React Interview Preparation

What is React?

React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.

React is a tool for building UI components.

## How does React Work?

## React creates a VIRTUAL DOM in memory

## Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

## React only changes what needs to be changed!

## React finds out what changes have been made, and changes **only** what needs to be changed.

## React.JS History

## Initial Release to the Public (V0.3.0) was in July 2013.

* Current verstion:18.2.0 [Changelog](https://github.com/facebook/react/blob/main/CHANGELOG.md#1820-june-14-2022)
* To use React in production, you need npm which is included with [Node.js](https://nodejs.org/).

## The Render Function

The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

The purpose of the function is to display the specified HTML code inside the specified HTML element.

But render where?

There is another folder in the root directory of your React project, named "public". In this folder, there is an index.html file.

You'll notice a single <div> in the body of this file. This is where our React application will be rendered.

## What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

## Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement()  and/or appendChild() methods.

JSX converts HTML tags into react element

You are not required to use JSX, but JSX makes it easier to write React applications.

## React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8. There is an optional section on Class components for your reference.

### Class Component

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

### Function Component

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

### Function Component

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

## Lifecycle of Components

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: **Mounting**, **Updating**, and **Unmounting**.

### componentDidMount

The componentDidMount() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

import React from 'react';

import ReactDOM from 'react-dom/client';

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

componentDidMount() {

setTimeout(() => {

this.setState({favoritecolor: "yellow"})

}, 1000)

}

render() {

return (

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

## Updating

The next phase in the lifecycle is when a component is updated.

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

The render() method is required and will always be called, the others are optional and will be called if you define them.

### getDerivedStateFromProps

Also at updates the getDerivedStateFromProps method is called. This is the first method that is called when a component gets updated.

This is still the natural place to set the state object based on the initial props.

The example below has a button that changes the favorite color to blue, but since the getDerivedStateFromProps() method is called, which updates the state with the color from the favcol attribute, the favorite color is still rendered as yellow:

### Example:

If the component gets updated, the getDerivedStateFromProps() method is called:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

static getDerivedStateFromProps(props, state) {

return {favoritecolor: props.favcol };

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header favcol="yellow" />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_getderivedstatefromprops2)

### shouldComponentUpdate

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.

The default value is true.

The example below shows what happens when the shouldComponentUpdate() method returns false:

### Example:

Stop the component from rendering at any update:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

shouldComponentUpdate() {

return false;

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_shouldcomponentupdate)

### Example:

Same example as above, but this time the shouldComponentUpdate() method returns true instead:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

shouldComponentUpdate() {

return true;

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_shouldcomponentupdate2)

### render

The render() method is of course called when a component gets updated, it has to re-render the HTML to the DOM, with the new changes.

The example below has a button that changes the favorite color to blue:

### Example:

Click the button to make a change in the component's state:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_render2)

### getSnapshotBeforeUpdate

In the getSnapshotBeforeUpdate() method you have access to the props and state before the update, meaning that even after the update, you can check what the values were before the update.

If the getSnapshotBeforeUpdate() method is present, you should also include the componentDidUpdate() method, otherwise you will get an error.

The example below might seem complicated, but all it does is this:

When the component is mounting it is rendered with the favorite color "red".

When the component has been mounted, a timer changes the state, and after one second, the favorite color becomes "yellow".

This action triggers the update phase, and since this component has a getSnapshotBeforeUpdate() method, this method is executed, and writes a message to the empty DIV1 element.

Then the componentDidUpdate() method is executed and writes a message in the empty DIV2 element:

### Example:

Use the getSnapshotBeforeUpdate() method to find out what the state object looked like before the update:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

componentDidMount() {

setTimeout(() => {

this.setState({favoritecolor: "yellow"})

}, 1000)

}

getSnapshotBeforeUpdate(prevProps, prevState) {

document.getElementById("div1").innerHTML =

"Before the update, the favorite was " + prevState.favoritecolor;

}

componentDidUpdate() {

document.getElementById("div2").innerHTML =

"The updated favorite is " + this.state.favoritecolor;

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<div id="div1"></div>

<div id="div2"></div>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_getsnapshotbeforeupdate)

### componentDidUpdate

The componentDidUpdate method is called after the component is updated in the DOM.

The example below might seem complicated, but all it does is this:

When the component is mounting it is rendered with the favorite color "red".

When the component has been mounted, a timer changes the state, and the color becomes "yellow".

This action triggers the update phase, and since this component has a componentDidUpdate method, this method is executed and writes a message in the empty DIV element:

### Example:

The componentDidUpdate method is called after the update has been rendered in the DOM:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

componentDidMount() {

setTimeout(() => {

this.setState({favoritecolor: "yellow"})

}, 1000)

}

componentDidUpdate() {

document.getElementById("mydiv").innerHTML =

"The updated favorite is " + this.state.favoritecolor;

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<div id="mydiv"></div>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

[Run Example »](https://www.w3schools.com/react/showreact.asp?filename=demo2_react_lifecycle_componentdidupdate)

## Unmounting

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

* componentWillUnmount()

### componentWillUnmount

The componentWillUnmount method is called when the component is about to be removed from the DOM.

### Example:

Click the button to delete the header:

class Container extends React.Component {

constructor(props) {

super(props);

this.state = {show: true};

}

delHeader = () => {

this.setState({show: false});

}

render() {

let myheader;

if (this.state.show) {

myheader = <Child />;

};

return (

<div>

{myheader}

<button type="button" onClick={this.delHeader}>Delete Header</button>

</div>

);

}

}

class Child extends React.Component {

componentWillUnmount() {

alert("The component named Header is about to be unmounted.");

}

render() {

return (

<h1>Hello World!</h1>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Container />);