[https://github.com/sudheerj/reactjs-interview-questions#what-is-react](https://github.com/sudheerj/reactjs-interview-questions" \l "what-is-react)

What is React?

React is an open source JS library for creating rich user interfaces that run on user’s web browser. It is used for handling view layer for web and mobile apps.

What are the major features of React?

Major feature of React are:

It uses virtual DOM instead of Real DOM considering that Real DOM manipulations are expensive.

Supports server-side rendering.

Follows unidirectional data flow or data binding.

Uses reusable or composable UI components to develop the view.

What is JSX?

JSX is XML like syntax extension to ECMAScript (Javascript XML). Basically it just provides syntactic sugar for React.createElement function, giving us expressiveness of JS along with HTML like template syntax.

How to create components in React?

Two possible ways:

Functional component :

const greeting = (props) => {

return <h1>{props.greet}</h1>

}

Or

function greetings(){

return <h1>Greetings</h1>

}

Class Component :

Class MyComponent extends Component{

render(){

return (

<div>

<h1>Greetings</h1>

</div>

)

}

}

When to use 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 addition of Hooks, you could use state, lifecycle methods and other features that were only available in class component right in your function component.

What are Pure Components?

Pure Components is exactly the same as React.Component ,except that it handles the shouldComponentUpdate() method for you.

When state or props 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.

If your React component’s render() function renders the same result given the same props and state, you can use React.PureComponent for a performance boost in some cases.

What is state in React?

State of a component is an object that holds some information that may change over the lifetime of a component. They are private and fully controlled by the component i.e it is not accessible to any other component than the one that owns and sets it.

What is props in react?

Props are input 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 from the parent to the child component.

Difference between state and props?

Both props and state are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

Why should we not update the state directly?

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

//Wrong

this.state.message = 'Hello world'

Instead use setState() method. It schedules an update to a component's state object. When state changes, the component responds by re-rendering.

//Correct

this.setState({ message: 'Hello World' })

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'))

<https://medium.com/better-programming/when-to-use-callback-function-of-setstate-in-react-37fff67e5a6c>

### 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 should be in lowercase:

<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)

<https://medium.com/tfogo/advantages-and-pitfalls-of-arrow-functions-a16f0835799e>

<https://www.freecodecamp.org/news/this-is-why-we-need-to-bind-event-handlers-in-class-components-in-react-f7ea1a6f93eb/>

### How to bind methods or event handlers in JSX callbacks?

There are 3 possible ways to achieve this:

**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.

class Component extends React.Componenet {

constructor(props) {

super(props)

this.handleClick = this.handleClick.bind(this)

}

handleClick() {

// ...

}

}

**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.

handleClick = () => {

console.log('this is:', this)

}

<button onClick={this.handleClick}>

{'Click me'}

</button>

**Arrow functions in callbacks:** You can use arrow functions directly in the callbacks.

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

{'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.

### 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 array function

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

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

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

};

Arrow functions and function expressions

What’s the difference between context and scope? The context is (roughly) the ***object*** that ***calls*** the function. And the scope is all the ***variables*** visible to a function ***where it is defined***. One cares about how it is called, the other cares about how it is defined.

### 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.

### 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.

### **Differences between Shadow DOM and Virtual DOM**

The only thing which is common for both is that they help with performance issues. Both create a separate instance of the Document Object Model; besides this, both concepts are different. Virtual DOM is creating a copy of the whole DOM object, and Shadow DOM creates small pieces of the DOM object which has their own, isolated scope for the element they represent.

<https://www.blog.duomly.com/what-is-the-difference-between-shadow-dom-and-virtual-dom/#:~:text=Differences%20between%20Shadow%20DOM%20and%20Virtual%20DOM&text=Virtual%20DOM%20is%20creating%20a,for%20the%20element%20they%20represent.>

### 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()})

}

### 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.

### 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.

### 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 here 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.

### 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.

### 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>

npm install –save jest

For testing

npm install –save enzyme react-test-renderer enzyme-adapter-react-16

React Router

npm install –save react-router react-router-dom

import {BrowserRouter} from ‘react-router-dom’

only react-router-dom  is required for web development. It wraps react-router  and therefore uses it as a dependency.

import {Route} from ‘react-router-dom’;

<Route path=’/’ exact render={() => <h1>Home</h1>} />

<Route path=’/’ exact component={Posts} />

But this reloads the page. We don’t want reload.. we want only re-render.

For this we use <Link to=”/” >Home</Link>

<Link to={

{

pathname: ‘/new-post’,

hash: ‘#submit’,

search: ‘?quick-submit=true’

}

} >Home</Link>

Difference between React and React DOM

From the [React v0.14 Beta release announcement](https://facebook.github.io/react/blog/2015/07/03/react-v0.14-beta-1.html).

As we look at packages like [react-native](https://facebook.github.io/react-native/), [react-art](https://github.com/reactjs/react-art), [react-canvas](https://github.com/Flipboard/react-canvas), and [react-three](https://github.com/Izzimach/react-three), it's become clear that the beauty and essence of React has nothing to do with browsers or the DOM.

To make this more clear and to make it easier to build more environments that React can render to, we're splitting the main react package into two: react and react-dom.

Fundamentally, the idea of React has nothing to do with browsers, they just happen to be one of many targets for rendering trees of components into. The ReactDOM package has allowed the developers to remove any non-essential code from the React package and move it into a more appropriate repository.

The react package contains React.createElement, React.createClass and React.Component, React.PropTypes, React.Children, and the other helpers related to elements and component classes. We think of these as the isomorphic or universal helpers that you need to build components.

The react-dom package contains ReactDOM.render, ReactDOM.unmountComponentAtNode, and ReactDOM.findDOMNode, and in react-dom/server we have server-side rendering support with ReactDOMServer.renderToString and ReactDOMServer.renderToStaticMarkup.

These two paragraphs explain where the core API methods from v0.13 ended up.