**React Hooks**

**In React, Hooks** are **reusable functions**that **provide access to state**in **React Applications. Hooks** were introduced in the **16.8 version of React**. Hooks give access to states for functional components while creating a React application. It allows you to use state and other React features without writing a class.

**Table of Content**

* [What are React Hooks?](https://www.geeksforgeeks.org/reactjs-hooks/#what-are-react-hooks)
* [When to use React Hooks](https://www.geeksforgeeks.org/reactjs-hooks/#when-to-use-react-hooks)
* [Types of React Hooks](https://www.geeksforgeeks.org/reactjs-hooks/#types-of-react-hooks)
* [Benefits of using Hooks](https://www.geeksforgeeks.org/reactjs-hooks/#benefits-of-using-hooks)
* [Why the need for ReactJs Hooks?](https://www.geeksforgeeks.org/reactjs-hooks/#why-the-need-for-reactjs-hooks)
* [Rules for using Hooks](https://www.geeksforgeeks.org/reactjs-hooks/#rules-for-using-hooks)
* [Using Hooks in React](https://www.geeksforgeeks.org/reactjs-hooks/#using-hooks-in-react)
* [Difference Between Hooks and Class Components](https://www.geeksforgeeks.org/reactjs-hooks/#difference-between-react-hooks-and-class-components)
* [Important things to remember while using hooks](https://www.geeksforgeeks.org/reactjs-hooks/#important-things-to-remember-while-using-hooks)

**What are React Hooks?**

**React Hooks**provide [**functional components**](https://www.geeksforgeeks.org/reactjs-functional-components/)with the **ability to use states and manage side effects**. They were first introduced in **React 16.8**, and allow developers to hook into the state and other React features **without having to write a class**. They provide a cleaner and more concise way to handle state and side effects in React applications.

Although Hooks generally replace class components, no plans exist to remove classes from React.

***Note: Hooks cannot be used with*** [*class components*](https://www.geeksforgeeks.org/reactjs-class-components/)

**When to use React Hooks?**

If you’ve created a functional component in React and later need to add state or other React features that were previously only available in class components, you no longer have to convert it into a class component.

With React Hooks, you can add state and other features directly to a functional component. Hooks like[useState](https://www.geeksforgeeks.org/reactjs-usestate-hook/) and [useEffect](https://www.geeksforgeeks.org/reactjs-useeffect-hook/" \t "_blank)enable you to manage state, side effects, and other logic within your existing functional components, making your code more concise and avoiding the need to rewrite components as classes. React hooks have transformed the way we manage state and side effects in functional components. The [**ReactJS Course**](https://gfgcdn.com/tu/Sz5/) offers comprehensive lessons on all the hooks, providing you with the tools you need to build modern React applications effectively.

**Types of React Hooks**

The[Built-in React Hooks](https://www.geeksforgeeks.org/built-in-react-hooks/) are:

* **State Hooks**
* **Context Hooks**
* **Ref Hooks**
* **Effect Hooks**
* **Performance Hooks**
* **Resource Hooks**
* **Other Hooks**

**1. React State Hooks**

State Hooks stores and provide access to the information. To add state in Components we use:

* [**useState Hook**](https://www.geeksforgeeks.org/reactjs-usestate-hook/)**:**useState Hooks provides state variable with direct update access.
* [**useReducer Hook**](https://www.geeksforgeeks.org/reactjs-usereducer-hook/)**:**useReducer Hook provides a state variable along with the update logic in reducer function.

const [count, setCount] = useState(0)

**2. React Context Hooks**

Context hooks make it possible to access the information without being passed as a prop.

* [**useContext** **Hooks**](https://www.geeksforgeeks.org/reactjs-usecontext-hook/): shares the data as a global data with passing down props in component tree.

const context = useContext(myContext);

**3. React Ref Hooks**

Refs creates the variable containing the information not used for rendering e.g. DOM nodes.

* [**useRef**](https://www.geeksforgeeks.org/react-js-useref-hook/): Declares a reference to the DOM elements mostly a DOM Node.
* [**useImperativeHandle**](https://www.geeksforgeeks.org/react-js-useimperativehandle-additional-hook/): It is an additional hook that declares a customizable reference

const textRef = useRef(null);

**4. Effect Hooks:**

Effects connect the components and make it sync with the system. It includes changes in browser DOM, networks and other libraries.

* [**useEffect**](https://www.geeksforgeeks.org/reactjs-useeffect-hook/): useEffect Hook connects the components to external system
* [**useLayoutEffect**](https://www.geeksforgeeks.org/reactjs-uselayouteffect-hook/): used to measure the layout, fires when the screen rerenders.
* [**useInsertionEffect**](https://www.geeksforgeeks.org/react-useinsertioneffect-hook/): used to insert the CSS dynamically, fires before the changes made to DOM.

useEffect(()->{  
 // Code to be executed  
}, [dependencies] )

**5. React Performance Hooks:**

Performace hooks are a way to skip the unnecessary work and optimise the rendering preformance.

* [**useMemo**](https://www.geeksforgeeks.org/react-js-usememo-hook/): return a memoized value reducing unnecessary computations.
* [**useCallback**](https://www.geeksforgeeks.org/react-js-usecallback-hook/): returns a memoized callback that changes if the dependencies are changed.

const memoizedValue = useMemo(functionThatReturnsValue, arrayDependencies)

To prioritize rendering we can use these:

* [**useTransition**](https://www.geeksforgeeks.org/reactjs-usetransition-hook/): enables us to specify which state changes are critical or urgent and which are not.
* [**useDefferedValue**](https://www.geeksforgeeks.org/what-is-usedeferredvalue-hook-and-how-to-use-it/): allows you to postpone the rendering of a component until a specific condition is met.

**6. React Resource Hooks:**

It allows component to access the resource without being a part of their state e.g., accessing a styling or reading a promise.

* **use**: used to read the value of resources e.g., context and promises.

const data = use(dataPromise);

**7. Additional React Hooks:**

These are rarely used hooks mostly used in libraries.

* [**useDebugValue**](https://www.geeksforgeeks.org/reactjs-usedebugvalue-hook/): helps developers debug custom hooks in React Developer Tools by adding additional information and labels to those hooks.
* [**useID**](https://www.geeksforgeeks.org/reactjs-useid-hook/): generates unique IDs i.e, returns a string that is stable across both the server and the client sides.
* **useSyncExternalStore:**helps components subscribe to external stores.

Apart from these Built-in React hooks we can also[**create Custom Hooks in React**](https://www.geeksforgeeks.org/reactjs-custom-hooks/)**.**

You can see a Complete list of React Hooks in[**ReactJS Hooks Complete Reference**](https://www.geeksforgeeks.org/reactjs-hooks-complete-reference/)**.**

**Benefits of using Hooks**

Hooks can improve code reusability and make it easier to split complex components into smaller functions.

* **Simpler, cleaner code:**Functional components with hooks are often more concise and easier to understand than class components.
* **Better for complex UIs**: Hooks make it easier to manage state and side effects in components with intricate logic.
* **Improved maintainability**: Code using hooks is often easier to test and debug.

**Why the need for ReactJs Hooks?**

There are multiple reasons responsible for the introduction of the Hooks which may vary depending upon the experience of developers in developing React application. Needs for react hooks are:

* Use of ‘this’ keyword
* Reusable stateful logics
* Simplifying complex scenarios

**1. Use of ‘this’ keyword**

* Working with classes in React involves understanding [JavaScript’s ‘this’ keyword](https://www.geeksforgeeks.org/javascript-this-keyword/) intricacies, causing challenges uncommon in other languages.
* Implementing class components requires binding event handlers, adding complexity compared to the simplicity of props and state
* React developers note that classes lack efficiency and may hinder hot reloading reliability, a concern Hooks address effectively

**2. Reusable stateful logics:**

* Addressing higher-level concepts like [Higher-order components (HOC)](https://www.geeksforgeeks.org/react-js-higher-order-components/) and render props, reusing stateful logic is challenging.
* Solutions like HOC and render props can lead to an inefficient code base, complicating readability with nested components.
* Hooks offer a cleaner way to share stateful logic without altering component hierarchy, enhancing code organization and clarity.

**3. Simplifying complex scenarios:**

* In complex scenarios, life-cycle methods may scatter code, making it challenging to organize related logic in one place.
* Hooks address this issue by allowing the organization of code based on related functionality rather than life-cycle methods.

**Rules for using Hooks**

* Only functional components can use hooks
* Hooks must be imported from React
* Calling of hooks should always be done at top level of components
* Hooks should not be inside conditional statements

**Using Hooks in React**

This example demonstrate the use of [react useState hook](https://www.geeksforgeeks.org/reactjs-usestate-hook/) in the application.

JavaScript

*// Filename - index.js*

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

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

**function** App() {

**const** [click, setClick] = useState(0);

*// Using array destructuring here*

*// to assign initial value 0*

*// to click and a reference to the function*

*// that updates click to setClick*

**return** (

<div>

<p>You clicked {click} times</p>

<button onClick={() => setClick(click + 1)}>

Click me

</button>

</div>

);

}

**const** root = ReactDOM.createRoot(

document.getElementById("root")

);

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

**Output:**

We have used useState hook which lets functional components have access to states with the help of which we are able to manipulate states



**Difference Between Hooks and Class Components**

| **Feature** | **Class Components** | **React Hooks** |
| --- | --- | --- |
| **State Management** | this.state and lifecycle methods | useState and useEffect |
| **Code Structure** | Spread across methods, can be complex | Smaller, focused functions |
| **Reusability** | Difficult to reuse logic | Easy to create and reuse custom hooks |
| **Learning Curve** | Familiar to OOP developers | Requires different mindset than classes |
| **Error Boundaries** | Supported | Not currently supported |
| **Third-party Libraries** | Some libraries rely on them | May not all be compatible yet |

**Important things to remember while using hooks**

* Hooks are optional in React 16.8+, allowing partial or full project adoption without rewriting existing code.
* Hooks are backward-compatible, ensuring smooth integration with existing components and preventing breaking changes.
* React has no plans to eliminate classes; Hooks and class components can coexist.
* React projects can seamlessly blend class-based and functional components with Hooks.
* Hooks provide a direct API for key React concepts, such as props, state, context, refs, and lifecycle.

Want to be a Software Developer or a Working Professional looking to enhance your **Software Development Skills**? Then, master the concepts of Full-Stack Development. Our [**Full Stack Development - React and Node.js Course**](https://gfgcdn.com/tu/Q3I/) will help you achieve this quickly. Learn everything from **Front-End to Back-End Development** with hands-on **Projects**and real-world examples. This course enables you to build scalable, efficient, dynamic web applications that stand out. Ready to become an expert in Full-Stack? Enroll Now and Start Creating the Future!