**React-JS**

* **React** is a powerful JavaScript library for building dynamic and interactive user interfaces (UIs). It is developed by Facebook. React is known for its **component-based architecture**which allows you to create reusable UI elements, making complex web applications easier to manage and maintain. React is used to build single-page applications.

In this **React tutorial**, you’ll learn all the basic to advanced concepts of React such as **React components React props, React state, Functional components in React, React hooks, etc.**

## ****Which Topics We Will Learn in this React Tutorial?****

* Setup for this react development tutorial from scratch.
* Fundamental of React
* Building a first React app
* Important React packages
* Advanced concepts in React

### ****React Tutorial Prerequisites****:

* [HTML](https://www.geeksforgeeks.org/html-tutorial/)
* [CSS](https://www.geeksforgeeks.org/css-tutorial/)
* [JavaScript](https://www.geeksforgeeks.org/javascript/)

import React from 'react';

import ReactDOM from 'react-dom/client';

function Hello(props) {

return <h1>Hello GeeksforGeeks</h1>;

}

const container = document.getElementById("root");

const root = ReactDOM.createRoot(container);

root.render(<Hello />);

## ****React JS?****

React, the popular JavaScript library, offers several exciting reasons for developers to learn it.

First, React is flexible – once you learn its concepts, you can use it across various platforms to build quality user interfaces. Unlike a framework, React’s library approach allows it to evolve into a remarkable tool.

Second, React has a great developer experience, making it easier to understand and write code. Third, it benefits from Facebook’s support and resources, ensuring regular bug fixes, enhancements, and documentation updates. Additionally, React’s broader community support, excellent performance, and ease of testing make it an ideal choice for web development.

## Features of React

**1. JSX (JavaScript Syntax Extension)**:

* JSX combines HTML and JavaScript, allowing you to embed JavaScript objects within HTML elements.
* It enhances code readability and simplifies UI development.

**Example**:

const name = "GeekforGeeks";  
const ele = <h1>Welcome to {name}</h1>;

**2. Virtual DOM (Document Object Model)**:

* React uses a virtual DOM, which is an exact copy of the real DOM.
* When there are modifications in the web application, React updates the virtual DOM first and then computes the differences between the real DOM and the virtual DOM.
* This approach minimizes unnecessary re-rendering and improves performance.

**3. One-way Data Binding**:

* React follows one-way data binding, where data flows from parent components to child components.
* Child components cannot directly return data to their parent components, but they can communicate with parents to modify states based on provided inputs.

**4. Performance**:

* React’s virtual DOM and component-based architecture contribute to better performance.
* Separate components allow efficient rendering and faster execution.

**5. Extension**:

* React has a rich ecosystem and supports various extensions.
* Explore tools like **Flux**, **Redux**, and **React Native** for mobile app development and server-side rendering.

**6. Conditional Statements in JSX**:

* JSX allows writing conditional statements directly.
* Display data in the browser based on provided conditions.

**Example**:

const age = 12;  
if (age >= 10) {  
 return <p>Greater than {age}</p>;  
} else {  
 return <p>{age}</p>;  
}

**7. Components**:

* React divides web pages into reusable and immutable components.
* Component-based development simplifies code organization and maintenance.

## ****ReactJS Advantages****

* **Composable:** We can divide these codes and put them in custom components. Then we can utilize those components and integrate them into one place.
* **Declarative:** In ReactJS, the DOM is declarative. We can make interactive UIs by changing the state of the component and ReactJS takes care of updating the DOM according to it.
* **SEO Friendly:** ReactJS affects the SEO by giving you a SPA (Single Page Application) which requires Javascript to show the content on the page which can be rendered and indexed.
* **Community:** ReactJS has a huge community because of its demand each company wants to work with ReactJS. Companies like Meta, Netflix, etc built on ReactJS.
* **Easy to learn:** HTML-like syntax of JSX makes you comfortable with the codes of React, it only requires a basic knowledge of HTML, CSS, and JS fundamentals to start working with it.
* If you want to learn more refer to this article [React JS Advantages](https://www.geeksforgeeks.org/what-are-the-advantages-of-react-js/)
* **Debugging is Easy:** The debugging of ReactJS is unidirectional which means while designing any app using ReactJS the child components are nested within parent components. So, the data flow is in a single direction it gets more easier to debug errors.

## React Tutorial

## React Basic Concepts

* [Introduction](https://www.geeksforgeeks.org/reactjs-introduction/)
* [Import and Export](https://www.geeksforgeeks.org/reactjs-importing-exporting/)
* [JSX Introduction](https://www.geeksforgeeks.org/reactjs-jsx-introduction/)
* [Components](https://www.geeksforgeeks.org/reactjs-components/)
* [Conditional Rendering](https://www.geeksforgeeks.org/reactjs-conditional-rendering/)
* [PropTypes](https://www.geeksforgeeks.org/reactjs-proptypes/)
* [Prop Drilling](https://www.geeksforgeeks.org/what-is-prop-drilling-and-how-to-avoid-it/)
* [React Lists](https://www.geeksforgeeks.org/reactjs-lists/)
* [Context API](https://www.geeksforgeeks.org/explain-new-context-api-in-react/)
* [React Redux](https://www.geeksforgeeks.org/introduction-to-react-redux/)

## React Hooks

* [Hooks Introduction](https://www.geeksforgeeks.org/reactjs-hooks/)
* [useState Hook](https://www.geeksforgeeks.org/reactjs-usestate-hook/)
* [useEffect Hook](https://www.geeksforgeeks.org/reactjs-useeffect-hook/)
* [useRef Hook](https://www.geeksforgeeks.org/react-js-useref-hook/)
* [useMemo Hook](https://www.geeksforgeeks.org/react-js-usememo-hook/)
* [useContext Hook](https://www.geeksforgeeks.org/reactjs-usecontext-hook/)

## ****React DOM Events****

* [React Events Introduction](https://www.geeksforgeeks.org/react-js-events/)
* [onclickcapture Event](https://www.geeksforgeeks.org/what-is-onclickcapture-event-in-reactjs/)
* [onMouseDown Event](https://www.geeksforgeeks.org/react-onmousedown-event/)
* [onDoubleClick Event](https://www.geeksforgeeks.org/react-ondoubleclick-event/)
* [onSubmit Event](https://www.geeksforgeeks.org/react-onsubmit-event/)
* [onScroll Event](https://www.geeksforgeeks.org/react-onscroll-event/)
* [onBlur Event](https://www.geeksforgeeks.org/react-onblur-event/)

## ****Lifecycle of Components****

* [Introduction to lifecycle of components](https://www.geeksforgeeks.org/reactjs-lifecycle-components/)
* [constructor](https://www.geeksforgeeks.org/react-js-constructor-method/)
* [render](https://www.geeksforgeeks.org/react-js-render-method/)
* [componentDidMount](https://www.geeksforgeeks.org/reactjs-componentdidmount-method/)
* [componentWillUnmount](https://www.geeksforgeeks.org/reactjs-componentwillunmount-method/)
* [componentDidCatch](https://www.geeksforgeeks.org/reactjs-componentdidcatch-method/)
* [componentDidUpdate](https://www.geeksforgeeks.org/reactjs-componentdidupdate-method/)
* [shouldComponentUpdate](https://www.geeksforgeeks.org/reactjs-shouldcomponentupdate-method/)

## Important React Packages

* [Redux](https://www.geeksforgeeks.org/introduction-to-react-redux/)
* [Material UI](https://www.geeksforgeeks.org/react-material-ui/)
* [react-bootstrap](https://www.geeksforgeeks.org/react-bootstrap/)
* [Tailwind](https://www.geeksforgeeks.org/tailwind-css/)
* [Framer Motion](https://www.geeksforgeeks.org/framer-motion-introduction-and-installation/)

# React-JS Components

Components are bits of code that are freelance and reusable. They serve a similar purpose as JavaScript functions. However, work isolation and returns HTML via a render function.

A **Component** is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of pieces called components. Components make the task of building UIs much easier. You can see a UI broken down into multiple individual pieces called components and work on them independently and merge them all in a parent component which will be your final UI.

Components in React basically return a piece of JSX code which tells what should be rendered on the screen. In React, we mainly have two types of components:

* **Functional Components:** Functional components are simply JavaScript functions. We can create a functional component to React by writing a JavaScript function. These functions may or may not receive data as parameters, we will discuss this later in the tutorial. Below example shows a valid functional component in React:

## JavaScript-

## function Democomponent()

## {

## return <h1>Welcome Message!</h1>;

## }

* **Class Components:** The class components are a little more complex than the functional components. The functional components are not aware of the other components in your program whereas the class components can work with each other. We can pass data from one class component to other class components. We can use JavaScript ES6 classes to create class-based components in React.

Below example shows a valid class-based component in React:

## JavaScript-

class Democomponent extends React.Component

{

   render(){

         return <h1>Welcome Message!</h1>;

   }

}

# React Components-

# **Components in React**serve as independent and reusable code blocks for UI elements. They represent different parts of a web page and contain both **structure**and **behavior**. They are similar to**JavaScript functions**and make creating and managing complex user interfaces easier by breaking them down into smaller, reusable pieces.

## ****What are React Components?****

**React Components**are the building block of React Application. They are the reusable code blocks containing logics and and UI elements. They have the same purpose as [**JavaScript functions**](https://www.geeksforgeeks.org/functions-in-javascript/) and return [**HTML**](https://www.geeksforgeeks.org/html-introduction/). Components make the task of building UI much easier.

A UI is broken down into multiple individual pieces called components. You can work on components independently and then merge them all into a **parent component** which will be your final UI.

Components promote**efficiency** and **scalability**in web development by allowing developers to compose, combine, and customize them as needed

## ****Types of Components in React****

In React, we mainly have two types of components:

* **Functional Components**
* **Class based Components**

### ****Functional Component in React****

[**Functional components**](https://www.geeksforgeeks.org/reactjs-functional-components/)are just like JavaScript functions that accept properties and return a React element.

We can create a functional component in React by writing a JavaScript function. These functions may or may not receive data as parameters, we will discuss this later in the tutorial. The below example shows a valid functional component in React:

#### ****Syntax:****

function demoComponent() {  
 return (<h1>  
 Welcome Message!  
 </h1>);  
}

**Example:**Create a function component called welcome.

Javascript

**function** welcome() {

**return** <h1>Hello, Welcome to GeeksforGeeks!</h1>;

}

### ****Class Component in React****

The [class components](https://www.geeksforgeeks.org/reactjs-class-components/) are a little more complex than the functional components. A class component can show **inheritance**and access data of other components.

Class Component must include the line “**extends React.Component**” to pass data from one class component to another class component. We can use JavaScript ES6 classes to create class-based components in React.

#### ****Syntax:****

class Democomponent extends React.Component {  
 render() {  
 return <h1>Welcome Message!</h1>;  
 }  
}

The below example shows a valid class-based component in React:

**Example:**Create a class component called welcome**.**

Javascript

**class** Welcome **extends** Component {

render() {

**return** <h1>Hello, Welcome to GeeksforGeeks!</h1>;

}

}

The components we created in the above two examples are equivalent, and we also have stated the basic difference between a functional component and a class component. We will learn about more properties of class-based components in further tutorials.

## Functional Component vs Class Component

A functional component is best suited for cases where the component doesn’t need to interact with other components or manage complex states. Functional components are ideal for **presenting static UI elements** or composing multiple simple components together under a single parent component.

While class-based components can achieve the same result, they are generally**less efficient** compared to functional components. Therefore, it’s recommended to not use class components for general use.

## ****Rendering React Components****

**Rendering Components**means turning your component code into the UI that users see on the screen.

React is capable of rendering user-defined components. To render a component in React we can initialize an element with a user-defined component and pass this element as the first parameter to[**ReactDOM.render()**](https://www.geeksforgeeks.org/explain-the-purpose-of-render-in-reactjs/) or directly pass the component as the first argument to the ReactDOM.render() method.

The below syntax shows how to initialize a component to an element:

**const elementName = <ComponentName />;**

In the above syntax, the ComponentName is the name of the user-defined component.   
**Note**: The name of a component should always start with a capital letter. This is done to differentiate a component tag from an [**HTML tag**](https://www.geeksforgeeks.org/html-tags-a-to-z-list/).

**Example:**This example renders a component named Welcome to the Screen**.**

javascript

*// Filename - src/index.js:*

**import** React **from** "react";

**import** ReactDOM **from** "react-dom";

*// This is a functional component*

**const** Welcome = () => {

**return** <h1>Hello World!</h1>;

};

ReactDOM.render(

<Welcome />,

document.getElementById("root")

);

**Output:**This output will be visible on the **http://localhost:3000/**on the browser window.

rendering component output

**Explanation:**

Let us see step-wise what is happening in the above example:

* We call the ReactDOM.render() as the first parameter.
* React then calls the component Welcome, which returns <h1>Hello World!</h1>; as the result.
* Then the ReactDOM efficiently updates the DOM to match with the returned element and renders that element to the DOM element with id as “root”.

## Props

* **Props(short for properties)** are a way to pass data from the parent component to the child component.
* Props are like **function arguments**, you can use them as attributes in components.

**Example:**

Javascript

**function** Message(props) {

**return** <h2>{props.text}</h2>;

}

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

root.render(<Message text="Hello, world!" />);

## Components in Components

We can call components inside another component

**Example:**

Javascript

*// Filename - src/index.js:*

**import** React **from** "react";

**import** ReactDOM **from** "react-dom";

**const** Greet = () => {

**return** <h1>Hello Geek</h1>

}

*// This is a functional component*

**const** Welcome = () => {

**return** <Greet />;

};

ReactDOM.render(

<Welcome />,

document.getElementById("root")

);

The above code will give the same output as other examples but here we have called the Greet component inside the Welcome Component.

For more information on components open [Component Set 2](https://www.geeksforgeeks.org/reactjs-components-set-2/)

## Conclusion

Components in React allow developers to divide the page UI into many small parts. These parts work independently and can be used multiple times just like functions.

This tutorial introduces you to the concept of components. We have discussed types of components and their purpose in [web development](https://www.geeksforgeeks.org/web-development/). This guide will help you understand web UI and how can you create visually appealing web UI.

# React-JS Props-

In React, components can receive information from a parent component by utilizing props (short for properties). A prop is an object accessible to all React components. It serves as a means to pass data from a parent component to a child component.

<Welcome fullName = "Ashish Bhardwaj" />

In the provided code, the prop `fullName` is passed with the value “Ashish Bhardwaj” to the `Welcome` component, and accessing this prop is demonstrated succinctly.

props = { fullName: "Ashish Bhardwaj" }

In case of functional components, we can access a prop value as shown below.

props.propName;

**Example:**Below is the basic implementation of the React Props.

//index.js

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

ReactDOM.render(

    <React.StrictMode>

        <App fullName={"Ashish Bhardwaj"} />

    </React.StrictMode>,

    document.getElementById("root")

);

Output:



**React JS Props Reference:**

| **Props Reference** | **Description** |
| --- | --- |
| [**React JS Methods as Props**](https://www.geeksforgeeks.org/reactjs-methods-as-props/) | We know that everything in **ReactJS** is a **component** and to pass in data to these components, **props** are used. |
| [**React JS PropTypes**](https://www.geeksforgeeks.org/reactjs-proptypes/) | PropTypes help in specifying the expected data types of the props (properties) that a component should receive during its use. |
| [**React JS Props – Set 1**](https://www.geeksforgeeks.org/reactjs-props-set-1/) | React allows us to pass information to a Component using something called **props** |
| [**React JS Props – Set 2**](https://www.geeksforgeeks.org/reactjs-props-set-2/) | We will learn about some more advanced concepts about props in this article |
| [**Unidirectional Data Flow**](https://www.geeksforgeeks.org/unidirectional-data-flow/) | Unidirectional data flow is a technique that is mainly found in functional reactive programming. |
| [**React JS State in React**](https://www.geeksforgeeks.org/reactjs-state-react/) | The state in React is an instance of the React Component Class that can be defined as an object of a set of **observable** properties that control the behavior of the component. |
| [**React JS State vs props**](https://www.geeksforgeeks.org/reactjs-state-vs-props/) | States and props are two of the most important concepts of React and everything in React is based upon them. |
| [**React JS Implementing State & Lifecycle**](https://www.geeksforgeeks.org/reactjs-implementing-state-lifecycle/) | lifecycle methods are used to control the components at different stages from initialization till unmounting. |

# React JS Hooks-

React hooks are functions that enable functional components to use state and lifecycle features that were previously only available in class components.

**Example:**Below is the basic representation of the React JS Hooks useState.

# javaScrpit-

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

# import './App.css'

# const App = () => {

# const [num, setNum] = useState(0);

# const handleClick = () => {

# setNum(num + 1);

# };

# return (

# <div className="App">

# <h2> {num}</h2>

# <button onClick={handleClick}>

# Add one

# </button>

# </div>

# );

# };

# export default App;

CSS-

/\* Write CSS Here \*/

.App {

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

}

body {

background-color: antiquewhite;

}

.App>h2 {

text-align: center;

}

.App>button {

width: 8rem;

font-size: larger;

padding: 2vmax auto;

height: 1.8rem;

color: white;

background-color: rgb(34, 34, 33);

border-radius: 10px;

}

button:hover {

background-color: rgb(80, 80, 78);

}

# React-JS State-

The State is a way to store and manage the information or data while creating a React Application. The state is a **JavaScript object** that contains the real-time data or information on the webpage.

**Table of Content**

* [What is React State?](https://www.geeksforgeeks.org/reactjs-state/#what-is-react-state)
* [Creating State Object](https://www.geeksforgeeks.org/reactjs-state/#creating-state-object)
* [Conventions of Using State in React](https://www.geeksforgeeks.org/reactjs-state/#conventions-of-using-state-in-react)
* [Updating State in React](https://www.geeksforgeeks.org/reactjs-state/#updating-state-in-react)

## ****What is a React State?****

The state in React is an instance of the [**React Component Class**](https://www.geeksforgeeks.org/reactjs-class-components/) that can be defined as an object of a set of **observable** properties that control the behavior of the [**component**](https://www.geeksforgeeks.org/reactjs-components/).

In other words, the State of a component is an object that holds some information that may change over the lifetime of the component.

**Creating State Object**

Creating a state is essential to building dynamic and interactive components.

We can create a state object within the **constructor**of the class component.

JavaScript

**import** **React** **from** 'react';

**class** **MyComponent** extends React.Component {

constructor(props) {

super(props);

this.state = {

brand: 'Ford', // Example property **in** the state

};

}

render() {

**return** (

<div>

<h1>My Car</h1>

{/\* Other component content \*/}

</div>

);

}

}

export default MyComponent;

**Conventions of Using State in React**

* The state of a component should prevail throughout its lifetime, **thus we must first have some initial state**, to do so we should define the State in the **constructor** of the component’s class.
* **The state should never be updated explicitly.** React uses an observable object as the state that observes what changes are made to the state and helps the component behave accordingly.
* React provides its own method [**setState()**](https://www.geeksforgeeks.org/reactjs-setstate/). setState() method takes a single parameter and expects an object which should contain the set of values to be updated. Once the update is done the method implicitly calls the [**render()**](https://www.geeksforgeeks.org/react-js-render-method/)method to repaint the page. Hence, the correct method of updating the value of a state will be similar to the code below.
* **State updates should be independent.** The state object of a component may contain multiple attributes and React allows to use setState() function to update only a subset of those attributes as well as using multiple setState() methods to update each attribute value independently.
* The only time we are allowed to define the state explicitly is in the constructor to provide the initial state.

**Updating State in React**

In React, a State object can be updated using **setState() method**.

React may update multiple setState() updates in a single go. Thus using the value of the current state may not always generate the desired result.

For example, let us take a case where we must keep a count (Likes of a Post). **Many developers may miswrite the code as below.**

this.setState({counter: this.state.count + this.props.diff});

**Correct Method to Update State**

In the below code, we are using the ES6 thick arrow function format to take the previous state and props of the component as parameters and are updating the counter. The same can be written using the default functional way as follows.

this.setState((prevState, props) => ({  
 counter: prevState.count + props.diff  
}));

Now let us see an example where we will implement state in React and use the state to create a counter

**Example**

This example demonstrates the use of React JS state creating a simple counter application.

JavaScript

// Filename - index.js

**import** **React** **from** "react";

**import** **ReactDOM** **from** "react-dom/client";

**class** **App** extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0,

};

}

increment = () => {

this.setState((prevState) => ({

count: prevState.count + 1,

}));

};

decrement = () => {

this.setState((prevState) => ({

count: prevState.count - 1,

}));

};

render() {

**return** (

<div>

<h1>

The current count **is** :{" "}

{this.state.count}

</h1>

<button onClick={this.increment}>

Increase

</button>

<button onClick={this.decrement}>

Decrease

</button>

</div>

);

}

}

const root = ReactDOM.createRoot(

document.getElementById("root")

);

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

**Output:**

