What is Hook?

React Hooks are tools that allow you to use state and other React features without writing class components. They're designed to simplify your code and make it easier to share logic across components.

Simplify Your Code: Avoid the complexity of class components.

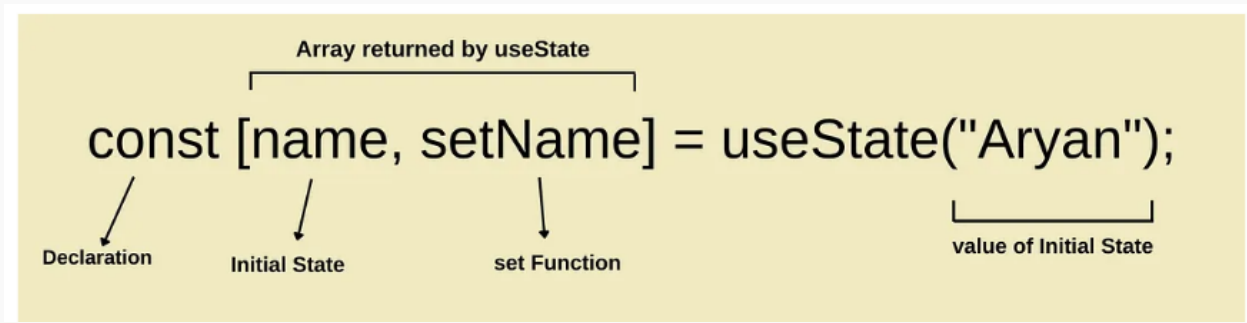
- Reusable Logic: Easily share and reuse stateful logic.
- Built-in Hooks: Such as `useState`, `useEffect`, and `useContext` for managing state, side effects, and context.

They can only be called **at the top level of your functional component** or **inside custom Hooks**. You cannot call Hooks inside loops, conditions, or nested functions. This rule is crucial for React to reliably track the state of your Hooks.

useState Hook

useState hook is used for managing states within the functional components. States are used to manage and store some data within a component. Whenever a state changes React will automatically re-render the component to display the updated value.

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



Counter Code:

```
import './App.css'
import { useState } from "react";


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

  function decrementCount(){
    setCount(prevCount => prevCount - 1)
  }

  function incrementCount(){
    setCount(prevCount => prevCount + 1)
  }

  return (
    <div className='counter-container'>
      <button onClick={decrementCount}>-</button>
      <span>{count}</span>
      <button onClick={incrementCount}>+</button>
    </div>
  );
}
```

```
export default App;
```

Css code:

```
/* App.css */

.counter-container {
  display: flex;
  align-items: center;
  justify-content: center;
  margin-top: 50px;
  font-size: 24px;
  font-family: Arial, sans-serif;
}

.counter-container button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 10px 20px;
  margin: 0 10px;
  font-size: 20px;
  cursor: pointer;
  border-radius: 4px;
}

.counter-container button:hover {
  background-color: #0056b3;
}

.counter-container span {
  min-width: 40px;
  text-align: center;
}
```

Mirror Input:

```
import './App.css'
import { useState } from "react";


function App() {
  const [text, setText] = useState('hello');

  function handleChange(e) {
    setText(e.target.value);
  }

  return (
    <>
      <input value={text} onChange={handleChange} />
      <p>You typed: {text}</p>
      <button onClick={() => setText('hello')}>
        Reset
      </button>
    </>
  );
}

export default App;
```

```
/* App.css */

body {
  font-family: Arial, sans-serif;
  padding: 20px;
}

input {
  padding: 8px;
  font-size: 16px;
  margin-right: 10px;
```

```
}

button {
  padding: 8px 12px;
  font-size: 16px;
  cursor: pointer;
  border-radius: 10px;
}

p {
  margin-top: 10px;
  font-size: 18px;
}
```

Toggler:-

```
import './App.css'
import { useState } from "react";


function App() {
  const [isOn, setIsOn] = useState(false); // false = OFF initially

  // Step 2: Create a function to toggle the state
  const handleToggle = () => {
    setIsOn(!isOn); // Flips true to false or false to true
  };

  // Step 3: Return the UI
  return (
    <div className='container'>
      <h2>Toggler App</h2>

      {/* Step 4: Show ON or OFF based on state */}
      <p className='text'>
        The switch is: <strong>{isOn ? "ON" : "OFF"}</strong>
      </p>
```

```
      {/* Step 5: Button to toggle */}
      <button onClick={handleToggle} className='button'>
        Toggle
      </button>
    </div>
  );

}

export default App;
```

```
.container{
  text-align: center;
  margin-top: 50px;
  font-family: sans-serif;
}

.text{
  font-size: 24px;
  margin: 20px 0;
}

.button{
  padding: 10px 20px;
  font-size: 18px;
  cursor: pointer;
}
```

❌ Wrong way (mutating state directly):

jsx Copy Edit

```jsx
const [numbers, setNumbers] = useState([1, 2, 3]);

numbers.push(4); // ❌ BAD! Directly changes the original array
setNumbers(numbers);
```

This doesn't work properly because React doesn't detect the change correctly.

✅ Right way (immutably updating state):

jsx Copy Edit

```jsx
const [numbers, setNumbers] = useState([1, 2, 3]);

setNumbers([...numbers, 4]); // ✅ Create a new array with the new value
```

Here, we're creating a **new array** with all the old values + the new one.
React sees this **new object** and knows it needs to re-render.

https://blog.webdevsimplified.com/2020-04/use-state/

https://react.dev/reference/react/useState