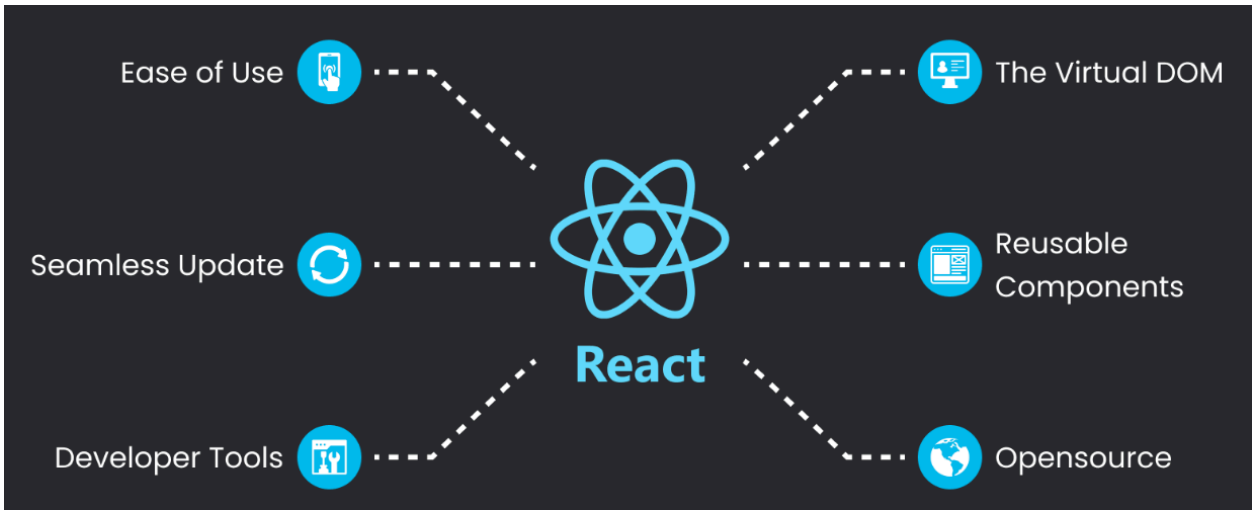
React JS



React is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta and a community of individual developers and companies. Wikipedia

Initial release date: 29 May 2013

Platform: Web platform

Developer(s): Meta and community

Original author(s): Jordan Walke

License: MIT License

Stable release: 18.2.0 / 14 June 2022; 8 months ago

Programming language: JavaScript

Official Website
https://reactjs.org/

Latest Version - 18

Tools to know

1. **Figma**
2. **Browser Stack**
3. **Rest client / PostMan**
4. **Redux dev tool**
5. **Git Bash and git commands.**

==========

Why do we use ReactJS?

The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. It uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses components and data patterns that improve readability and helps to maintain larger apps.

Content -

- React State
- React Props
- Styling React Using CSS(inline, internal, external)
- React Lists handle(e.g - array, objects) Using map function
- React Conditional Rendering
- React using JSON Data(Mock/Dummy Data or Production/Live Data - Using Internal and .json file)
- React Event (onChange, onClick)
- React Forms (onSubmit)
- React Validation
- React Router

==========

- React Hooks (Used in Functional Components)
- React useState Hook
- React useEffect Hooks
- React useRef Hook
- React Memo and Pure components
- React context API (useContext Hook)

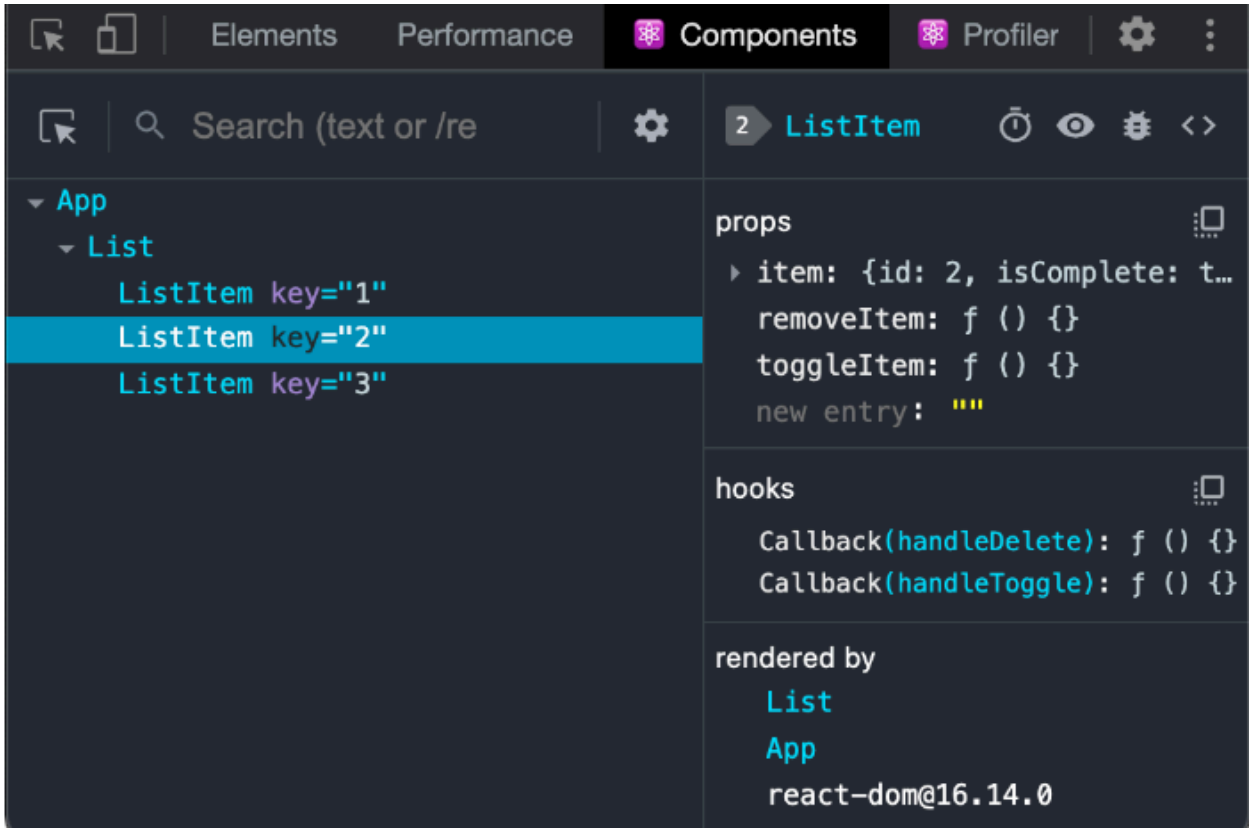
- React Fragments.
- React Lifecycle Methods (Used in Class Components)

    - Mounting
        - constructor()
        - getDerivedStateFromProps()
        - render()
        - componentDidMount()
    - Updating
        - getDerivedStateFromProps()
        - shouldComponentUpdate()
        - render()
        - getSnapshotBeforeUpdate()
        - componentDidUpdate()
    - Unmounting
        - componentWillUnmount()

- React Higher order component

React Dev Tool -

https://react.dev/learn/react-developer-tools



React 17 and React 18.

Mostly there are internal code changes that have been done by the react team in react 17 and 18 to improve the performance.

Like -

In React 17, if you change the state of the component two times, the component will re-render two times. Now, in React 18, the two updates will be batched, and the component will render only once. And that's only if you're using createRoot instead of render like below.

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

root.render(<App />);
```

React

The library for web and native user interfaces

React is an Open Source view library created and maintained by Facebook. It's a great tool to render the User Interface (UI) of modern web applications.

React is a popular JavaScript library for building reusable, component-driven user interfaces for web pages or applications.

React combines HTML with JavaScript functionality into its own markup language called JSX. React also makes it easy to manage the flow of data throughout the application.

What is the Virtual DOM?

The virtual DOM (VDOM) is a programming concept where virtual representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM.

Whenever you make changes to the state of any particular element of your component, virtual DOM takes it in its memory and only re-render that particular element and does not render other elements.

For example - whenever you click on a like button or ratings or reaction buttons , the state of that particular element gets changed , So only that particular like will get re-rendered and the count will increase But not your post component or entire web page will get re-render.

Is the Real DOM the same as the Virtual DOM?

A virtual DOM is the same as a real DOM, except one thing.
Upon any change of a property or state of any component, React Virtual DOM updates and re-render that particular component only and not the entire tree. But Real DOM updates and re-render entire document tree.

Is the Shadow DOM the same as the Virtual DOM?

No, they are different. 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 React libraries in JavaScript on top of browsers.

Create React Project -

Open VS code terminal or command prompt in any drive location and run below command.

> npx create-react-app app_name

1. Create a Simple JSX Element and rendering

JSX- (Javascript And XML)

React uses a syntax extension of JavaScript called JSX that allows you to write HTML directly within JavaScript. This has several benefits. It lets you use the full programmatic power of JavaScript within HTML, and helps to keep your code readable.

```
const simpleJSX = <h1>Hello JSX!!</h1>

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

root.render(

  <div>

     {simpleJSX}

  </div>

);
```

2. Create a Complex JSX Element

When rendering multiple elements, you can wrap them all in parentheses.

```
const complexJSX = (
  <div>
    <h1>This is having multiple Html Tags</h1>
    <h2>Hello JSX!!</h2>
    <ul>
      <li>Good Morning</li>
      <li>Good Evening</li>
    </ul>
  </div>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
     {complexJSX}
  </div>
);
```

3. Add Comments in JSX

To put comments inside JSX, you use the syntax {/* */} to wrap

around the comment text.

{/

/}

React Js functional component or stateless components -

Components are the core of React. Everything in React is a component.

There are two ways to create a React component. The first way is to use a **JavaScript function.** Defining a component in this way creates a *stateless functional component*.

Syntax -

```
Function functionName () {

Return (<h1>Hi</h1>)
}
export default functionName;
```

Example -
Create a DemoComp.js file.

```
function DemoComp() {
    let studentData = {
        name: "Alex",
        College: "IIT Delhi"
    }

    return (
        <div>
            <h3>I am Simple Component</h3>
```

```
            <p>{studentData.name}</p>
            <p>{studentData.College}</p>
        </div>
    )
}
export default DemoComp;
```

Import this file inside index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import DemoComp from './DemoComp;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
    <DemoComp />
  </div>

);
```

Using map function inside functional component to show list of items.

map() creates a new array from calling a function for every array element. map() does not execute the function for empty elements. map() does not change the original array.

```
function MyFavColors() {
    let colors = ["Red", "blue","white","yellow","pink"];
    const divStyle = {
```

```
        backgroundColor : "blue"
    }
    return (
        <ul>
        {colors.map(items => <li>{items}</li>)}
        </ul>
    )
}

export default MyFavColors;
```

Using objects inside functional components and rendering them.

```
function UsingObjectsInsideFunctionalComp() {
    let userData = [
        {
            "userId": 1,
            "id": 1,
            "title": "First Title"
        },
        {
            "userId": 1,
            "id": 2,
            "title": "Second Title"
        },
        {
            "userId": 1,
            "id": 3,
            "title": "Third Title"
        }
    ]

    return (
        <div>
```

```
            {userData.map(itms => (
                <ul>
                    <li>User ID : {itms.userId}</li>
                    <li>ID :{itms.id}</li>
                    <li>Title :{itms.title}</li>
                </ul>
            )
            )}
        </div>
    )
}

export default UsingObjectsInsideFunctionalComp;
```

Import UsingObjectsInsideFunctionalComp inside index.js file and call the component like below.

```
root.render(
  <div>
     <UsingObjectsInsideFunctionalComp  />
  </div>

);
```

Using .json file and import data and render on UI -

User-list.json

```
[{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
},
{
```

```
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
},
{
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
},
{
    "userId": 1,
    "id": 4,
    "title": "eum et est occaecati",
    "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit"
}];
```

MapFunctionComponent.js

```
import userData from './user-list.json';

const MapFunctionComponent = () => {
    return (
        <div>
            <h1>List of Mobiles</h1>
            <div>
                {
                    userData.map(values => (
                        <ul key={values.id}>
                            <li>User ID : {values.userId}</li>
                            <li>ID : {values.id}</li>
```

```
                            <li>Title : {values.title}</li>
                            <li>Details: {values.body}</li>
                        </ul>
                    ))
                }
            </div>
        </div>
    )
}
export default MapFunctionComponent;
```

Nested components -

Calling one or multiple components inside another component and so on…

Components are reusable and u can call them anywhere - let say inside index.js or inside App.js or inside any other component u created.

App.js

```
import SimpleComp from './SimpleComp';
function App() {
  return (
    <div>
      <h1>Hello React- Using Bootstrap</h1>
      <SimpleComp />
    </div>
  );
}

export default App;
```

SimpleComp.js file

```
function SimpleComp () {
    return (
        <div>
```

```
            <p>Hello</p>
        </div>
    )
}

export default SimpleComp;
```

Now your SimpleComp is imported/called inside App component, So you have to render App component inside index.js file like below

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
     <App />
  </div>
);
```

Using CSS -

We can use CSS in three ways -

1. ***Inline css*** - write css inside HTML Tag for example

```
<div style="background-color: #777; width: 300px; height: auto;
padding: 5px"></div>
```

2. ***Internal Css -*** you can also write css inside <head> tag like below

```
<head>
    <title>React App Title</title>
```

```
    <style type="text/css">
      .myStyle {
        background-color: #777;
        width: 300px;
        height: auto;
        padding: 5px
      }
    </style>
  </head>
  <body>
    <!-- Inline css -->
    <div id="root" class="myStyle"></div>
```

3. ***External css*** **-** you can create a .css file , write your css there and include inside HTML page

style.css

```
.myStyle {
    background-color: blue;
    width: 300px;
    height: auto;
    padding: 5px;
    border-color : red;
    border-width: 2px;
}
```

Include style.css file inside html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React App Title</title>
    <link rel="stylesheet" href="./style.css">
  </head>
  <body>
    <!-- Inline css -->
    <div id="root" class="myStyle"></div>
```

Using All Above Css types inside React Js -

Using inline CSS inside react -

```
function UserDataCompWithInlineCss() {
    let userData = [
        {
          "id": 1,
          "title": "sunt aut facere provident"
        },
        {
          "id": 2,
          "title": "qui est esse"
        }];

 return (
   userData.map(items => (
    <ul style={{backgroundColor: "#777", width: "300px",
      padding: "5px", borderColor : "red"}}>
      <li style={{color: "#000", padding: "5px"}}>{items.id}</li>
      <li style={{color: "#000",padding: "5px"}}>{items.title}</li>
    </ul>
   ))
 )
};
export default UserDataCompWithInlineCss;
```

Note ➖

When you have to write inline or internal css you have to do following -

1. Always write css as an object having a key value pair.
   {
   color : “white”,
   backgroundColor : “Black”,
   borderColor : “yellow”,
   textDecoration : “underline”
   }

2. Css properties written with (dash or hyphen that is **-**), hyphen will be removed and next letter will be capital like -
   background-color should be **backgroundColor**
   Border-color should be **borderColor**
   Text-decoration should be **textDecoration** and so on….

Internal CSS -

You can store css properties into javascript variables and then pass these variable names inside style like style={userDataUlStyle} as below.

```
function UserDataCompWithInternalCss() {
    let userData = [{
            "id": 1,
            "title": "sunt aut",
         },{
            "id": 2,
            "title": "qui est esse",
         }];
      // Internal CSS
    let userDataUlStyle = {
        backgroundColor: "#777",
        width: "300px",
        height: "auto",
        padding: "5px",
        borderColor: "red",
        borderWidth: "2px",
        borderRadius: "10px",
        boxShadow: "1px 4px blue"
    }
    let listStyle = {
        color: "#000",
        padding: "5px"
    };
    return (
        <div>
            <h1>Using Internal Css</h1>
            {
                userData.map(items => (
                    <ul style={userDataUlStyle}>
```

```
                        <li style={listStyle}>Id : {items.id}</li>
                        <li style={listStyle}>{items.title}</li>
                    </ul>
                ))
            }
        </div>
    )
};

export default UserDataCompWithInternalCss;
```

External css -

You can create a **.css** file and write your css there just like you write a normal css as below

Mystyle.css

```
.ulStyle {
        background-color: #777;
        width: 300px;
        height: auto;
        padding: 5px;
        border-color: red;
        border-width: 2px;
        border-radius: 10px;
        box-shadow: 1px 4px blue;
}
.listStyle {
        padding: 5px;
        border-color: red;
        border-width: 2px;
}
```

Note - This is a css file so you have to write normal css As above.

Now you can import this file in any component where you want to use these css.

In place of class you have to use className , Since class is a reserved keyword in React Js. like below -

```
Import './Mystyle.css';
function UserDataCompWithExternalCss() {
    let userData = [{
            "id": 1,
            "title": "sunt aut",
        },{
            "id": 2,
            "title": "qui est esse",
        }];

    return (
        <div>
            <h1>Using External Css</h1>
            {
                userData.map(items => (
                    <ul className="ulStyle">
                        <li className="listStyle">Id : {items.id}</li>
                        <li className="listStyle">{items.title}</li>
                    </ul>
                ))
            }
        </div>
    )
};
export default UserDataCompWithExternalCss;
```

React Js Passing Data from Parent to Child Components Using props -

Props is a predefined keyword used in react js to pass properties from parent component to child components. These properties contain data like - numbers, strings, arrays, objects etc.
When you create a component in React, you can pass data to it by using props. These props are essentially parameters that you can define when you use the component. The data passed through props is read-only within the receiving component; the receiving component cannot modify the props it receives directly.

App.js

```
import LaptopData from './LaptopData';
import MobileData from './MobileData';


function App() {
  let laptop = {
      name: "Mac book",
      price: 10000,
      inStock : "yes"
    };
    let mobile = {
      name : "One Plus",
      price : 30000,
      inStock : "yes"
    }

  return (
    <div className="container-fluid">
      <LaptopData laptopList={laptop}/>
      <hr/>
      <MobileData mobileList = {mobile}/>
    </div>
  );
}
export default App;
```

LaptopData.js

```
import React from "react";
class LaptopData extends React.Component {
    constructor(props) {
        super();
        console.log(props.laptopList);
    }
    render() {
        return (
            <div>
                <h1>Laptops</h1>
                <h2>Name :{this.props.laptopList.name}</h2>
                <h2>Price Rs -{this.props.laptopList.price}</h2>
                <h2>In Stock - {this.props.laptopList.inStock}</h2>
            </div>
        )
    }
}

export default LaptopData;
```

mobileData.js

```
import React from "react";
class MobileData extends React.Component {
    constructor(props) {
        super();
    }
    render() {
        return (
            <div>
            <h1>Mobiles</h1>
            <h2>Name :{this.props.mobileList.name}</h2>
            <h2>Price Rs -{this.props.mobileList.price}</h2>
            <h2>In Stock - {this.props.mobileList.inStock}</h2>
         </div>
```

```
        )
    }
}
export default MobileData;
```

Note : - we have used laptopList and mobileList as properties, you can give any name as you wish.

React Conditional Rendering -

App.js

```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../node_modules/bootstrap/dist/js/bootstrap.min';
import allUserListsJsonData from '../src/mockData/userData.json';
import ConditionalRendering from './components/ConditionalRendering';
function App() {

  return (
    <div className="container-fluid">
        <ConditionalRendering userData={allUserListsJsonData} />
    </div>
  );
}

export default App;
```

ConditionalRendering.js

```
import React from "react";
```

```
class ConditionalRendering extends React.Component {
    constructor(props) {
        super();
    }

    render() {
        return (
            <div>
                {
                    this.props.userData.map(i => (
                        (i.id % 2 === 0) ? (
                    <ul>
                        <li style={{ background: "green" }}>{i.id}</li>
                        <li style={{ background: "green" }}>{i.title}</li>
                    </ul>
                    ) : (
                    <ul>
                       <li style={{ background: "yellow" }}>{i.id}</li>
                       <li style={{ background: "yellow" }}>{i.title}</li>
                    </ul>
                    )
                    ))
                }
            </div>
        )
    }
}
export default ConditionalRendering;
```

Note - we are using javascript ternary operator

Syntax -

(condition) ? true body : false body;

If the condition satisfies it will go inside the true body else false body.

Using .json file -

You can save your json data inside .json file and import it as below

App.js

```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../node_modules/bootstrap/dist/js/bootstrap.min';
import allUserListsJsonData from '../src/mockData/userData.json';
import ConditionalRendering from './components/ConditionalRendering';
function App() {

  return (
    <div className="container-fluid">
        <ConditionalRendering userData={allUserListsJsonData} />
    </div>
  );
}

export default App;
```

React Js Class component or stateful components -

Create a Class Component -

When creating a React component, the component's name must start with an uppercase letter.

The component has to 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.

```
import React from "react"; // react is a library

class SimpleClassComponent extends React.Component {
    constructor() {
        super();
    }
    render() {
        return (
            <div>
                <h1>Simple Class Component</h1>
            </div>
        )
    }
}

export default SimpleClassComponent;
```

Using CSS inside class components -

There is no change of using css(inline, internal, external) in functional components or class components. Please follow the same approach.

```
import React from "react"; // react is a library
class SimpleClassComponent extends React.Component {
    constructor() {
        super();
         this.headingStyle = {
           fontSize : "24px",
           Color : "blue"
         }
     }
```

```
    render() {
        return (
            <div style={{backgroundColor : "black", padding: "4px"}}>
                <h1 style={this.headingStyle}>Today</h1>
            </div>
        )
    }
}

export default SimpleClassComponent;
```

Handling state in React class component which makes it stateful component.

If you want to do any action (add, edit, update, delete etc), React state gives you the ability to do so using state which is a predefined keyword and setState()which is a pre-defined method.

```
import React from "react"; // react is a library

class SimpleClassComponent extends React.Component {
    constructor() {
        this.name = "Sam"
        this.state = {
            count : 0
        }
    }
    Increment() {
        this.setState({
            count : this.state.count + 1
        })
    }
    render() {
        return (
```

```
            <div>
                <h1>My Name is {this.name}</h1>
                <p>{this.state.count}</p>
                <button onClick={() => this.Increment()}>+</button>
            </div>
        )
    }
}

export default SimpleClassComponent;
```

Note -
React.Component is pre defined. This is a class which is imported from react library by adding below code;

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

state is a predefined keyword to give initial state or initialize your class component.

setState is a predefined react method which is used to update the state
of your component from the previous state to the next state.

React Events -

Just like HTML DOM events, React can perform actions based on user events.
React has the same events as HTML: click, change, mouseover etc.

Adding Events
React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={showMessage}

Using Arrow function -

OnClick={() => showMessage()}

Onchange Event -

```
import React from "react";

class ReactOnChangeEvent extends React.Component {
    constructor(props) {
        super();
        this.state = {
            name: "",
            email: "",
            selectedState: ""
        }
        this.IndianStates = ["Uttar Pradsh", "Maharastra", "Mp",
"Rajasthan"]
    }

    handleName(e) {
        this.setState({
            name: e.target.value
        })
    }
    handleEmail(e) {
        this.setState({
            email: e.target.value
        })
    }
```

```
    handleStates(e) {
        console.log(e.target.value);
        this.setState({
            selectedState: e.target.value
        })

    }
    render() {
        return (
            <div className="row" style={{ marginTop: "10%" }}>
                <div className="col">
                    <input
                        type="text"
                        className="form-control"
                        placeholder="Name"
                        aria-label="Name"
                        onChange={(e) => this.handleName(e)}
                    />
                </div>
                <div className="col">
                    <input
                        type="text"
                        className="form-control"
                        placeholder="Email"
                        aria-label="Email"
                        onChange={(e) => this.handleEmail(e)}
                    />
                </div>
                <div className="col-md-4">
                    <label for="inputState"
classNameName="form-label">State</label>
                    <select
                        id="inputState"
                        className="form-select"
                        onChange={(e) => this.handleStates(e)}>
                        {
                            this.IndianStates.map(val => (
                                <option value={val}>{val}</option>
                            ))
                        }
```

```
                    </select>
                </div>

                <div>
                    <h1>User Input Data is </h1>
                    <p>{this.state.name}</p>
                    <p>{this.state.email}</p>
                    <p>{this.state.selectedState}</p>
                </div>
            </div>
        )
    }
}
export default ReactOnChangeEvent;
```

onClick Event -

```
import React from "react";
class NameFormClick extends React.Component {
    constructor(props) {
        super();
        this.state = {
            name: '',
            email: '',
            data: []
        };
    }
    commonHandle() {
        let userName = document.getElementById("name").value;
        let userEmail = document.getElementById("email").value;
        this.setState({
            name: userName,
            email: userEmail
        });
    }
    handleClick() {
        this.setState({
            data: [this.state.name, this.state.email]
```

```
        })

    }

    render() {
        return (
            <>
                <div>
                    <div>
                        <label>  Name:</label>
                        <input type="text" value={this.state.name}
id="name" placeholder="Name.." onChange={() => this.commonHandle()} />
                    </div>

                    <div>
                        <label>Email</label>
                        <input type="text" value={this.state.email}
id="email" placeholder="email.." onChange={() => this.commonHandle()} />
                        <br />
                    </div>


                    <button type="button" onClick={() =>
this.handleClick()}>SignUp</button>
                </div>
                <div>
                    {
                        this.state.data.map(i => <p>{i}</p>)
                    }
                </div>
            </>
        );
    }
}

export default NameFormClick;
```

React Forms -

```
import React from "react";
class ReactFormOnSubmit extends React.Component {
    constructor(props) {
        super();
        this.state = {
            FName : '',
            LName : '',
            values : []
        }
    }
    handleChange() {
        let userFName = document.getElementById("firstName").value;
        let userLName = document.getElementById("lastName").value;
        this.setState({
            FName : userFName,
            LName : userLName
        })
    }
    handleSubmit(e) {
        e.preventDefault();
        this.setState({
            values : [this.state.FName, this.state.LName]
        })
    }
    render() {
        console.log(this.state.values);
        return (
            <>
            <form onSubmit={(e) => this.handleSubmit(e)}>
                <div className="row g-3">
                <div className="col">
                    <input
                        type="text"
                        className="form-control"
```

```
                        placeholder="First name"
                        id="firstName"
                        onChange={() => this.handleChange()} />
                </div>
                <div className="col">
                    <input
                        type="text"
                        className="form-control"
                        placeholder="Last name"
                        id="lastName"
                        onChange={() => this.handleChange()} />
                </div>
                <div className="col-12">
                    <input type="submit" className="btn btn-primary"
value="Submit" />
                </div>
            </div>
            </form>
            <div>
                {
                    this.state.values.map( i => <p>{i}</p>)
                }
            </div>
            </>
        )
    }
}
export default ReactFormOnSubmit;
```

React Validation -

Form validation in React allows an error message to be displayed if the user has not correctly filled out the form with the expected type of input.

Validation using javascript -

HandlingSignupForm.js

```
import React from "react";

class HandlingSignupForm extends React.Component {
    constructor() {
        super();
        this.state = {
            name: '',
            email: '', // it shoud match email regex
            password: '', // check for length of paassword min 10 char
            contactNo: '', // regex for only numbrs with exact 10 length.
            states: '', // user has selected any state
            userData: []
        }
    }
    handleUserName = (e) => {
        const userName = e.target.value;
        const userNameRegex = /^[a-z A-Z]*$/;
        if (userNameRegex.test(userName)) {
            document.getElementById("user-name").style.border = "1px solid
#777";
            this.setState({
                name: userName
            })
        } else {
            document.getElementById("user-name").style.border = "1px solid
red";
            this.setState({
                name: userName
            })
        }
    }
    handleEmail = (e) => {
        const userMail = e.target.value;
        const emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
        if (emailRegex.test(userMail)) {
            document.getElementById("user-email").style.border = "1px
solid #777";
```

```
            this.setState({
                email: userMail
            })
        } else {
            document.getElementById("user-email").style.border = "1px
solid red";
            this.setState({
                email: userMail
            })
        }
    }
    handleState = (e) => {
        const stateName = e.target.value;
        if (stateName !== 'Select') {
            document.getElementById("user-state").style.border = "1px
solid #777";
            this.setState({
                states: stateName
            })
        } else {
            document.getElementById("user-state").style.border = "1px
solid red";
            this.setState({
                states: stateName
            })
        }

    }
    signup = () => {
        console.log(this.state.states)
        if (this.state.name !== '' && this.state.email !== '' &&
this.state.states !== '') {
            this.setState({
                userData: [this.state.name, this.state.email,
this.state.states]
            })
        } else {
            this.setState({
                userData: ["Please Enter User Name and Password.", "Please
Select State."]
```

```
            })
        }

    }
    render() {
        return (
            <form className="container">
                <div className="row">
                    <div className="col-sm-6">
                        <label>User Name</label>
                        <input type="text"
                            id="user-name"
                            className="form-control"
                            value={this.state.name}
                            onChange={(e) => this.handleUserName(e)} />
                    </div>
                    <div className="col-sm-6">
                        <label>Email</label>
                        <input type="text"
                            id="user-email"
                            className="form-control"
                            value={this.state.email}
                            onChange={(e) => this.handleEmail(e)} />
                    </div>
                </div>
                <div className="row">
                    <div className="col-sm-6">
                        <label>password</label>
                        <input type="text"
                            className="form-control"
                            value="" />
                    </div>
                </div>
                <div className="row">
                    <div className="col-sm-6">
                        <label>Contact Number</label>
                        <input type="text"
                            className="form-control"
                            value="" />
                    </div>
```

```
                </div>
                <div className="row">
                    <div className="col-sm-6">
                        <label>State</label>
                        <select type="text"
                            className="form-select"
                            id="user-state"
                            onChange={(e) => this.handleState(e)}>
                            <option>Select</option>
                            <option>MP</option>
                            <option>UP</option>
                        </select>
                    </div>
                </div>
                <button type="button"
                    className="btn btn-success my-2"
                    onClick={() => this.signup()}>Signup</button>
                <h1>User Data</h1>
                <ul>
                    {
                        this.state.userData.map(i => <li>{i}</li>)
                    }
                </ul>
            </form>
        )
    }
}
export default HandlingSignupForm;
```

React validation using jquery -

Install jquery - npm i jquery

```
import React from "react";
import './validation.css';
import $ from "jquery";
```

```
class FormValidationJquery extends React.Component {
    constructor(props) {
        super();
        this.state = {
            email : '',
            password: '',
            myState: '',
            data : []
        }
    }
    handleEmail(e) {
        let userEmail = e.target.value;
        let emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
        if( userEmail !== '' && emailRegex.test(userEmail) === true) {
            this.setState({
                email : userEmail
            });
           $("#emailError").addClass("emailSuccess");
           $("#btnSubmit").prop('disabled', false);
        } else {
            this.setState({
                email : ''
            });
            $("#emailError").removeClass("emailSuccess");
            $("#btnSubmit").prop('disabled', true);
        }
    }
    handlePassword(e) {
        let userPassword = e.target.value;
        if(userPassword !== '' && userPassword.length > 5) {
            this.setState({
                password : userPassword
            });
     $("#passwordError").addClass("passwordSuccess"); // hiding error span
     $("#btnSubmit").prop('disabled', false); // enabling sign In button

        } else {
            this.setState({
                password : ''
            });
```

```
 $("#passwordError").removeClass("passwordSuccess"); // showing error span
 $("#btnSubmit").prop('disabled', true); // disabling sign In button
        }
    }
    handleStates(e) {
        let userState = e.target.value;
        if(userState === 'Choose') {
            this.setState({
                myState : ''
            })
    $("#stateError").removeClass("stateSuccess"); // showing error span
    $("#btnSubmit").prop('disabled', true); // disabling sing In button
        } else {
            this.setState({
                myState : userState
            });
     $("#stateError").addClass("stateSuccess"); // showing error span
     $("#btnSubmit").prop('disabled', false); // disabling sign In button
        }
    }

    handleSubmit(e)  {
        e.preventDefault();
    }
    render() {
        return (
            <>
                <form className="row g-3" onSubmit={(e) =>
this.handleSubmit(e)}>
                    <div className="col-md-6">
                        <label for="inputEmail4"
className="form-label">Email</label>
                        <input type="email" className="form-control"
id="inputEmail4" onChange={(e) => this.handleEmail(e)}/>
                        <span id="emailError" className="errorField
emailSuccess">Please Enter a valid Email</span>
                    </div>
                    <div className="col-md-6">
                        <label for="inputPassword4"
className="form-label">Password</label>
```

```
                        <input type="password" className="form-control"
id="inputPassword4" onChange={(e) => this.handlePassword(e)}/>
                        <span id="passwordError" className="errorField
passwordSuccess">Please Enter a password with Minium 6 Character.</span>
                    </div>
                    <div className="col-md-4">
                        <label for="inputState"
className="form-label">State</label>
                        <select id="inputState" className="form-select"
onChange={(e) => this.handleStates(e)}>
                            <option selected>Choose</option>
                            <option>MP</option>
                            <option>Maharastra</option>
                        </select>
                        <span id="stateError" className="errorField
stateError">Please Select a State.</span>
                    </div>
                    <div className="col-12">
                        <button type="submit" className="btn btn-primary"
id="btnSubmit">Sign in</button>
                    </div>
                </form>
                <div>
                    <h3>{this.state.email}</h3>
                </div>
            </>
        )
    }
}

export default FormValidationJquery;
```

Validation.css -

```
.errorField {
    color: red;
}
```

```
.emailSuccess, .passwordSuccess, .stateSuccess {
    display: none;
}
```

React Routing -

Add React Router -
To add React Router in your application, run this in the terminal from the root directory of the application:

npm i react-router-dom

Setting the path to * will act as a catch-all for any undefined URLs. This is great for a 404 error page.

App.js

```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../node_modules/bootstrap/dist/js/bootstrap.min';

import React from 'react';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from './components/pgs/Home';
import Blogs from './components/pgs/Blog';
import Contact from './components/pgs/Contact';
import Navigation from './components/pgs/Navigation';
import NoPage from './components/pgs/PageNotFound';

function App() {
  return (
    <div className="container-fluid">
      <BrowserRouter>
        <Routes>
```

```
          <Route path="/" element={<Navigation />}>
            <Route index element={<Home />} />
            <Route path="blogs" element={<Blogs />} />
            <Route path="contact" element={<Contact />} />
            <Route path="*" element={<NoPage />} />
          </Route>
        </Routes>
      </BrowserRouter>
    </div>
  );
}
export default App;
```

Navigation.js

The `Navigation` component has `<Outlet>` and `<Link>` elements.

The `<Outlet>` renders the current route selected.

`<Link>` is used to set the URL and keep track of browsing history.

```
import { Outlet, Link } from "react-router-dom";

const Navigation = () => {
  return (
    <>
      <nav className="nav">
        <Link to="/" className="nav-link active">Home</Link>

        <Link to="/blogs" className="nav-link">Blogs</Link>

        <Link to="/contact" className="nav-link">Contact</Link>
      </nav>
      <Outlet />
    </>
  )
};
export default Navigation;
```

Home.js

```
const Home = () => {
    return <h1>Home</h1>;
  };

  export default Home;
```

Blogs.js

```
const Blogs = () => {
    return <h1>Blog Articles</h1>;
  };
  export default Blogs;
```

Contact.js

```
const Contact = () => {
    return <h1>Contact Me</h1>;
  };
  export default Contact;
```

PageNotFound.js

```
const PageNotFound = () => {
    return <h1>404</h1>;
  };

export default PageNotFound;
```

- React Hooks -

Like class component state, you can also use the state feature in functional components using useState method.

1. useState() -

The React useState Hook allows us to track state in a function component.

By using useState hook, We can use state inside functional components known as stateless components and make them as stateful components like class components.

React components made using useState are known as controlled components.

Controlled Components: A controlled component is a form input component whose value is controlled by React. It means that the value of the input element is stored in the component's state, and any changes to the input value are handled through event handlers provided by React. The component re-renders whenever the state changes, and the input value is always kept in sync with the component's state. In a controlled component, you typically provide an `onChange` event handler that updates the state when the value changes.

```
import { useState } from "react";

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

    return (
        <div>
            <h1>Count is - {count}</h1>
            <button onClick={() => setCount(count + 1)}>Increment</button>
        </div>
    )
```

```
}
export default UseStateHook;
```

React Form using useState Hook -

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

function SimpleForm(props) {
    const [firstName, setFirstName] = useState('');
    const [lastName, setLastName] = useState('');
    const [age, setAge] = useState(null);
    const [data, setData] = useState([]);
    // show data after submit
    const handleSubmit = (e) => {
        e.preventDefault();
        setData([firstName, lastName, age]);
    }
    return (
        <>
            <form onSubmit={(e) => handleSubmit(e)}>
                <label>
                    First Name:
                    <input
                        type="text"
                        value={firstName}
                        onChange={(e) => setFirstName(e.target.value)}
                    />
                </label>
                <br />
                <label>
                    Last Name:
                    <input
                        type="text"
                        value={lastName}
                        onChange={(e) => setLastName(e.target.value)}
                    />
```

```
                </label>
                <br />
                <label>
                    Age:
                    <input
                        type="number"
                        value={age}
                        onChange={(e) => setAge(e.target.value)}
                    />
                </label>
                <br />
                <input type="submit" value="Submit" />

            </form>
            <div>
                <h1>Form Data</h1>
                {
                    data.map(val => <h6>{val}</h6>)
                }
            </div>
        </>
    );
}
export default SimpleForm;
```

React useEffect -

The `useEffect` Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

`useEffect` accepts two arguments. The second argument is an array which is optional. **[] empty array tells that useEffect has no any other dependencies and useEffect will be called or executed only once.**

```
useEffect(() => {

  }, []);
```

Calling API and rendering data -

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function ApiCalls() {
    const [sampleObj, setSampleObj] = useState([]);
    useEffect(() => {
        axios.get("https://jsonplaceholder.typicode.com/posts")
            .then((res) => {
                let finaldata = res.data ? res.data : [];
                setSampleObj(finaldata);
            });
    }, []);

    return (
        <div className="App">
            <h1>Hello CodeSandbox</h1>
            <ul>{sampleObj && sampleObj.map((item) =>
              <li>{item.title}</li>)}
            </ul>
        </div>
    );
}
export default ApiCalls;
```

Call API using async await -

```
useEffect(() => {
  const loadPost = async () => {
      // Till the data is fetch using API
```

```
      // Await make wait until that
      const response = await axios.get(
      "https://jsonplaceholder.typicode.com/posts/");
      setSampleObj(response.data);
  }
  // Call the function
  loadPost();
}, []);
```

React useRef -

Uncontrolled Components: On the other hand, uncontrolled components are form inputs whose values are handled by the DOM rather than React. In this approach, you rely on the DOM to manage the state of the form input, and you access its value using a reference or other DOM-related methods. With uncontrolled components, you typically don't need to write explicit event handlers to update the state because React doesn't control the input value directly.

React components made using useRef are known as uncontrolled components.

```
import React, { useRef, useState } from "react";

function Ref() {
    const inputRef = useRef(null);
    const [data, setData] = useState("");

    return (
        <div className="App">
            <h1>useRef example</h1>
            <input
                ref={inputRef}
                type="text"
                onChange={() => setData(inputRef.current.value)}
            />
            <h3>{data}</h3>
```

```
        </div>
    );
}
export default Ref;
```

Difference between controlled & uncontrolled components -

Both controlled and uncontrolled components have their use cases.
Controlled components provide more control and allow you to perform validation, apply formatting, or trigger other actions based on the input value changes.

Uncontrolled components can be useful when you have simple form inputs that don't require much manipulation or when you need to integrate with third-party libraries that expect direct DOM access.

React Memo and Pure components

Using memo will cause React to skip rendering a component if its props have not changed.

React.memo is the functional component equivalent of React.PureComponent. It is a higher-order component. If React.memo wraps a component, it memorizes the rendered output and skips subsequent renders if state, props, or context have not changed.React.memo makes a shallow comparison of props.

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

// A simple functional component
```

```jsx
const MyComponent = (props) => {
  console.log('Rendering MyComponent', props);
  return <div>Hello, {props.name}!</div>;
};
// Wrapping the component with `memo`
const MemoizedComponent = memo(MyComponent);
// The parent component
const App = () => {
  const [count, setCount] = React.useState(0);
  const handleClick = () => {
    setCount(count + 1);
  };
  return (
    <div>
      <button onClick={handleClick}>Increment Count</button>
      <MemoizedComponent name="John" />
    </div>
  );
};
export default App;
```

In this example, the `MyComponent` is wrapped with the `memo` function to create the `MemoizedComponent`. The parent component, `App`, has a button that increments the `count` state. However, since the `MyComponent` receives only the `name` prop and doesn't depend on the `count`, it will not re-render when the count is updated.

Pure components -

Pure components are similar to React memos. **The difference is pure components are used with class components but memo is used with functional components.**

```
class App extends React.PureComponent{
  constructor(props){
     super(props);
     this.state = { };
 }
 render(){
  return <div className = "app"> Hello World! </div>
 }
}
```

React Context API -

Context provides a way to pass data or state through the component tree without having to pass props down manually through each nested component. It is designed to share data that can be considered as global data for a tree of React components, such as the user authentication or applying theme.

Context API uses Context.Provider and Context.Consumer Components pass down the data but it is very cumbersome and difficult to write the long functional code to use this Context API. So **useContext** hook helps to make the code more readable, less verbose and removes the need to introduce Consumer Components. The useContext hook is the new addition in React 16.8

Working with Theme using useContext -

Using external css file -

App.js

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

import '../node_modules/bootstrap/dist/css/bootstrap.min.css';

import '../node_modules/bootstrap/dist/js/bootstrap.min';

import Comp1 from './components/contextAPI/Comp1';

import Comp2 from './components/contextAPI/Comp2';

import light from './components/contextAPI/lightTheme.module.css';

import dark from './components/contextAPI/darkTheme.module.css';

export const MyContext  = createContext();

function App() {

  const [defaultTheme, setDarkTheme] = useState(light.lightTheme);

  const handleDarkTheme = () => {

    setDarkTheme(dark.darkTheme)

  }

  return (

    <div className="container-fluid">

      <button onClick={() => handleDarkTheme()}

      className='btn btn-primary'>Apply Dark Theme</button>

      <MyContext.Provider value={defaultTheme}>

      <div>

        <Comp1 />

        <Comp2 />
```

```
      </div>
    </MyContext.Provider>
    </div>
  );
}
export default App;
```

Comp1.js

```
import React, { useContext } from 'react';
import { MyContext } from '../../App';
import CComp from './CComp';
function Comp1() {
    const theme = useContext(MyContext);
    return (
        <>
            <div className={theme}>
                This is Component
                <h1>1</h1>
                <CComp/> {/* child component */}
                </div>
        </>
    );
```

```
}

export default Comp1;
```

Child component -

CComp.js

```
import React, { useContext } from 'react';
import { MyContext } from '../../App';
function CComp() {
    const theme = useContext(MyContext);
    return (
        <>
            <div className={theme}>
      This is Child <h1>C 1</h1> of component 1</div>
        </>
    );
}
export default CComp;
```

Comp2.js

```
import React, { useContext } from 'react';
import {MyContext} from '../../App';

function Comp2() {
    const theme = useContext(MyContext);
    return (
        <>
            <div className={theme}>This is Component <h1>2</h1></div>
        </>
    )
  }

export default  Comp2;
```

Css files for Theme -

lightTheme.module.css

```
.lightTheme {
    background : #ddd;
    color: #000;
    padding: 5px;
    margin: 50px 0 20px 0;
}
```

darkTheme.module.css

```
.darkTheme {
    background : #000;
    color: #fff;
    padding: 5px;
    margin: 50px 0 20px 0;

}
```

React Fragments -

We know that we make use of the render method inside a component whenever we want to render something to the screen. We may render a single element or multiple elements, though rendering multiple elements will require a 'div' tag around the content as the render method will only render a single root node inside it at a time.

Reason to use Fragments: As we saw in the above code when we are trying to render more than one root element we have to put the entire content inside the 'div' tag which is not loved by many developers. So in React 16.2 version, Fragments were introduced, and we use them instead of the extraneous 'div' tag.

Syntax -

<React.Fragment>

```
      <h2>Child-1</h2>
      <p> Child-2</p>
</React.Fragment>
```

Or

```
<>
      <h2>Child-1</h2>
      <p> Child-2</p>
</>
```

React higher order component -

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

Syntax-

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

Example -

App.js

```
import React from "react";
import ButtonComp from '../src/components/HOC/ButtonComp';

function App() {
  return (
    <div className="App">
      <h1>Hello CodeSandbox</h1>
      <h2>working with HOC</h2>
      <ButtonComp />
    </div>
  );
}
export default App;
```

ButtonComp.js

```
import React from "react";
import Comp2 from "./Comp12";

function ButtonComp(props) {
  const clickHandler = (e) => {
    console.log("Button has been clicked from Button Component.");
  }
  return <button onClick={() => clickHandler()}>Click</button>;
}

//const EnhancedComponent = higherOrderComponent(WrappedComponent);
  const EnhancedComponent =   Comp2(ButtonComp);
export default EnhancedComponent;
```

Comp2.js

```
import React from "react";
function Comp2(EnhancedComponent) {
  return function () {
    return (
      <div style={{ backgroundColor: "red", width: "400px" }}>
        This is from Higher Order Components.
        {/* <props.cmp /> */}
        <EnhancedComponent />
      </div>
    );
  };
}
export default Comp2;
```

React Lifecycle methods

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**.

Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the **constructor** of a Component Class.

Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.

Updating: Updating is the stage when the state of a component is updated and the application is repainted.

Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

Sequence -

- React Lifecycle Methods (Used in Class Components)
  - **Mounting**
    - constructor()
    - getDerivedStateFromProps()
    - render()
    - componentDidMount()
  - **Updating**
    - getDerivedStateFromProps()
    - shouldComponentUpdate()

        - render()
        - getSnapshotBeforeUpdate()
        - componentDidUpdate()
    - **Unmounting**
        - componentWillUnmount()

getDerivedStateFromProps -

This method is called before the rendering or before any updation of the component. This method is majorly used to update the state, before the rendering of the component, which depends upon the props received by the component. This method is called on every rerendering of the component.

Syntax

```
static getDerivedStateFromProps(props, state){
}
```

componentDidMount -

componentDidMount is the final step of the mounting process. Using the componentDidMount() method, we can execute the React code when the component has already been placed in the DOM (Document Object Model). It is used for handling all network requests like API calls.

Syntax -

```
componentDidMount() {
}
```

```
class ABC extends React.Component {
  constructor(props) {
```

```
    super(props);
  }
  componentDidMount() {
    //API Call
  }
  render() {
    Return(
      //JSX
    )
  }
}
```

shouldComponentUpdate -

The shouldComponentUpdate is a lifecycle method in React. This method makes the component re-render only when there is a change in state or props of a component and that change will affect the output.

The shouldComponentUpdate() is invoked before rendering an already mounted component when new props or states are being received.

Syntax:

```
shouldComponentUpdate(nextProps, nextState){
}
```

getSnapshotBeforeUpdate -

The getSnapshotBeforeUpdate() method is invoked just before the DOM is being rendered. It is used to store the previous values of the state after the DOM is updated.

componentDidUpdate -

The componentDidUpdate() method allows us to execute the React code when the component is updated. All the network requests that are to be made when the props passed to the component changes are coded here.

The componentDidUpdate()is called after componentDidMount() and can be useful to perform some action when the state of the component changes.

Syntax -

componentDidMount() {

}

componentWillUnmount -

The componentWillUnmount() method allows us to execute the React code when the component gets destroyed or unmounted from the DOM (Document Object Model).

All the cleanups such as invalidating timers, canceling network requests, should be coded in the componentWillUnmount() method block.

Syntax -

componentWillUnmount() {

}

- React Important Question -

1. what happens when we call multiple render() in the same component , let say App.js component?
Ans - Last render JSX will show on UI.
---
2. what happens when we call multiple return() in the same component , let say App.js component?
Ans - first return JSX will be called and displayed in UI.
---
3. How can we pass values from one independent comp. to another in React ?
Ans - Using props or redux state we can do so. We can also use react context API features, like context Provider and useContext.
---
4. What is the hook in react ?
Ans - Hooks let you use state and other React features without writing a class.
You can also build your own Hooks to share reusable stateful logic between components.
Note -
As of React Version ≤ 16.7, if a certain component needs to have state and/or access to life cycle methods, it had to be built as a Class component since Functional components didn't have the concept of instances.
Features -
1. Hooks let us organize the logic inside a component into reusable isolated units.
2. Unlike patterns like render props or higher-order components, Hooks don’t introduce unnecessary nesting into your component tree.
---
5. What is HOC in js? give an example.

Ans- A higher-order component (HOC) is an advanced technique in React for reusing component logic.
A higher order component accepts a component as an argument and returns a new component —
React Redux's connect function is an example of this and returns an HOC.
Example -

```
import { addTodo } from './actionCreators'

const mapDispatchToProps = {
  addTodo
}

export default connect(
  null,
  mapDispatchToProps
)(TodoApp) // TodoApp is our component.
```

Syntax -

```
const EnhancedComponent =
higherOrderComponent(WrappedComponent);
```