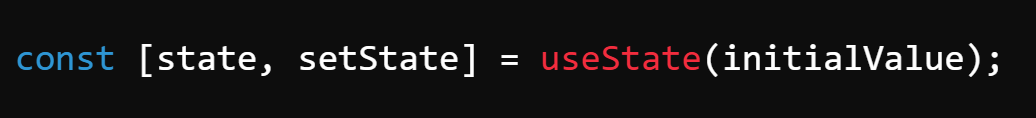
**Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)**

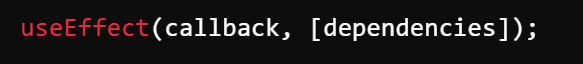
1. What are React hooks? How do useState() and useEffect() hooks work in functional components?

* React Hooks are functions that let you use state and lifecycle features in functional components without writing class components.
* Introduced in React 16.8, they allow functional components to have local state and perform side effects.

1. useState() Hook

* The useState hook is used to add state to a functional component.
* syntax:
* state: The current state value.
* setState: A function to update the state.
* initialValue: The initial state value.

2. useEffect() Hook

* The useEffect hook is used to handle side effects, such as data fetching, DOM manipulation, or setting up subscriptions.

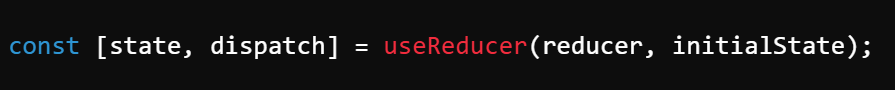
1. What problems did hooks solve in React development? Why are hooks considered an important addition to React?

* Before hooks were introduced in React 16.8, developers primarily used **class components** for managing state and lifecycle methods. However, class components had several issues

1. Complex Component Logic
2. Code Reusability & HOC Issues
3. Verbose and Difficult to Read Code
4. Difficult Lifecycle Management
5. Inefficient Performance Optimization

* Why Hooks Are an Important Addition to React.
* Hooks solved these issues by introducing a simpler and more flexible way to manage state and side effects in functional components.

1. Simplifies Code
2. Encourages Reusability
3. Better Readability & Maintainability
4. Easier Side Effect Management
5. Functional Components Everywhere
6. Performance Optimizations
7. What is useReducer ? How we use in react app?

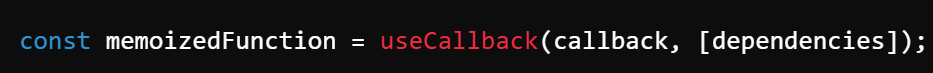
* useReducer is a hook that provides a **more structured way to manage state**, especially for **complex state logic** involving multiple actions.
* It is an alternative to useState but is particularly useful when:
* The state has multiple sub-values.
* State transitions involve **complex logic** (e.g., toggles, counters, form handling).
* You need to manage **state updates based on previous state** efficiently.
* **Syntax of useReducer**
* state: The current state value.
* dispatch: A function used to send actions to update state.
* reducer: A function that takes the current state and an action and returns the new state.
* initialState: The initial state value.
* **How it works:**
* The reducer function manages **state changes based on action types**.
* The dispatch function sends actions to the reducer.

1. What is the purpose of useCallback & useMemo Hooks?

* Both useCallback and useMemo are **performance optimization hooks** in React.
* They help prevent unnecessary re-renders by **memoizing** functions and values.

1. **useCallback Hook**

* useCallback **memoizes a function**, preventing it from being re-created on every render. This is useful when passing functions as props to child components.

**Syntax:**

callback: The function to memoize.

dependencies: The function is re-created only if dependencies change

1. **useMemo Hook**

* useMemo **memoizes a computed value**, avoiding expensive recalculations on every render.

**Syntax:**

**computeFunction(): The function whose return value is memoized.**

**dependencies: Recomputes the value only if dependencies change.**

1. What’s the Difference between the useCallback & useMemo Hooks?

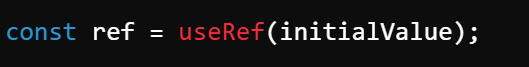
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | useCallback | |  | | --- | |  |  |  | | --- | | **useMemo** | |
| Memoizes | A function | |  | | --- | |  |  |  | | --- | | A computed value | |
| Returns | The memoized function | The memoized value |
| Use case | When passing functions to child components to prevent re-renders | When optimizing expensive calculations |
| Prevents | Function re-creation | |  | | --- | |  |  |  | | --- | | Unnecessary recalculations | |
| Primary Goal | Optimize function references | |  | | --- | |  |  |  | | --- | | Optimize computed values | |

1. What is useRef ? How to work in react app?

* useRef is a React Hook that allows you to **persist values across renders** without triggering a re-render. It is primarily used for:

1. **Accessing and manipulating DOM elements.**
2. **Storing mutable values that do not cause re-renders.**
3. **Keeping track of previous state values.**

Syntax :



ref.current holds the value, which can be changed without re-rendering the component.

Unlike useState, changing a useRef value does **not** cause a re-render.

* Uses of useref hook

1. for DOM Manipulation
2. Store Mutable Values
3. Store Previous State Values

**Lists and Keys**

1. How do you render a list of items in React? Why is it important to use keyswhen rendering lists?

* In React, lists are commonly rendered using the .map() function to iterate over an array and return a list of elements.
* **How it works:**
* The map() method loops through the items array.
* Each list item (<li>) is rendered inside a <ul>.
* A **key** is assigned to each <li> to help React manage updates efficiently.
* Why Are keys Important When Rendering Lists
* Keys help React **identify which items have changed, been added, or removed**, allowing for efficient updates to the UI.
* What Happens If You Don’t Use Keys?
  + **If** **no keys are provided, React will display a** warning **and may inefficiently re-render the entire list instead of making targeted updates.**

1. What are keys in React, and what happens if you do not provide a unique key?

* Keys are **special attributes** in React that help identify which list items have changed, been added, or removed. They allow React to efficiently update the UI when rendering lists.
* **Why Are Keys Important?**
* **Improves Performance:** Helps React efficiently re-render only the changed items.
* **Prevents UI Bugs:** Avoids unnecessary reordering or re-creation of elements.
* **Optimizes Virtual DOM Diffing:** Helps React track and update individual elements instead of re-rendering the whole list.
* **What Happens If You Don’t Provide a Unique Key?**

1. **React gives a warning:**

"Each child in a list should have a unique key prop."

1. **Performance issues:** React may re-render the entire list instead of updating only the changed items.
2. **Incorrect UI updates:** React might mistakenly **reuse elements** in the wrong order, leading to unexpected behavior.