**RACT-JS LIST AND HOOKS**

**React Lists-**

**React Hooks-**

**React List-**

**Lists**are very useful when it comes to developing the UI of any website. **React Lists**are mainly used for **displaying menus** on a website, for example, the **navbar menu**. In regular JavaScript, we can use **arrays** for creating lists. In React, rendering lists efficiently is critical for maintaining performance and ensuring a smooth user experience. React provides powerful tools to handle lists, allowing developers to create dynamic and responsive applications.

In this tutorial, we will learn about **React lists with examples**. We will cover React list fundamentals like **creating lists**, **traversing lists**, and **rendering lists** in React with examples. Let’s start by learning how to create and traverse React lists.

**Table of Content**

* [Steps to Create and Traverse React JS Lists](https://www.geeksforgeeks.org/reactjs-lists/#steps-to-create-and-traverse-react-js-lists)
* [Rendering lists inside Components](https://www.geeksforgeeks.org/reactjs-lists/#rendering-lists-inside-components)
* [Key in React List](https://www.geeksforgeeks.org/reactjs-lists/#key-in-react-list)

## Rendering Lists with the map() Function

We can create lists in React just like we do in regular [**JavaScript**](https://www.geeksforgeeks.org/introduction-to-javascript/)i.e. by storing the list in an [**array**](https://www.geeksforgeeks.org/reactjs-lists/JavaScript%20Arrays). To traverse a list we will use the [map()](https://www.geeksforgeeks.org/javascript-array-map-method/) function. To create a React list, follow these given steps:

**Step 1:**Create a list of elements in React in the form of an array and store it in a variable. We will render this list as an unordered list element in the browser.

**Step 2:**We will then traverse the list using the JavaScript map() function and update elements to be enclosed between <li> </li> elements.

**Step 3:**Finally we will wrap this new list within <ul> </ul> elements and render it to the DOM.

### ****React List Examples-js****

// Filename - index.js

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

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

**const** numbers = [1,2,3,4,5];

**const** updatedNums = numbers.map((number)=>{

**return** <li>{number}</li>;

});

ReactDOM.render(

<ul>

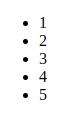
{updatedNums}

</ul>,

document.getElementById('root')

);

**Output:**The above code will render an unordered list as shown below



## ****Rendering lists inside Components****

In the above code in React, we directly rendered the list to the [**DOM**](https://www.geeksforgeeks.org/dom-document-object-model/). But usually, this is not a good practice to render lists in React. We already have talked about the uses of [**Components**](https://www.geeksforgeeks.org/reactjs-components/)and have seen that everything in React is built as individual components.

Consider the example of a Navigation Menu. It is obvious that in any website the items in a navigation menu are not hard coded. This item is fetched from the database and then displayed as a list in the browser. So from the component’s point of view, we can say that we will pass a list to a component using [**props**](https://www.geeksforgeeks.org/reactjs-props-set-1/)and then use this component to render the list to the DOM.

We can update the above code in which we have directly rendered the list to now a component that will accept an array as props and return an unordered list.

*// Filename - index.js*

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

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

*// Component that will return an*

*// unordered list*

**function** Navmenu(props)

{

**const** list = props.menuitems;

**const** updatedList = list.map((listItems)=>{

**return** <li>{listItems}</li>;

});

**return**(

<ul>{updatedList}</ul>

);

}

**const** menuItems = [1,2,3,4,5];

ReactDOM.render(

<Navmenu menuitems = {menuItems} />,

document.getElementById('root')

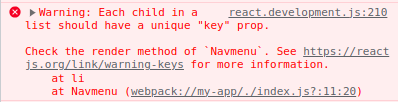
);

**Output:**

ReactJS list

*ReactJS Lists*

**The above code will give a warning in the console of the browser like:**



*ReactJS Lists Warning*

The above warning message says that each of the list items in our unordered list should have a unique key.

## Using Keys in Lists-

**Keys** are a important aspect of rendering lists in React. They help React identify which items have changed, are added, or are removed. Providing a unique key for each list item significantly improves performance and avoids potential bugs.

**Example:**This is the updated code for React List with keys:

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

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

*// Component that will return an*

*// unordered list*

**function** Navmenu(props)

{

**const** list = props.menuitems;

**const** updatedList = list.map((listItems)=>{

**return**(

<li key={listItems.toString()}>

{listItems}

</li>

);

});

**return**(

<ul>{updatedList}</ul>

);

}

**const** menuItems = [1,2,3,4,5];

ReactDOM.render(

<Navmenu menuitems = {menuItems} />,

document.getElementById('root')

);

**Output:** In the below-shown output you can see the rendered output is the same but this time without any warning in the console.

https://media.geeksforgeeks.org/wp-content/uploads/20230728092828/gfg.png

*ReactJS Lists with Keys*

Keys are used in React to identify which items in the list are changed, updated, or deleted. We will learn about keys in more detail in our [ReactJS keys article](https://www.geeksforgeeks.org/reactjs-keys/).

**Conclusion**

In summary, React lists are arrays with values. To render the elements of an array we iterate over each element and create a JSX element for each item. We use keys to label list elements, and when the changes are made to the list we don’t need to re-render the entire list.

This tutorial explains React lists and uses JavaScript codes to show examples of all different concepts. After completing this tutorial, you will be able to use lists in your React project.

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.

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;

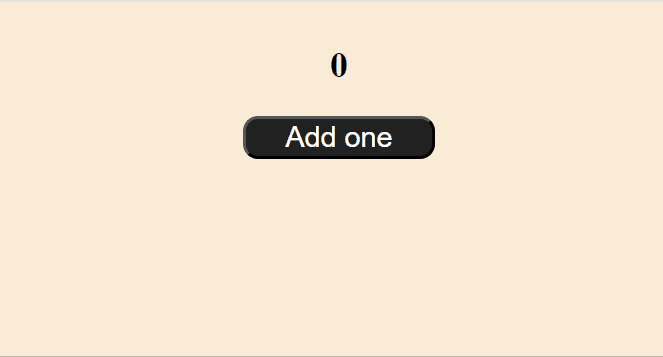
}

button:hover {

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

}

**Output:**



**eact JS Hooks Reference:**

| **React Hooks** | **Description** |
| --- | --- |
| [**Introduction to React Hooks**](https://www.geeksforgeeks.org/introduction-to-react-hooks/) | React hooks are functions that enable functional components to use state and lifecycle features that were previously only available in class components. |
| [**ReactJS Custom Hooks**](https://www.geeksforgeeks.org/reactjs-custom-hooks/) | We know that hooks like useState, useEffect are reusable components. Somtimes we make components that we have to reuse again and again in the application. |
| [**ReactJS useContext Hook**](https://www.geeksforgeeks.org/reactjs-usecontext-hook/) | Context provides a way to pass data or state through the component tree without having to pass props down manually through each nested component. |
| [**ReactJS useEffect Hook**](https://www.geeksforgeeks.org/reactjs-useeffect-hook/) | React JS useEffect hooks is a predefined hook that handles the effects of the dependency array. |
| [**ReactJS useReducer Hook**](https://www.geeksforgeeks.org/reactjs-usereducer-hook/) | The **useReducer** Hook is the better alternative to the **useState** hook and is generally more preferred over the **useState** hook when you have complex state-building logic |
| [**ReactJS useMemo Hook**](https://www.geeksforgeeks.org/react-js-usememo-hook/) | The useMemo is a hook used in the functional component of react that returns a memoized value. |
| [**ReactJS useRef Hook**](https://www.geeksforgeeks.org/react-js-useref-hook/) | The useRef is a hook that allows to directly create a reference to the DOM element in the functional component |
| [**ReactJS useState Hook**](https://www.geeksforgeeks.org/reactjs-usestate-hook/) | useState() hook allows one to declare a state variable inside a function. It should be noted that one use of useState() can only be used to declare one state variable. |
| [**ReactJS useLayoutEffect Hook**](https://www.geeksforgeeks.org/reactjs-uselayouteffect-hook/) | The useLayoutEffect hook works in the same phase as componentDidMount and componentDidUpdate methods |