigating across distinct webpages while in actuality they aren’t.
* **Pages –** Whereas in React routing only a single HTML page is involved, each view corresponds to a new file in conventional routing.

#### ****Question****: ****How does the state differ from props in React?****

**Answer**:

* Changes inside child components are possible with props but not with state
* Changes inside the component aren’t possible with props but with state
* Props allow for a parent component to change the value, state doesn’t

#### ****Question****: ****How will you distinguish Redux from Flux?****

**Answer**:

* **Components –**React components subscribe to the store in flux whereas in redux, container components utilize connect
* **Dispatcher –** There is no dispatcher in redux. On the other hand, flux has a singleton dispatcher
* **Number of Stores –** While flux has several stores, there is only a single store for redux
* **State –** It is mutable for flux but immutable for redux
* **Store –** Influx, the store contains state as well as change logic. Contrary to this, the store in redux is separate from the change logic
* **Store Type –** All stores in flux are disconnected and flat. This is not the case with redux, where there is a single store with hierarchical reducers

#### ****Question****: ****How would you create a form in React?****

**Answer:** React forms are identical to HTML forms. However, the state is contained in the state property of the component in React and is updateable only via the setState() method.

Therefore, the elements in a React form can’t directly update their state. Their submission is handled by a JS function, which has full access to the data entered into the form by a user.

Following code demonstrates creating a form in React:

handleSubmit(**event**) {

alert('A name was submitted: ' + **this**.state.**value**);

**event**.preventDefault();

}

render() {

**return** (

Top of Form

Name:  

Bottom of Form

);

}

#### ****Question: What are the advantages of using Redux?****

**Answer:**

* **Better Code Organization** – Redux is precise in terms of how the code needs to be organized. This results in a consistent code workable for any development team
* **Developer Tools** – Allow developers to track each and everything, ranging from actions to state changes, happening in the application in real-time
* **Easy Testing** – Redux code is mainly composed of functions that are isolated, pure, and small. Hence, testing is much easy and simple
* **Large-scale Community** – Redux is backed by a mammoth community. It contributes to an ever-growing and refined library and ready-to-use applications
* **Maintainability** – Thanks to a predictable outcome and strict structure, the code is easier to maintain.
* **Output Predictability** – There is no confusion about syncing the current state with actions as well as other parts of the application as there is only a [single source of truth](https://stackoverflow.com/questions/47182888/what-does-the-single-source-of-truth-mean), which is the store
* **Server-side Rendering** – There is a need of only passing the store created on the server-side to the client-side. In addition to this being useful for initial render, it also offers a better user experience because it optimizes the application performance

#### ****Question****: ****What do you understand by Props in React?****

**Answer:** Prop is a contraction for Properties in React. These read-only components need to be kept immutable i.e. pure. Throughout the application, props are passed down from the parent components to the child components.

In order to maintain the unidirectional data flow, a child component is restricted from sending a prop back to its parent component. This also helps in rendering the dynamically generated data.

#### ****Question****: ****Where would you put AJAX calls in your React code?****

**Answer:** It is possible to use any AJAX library with React, such as Axios, jQuery AJAX, as well as the inbuilt browser window.fetch.

Data with AJAX calls need to be added to the componentDidMount() lifecycle method. By doing this, it is possible to use the setState() method for updating component as soon as the data is retrieved.

#### ****Question****: ****Write a sample code to update the state of a component in React?****

**Answer**: State of a component in React is updated with this.setState() as demonstrated in the following code example:

**class** **MyComponent** **extends** **React**.**Component** {

constructor() {

**super**();

**this**.state = {

name: 'Akhil',

id: '101'

}

}

render()

{

setTimeout(()=>)},2000)

**return** (  
Hello  
Your Id **is**  
);

}

}

ReactDOM.render(

, document.getElementById('content')

);

#### ****Question****: ****You must’ve heard that “In React, everything is a component.” What do you understand from the statement?****

**Answer**: The building blocks of a React application’s UI are called components. Any app UI created using React is divisible into a number of small independent and reusable pieces, known as components.  
React renders each of the components independent of each other. Hence, there is no effect of rendering a component on the rest of the app UI.

#### ****Question: What are the distinct features of React?****

**Answer:**The distinct features of React include the following.

1. It uses Virtual DOM in place of Real DOM
2. It applies server-side rendering
3. It follows the unidirectional flow of data
4. It is data binding

#### ****Question: What are the advantages of React?****

**Answer:**There are varied advantages of React which include:

1. It improves the performance of the application.
2. It can be used for the client-side and the customer side.
3. It increases the readability of code's using JSX.
4. It is easy to integrate with various other frameworks, including Angular, Meteor, etc.
5. User Interface becomes more comfortable with React.

#### ****Question: Are there are any limitations to React?****

**Answer:**There are several limitations of React which include:

1. It acts as a library and not as a framework.
2. The contents of the library are so large that it consumes a considerable amount of time to understand.
3. It is difficult to understand for the novice.
4. The coding process becomes more complicated when inline templating and JSX are applied.

#### ****Question: Can browsers read JSX?****

**Answer:**No, the browsers cannot read JSX because it is not a regular JavaScript object.

#### ****Question: What are the differences between React and Angular?****

**Answer:**There are several differences between React and Angular, which include the following:

|  |  |  |
| --- | --- | --- |
| **Basis of Difference** | **React** | **Angular** |
| Architecture | It only supports the view of MVC | It supports a complete MVC view |
| Render | It offers server-side rendering | It offers client-side rendering |
| DOM | It applies virtual DOM | It applies real DOM |
| Data binding | It supports one-way data binding | It supports two-way data binding |
| Debug | It applies to compile-time debugging | It applies runtime debugging |
| Developer | Developed by Facebook | Developed by Google |

#### ****Question: What is the relation between React and its components?****

**Answer:**React, and its components are closely related. The components of React acts as the building blocks of React application for the user interface. The splitting up of this entire user interface in different small, independent, and reusable pieces help in creating React application's User Interface.

#### ****Question: What is Props in React?****

**Answer:**Props is referred to the properties in React, including immutable, child components, parent component, etc.

#### ****Question: What are states in React?****

**Answer:**States in react acts as a source of data and are kept simple so that the objects which determine component rendering and behavior become mutable other than props and develop a dynamic and interactive component.

#### ****Question: Can parent component change value in States and Props?****

**Answer:**The parent component can change the value in Props but not in the state.

#### ****Question: Can changes be made inside the component?****

**Answer:**The changes can be made inside the state but not in Props.

#### ****Question: Can we make changes inside child components?****

**Answer:**Yes, we can make changes inside the child component in Props but not in the case of States.

#### ****Question: What is a Stateful component?****

**Answer:**A Stateful component stores the change in memory to React. It has the authority to change state and contains vital information of past, current, and future changes.

#### ****Question: How is Stateless component different from a Stateful component?****

**Answer:**The stateless component calculates the internal state of the component but does not have the authority to change state. There is no knowledge about the past, current, or future but receives props from the Stateful component, which are treated as a callback function.

#### ****Question: Define Synthetic Events in React?****

**Answer:**The Synthetic Events in React are the objects in React, which acts as a cross-browser wrapper around the browser's native event. The main purpose is to combine the different browsers on the API so that the event shows various properties.

#### ****Question: Define Refs in React?****

**Answer:**Refs stands for References to React. It helps in storing a reference to a particular react element or component that can be returned by the component render configuration function.

#### ****Question: When are Refs mostly used?****

**Answer:**Refs are mostly used in the following cases:

1. When there is a need to manage focus, select the text, or apply media playback.
2. To initiate imperative animations.
3. To join with the third-party DOM libraries.

#### ****Question: Can we modularize code in React? How?****

**Answer:**Yes, we can modularize code in React. It can be done by using export and import properties.

#### ****Question: What are the controlled components in React?****

**Answer:**The controlled components in React are referred to as those components which can maintain their state. The data is controlled by their parent component, and they take into consideration the current values using props and thereafter, notify the changes using callbacks.

#### ****Question: How are uncontrolled components different from controlled components?****

**Answer:**The uncontrolled components maintain their state, and their data is controlled by DOM. The Refs are used in uncontrolled components to get their current values instead of using props in case of controlled components.

#### ****Question: Define HOC in React?****

**Answer:**The full form of HOC is the Higher-Order Component. It is an advanced way of reusing the component logic, which wraps another component along with it.

#### ****Question: What are the benefits of HOC?****

**Answer:**There are several benefits of HOC, which include the following:

1. Reuse of Code.
2. Application of logic and bootstrap abstraction
3. Offers a high hacking facility
4. Supports state abstraction and manipulation
5. It offers props manipulation

#### ****Question: What are pure components?****

**Answer:**Pure components include the components which are simple and easy to be written. They can easily replace any component which as a render().

#### ****Question: Define Reducers in React?****

**Answer:**Reducers are the pure functions that clearly states as to how the application state changes when certain actions are made. This way, it takes into account the previous state and action to turn out to a new state.

#### ****Question: What is a store in Redux?****

**Answer:**A store in Redux is a JavaScript object that can hold applications state and provide help to access them and applying dispatch actions along with register listeners.

#### ****Question: Why do we need a Router to React?****

**Answer:**We need a Router to React so that we could define the multiple routes whenever the user types a particular URL. This way, the application of a particular router can be made when the URL matches the path defined inside the router.