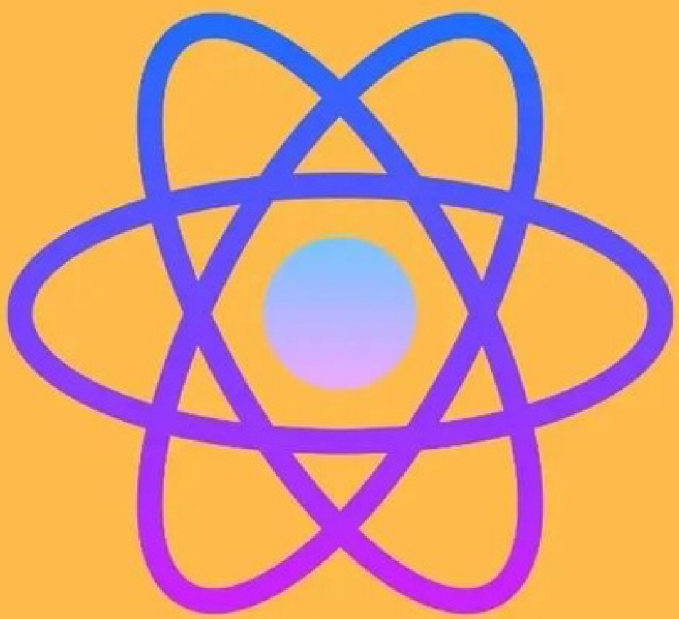


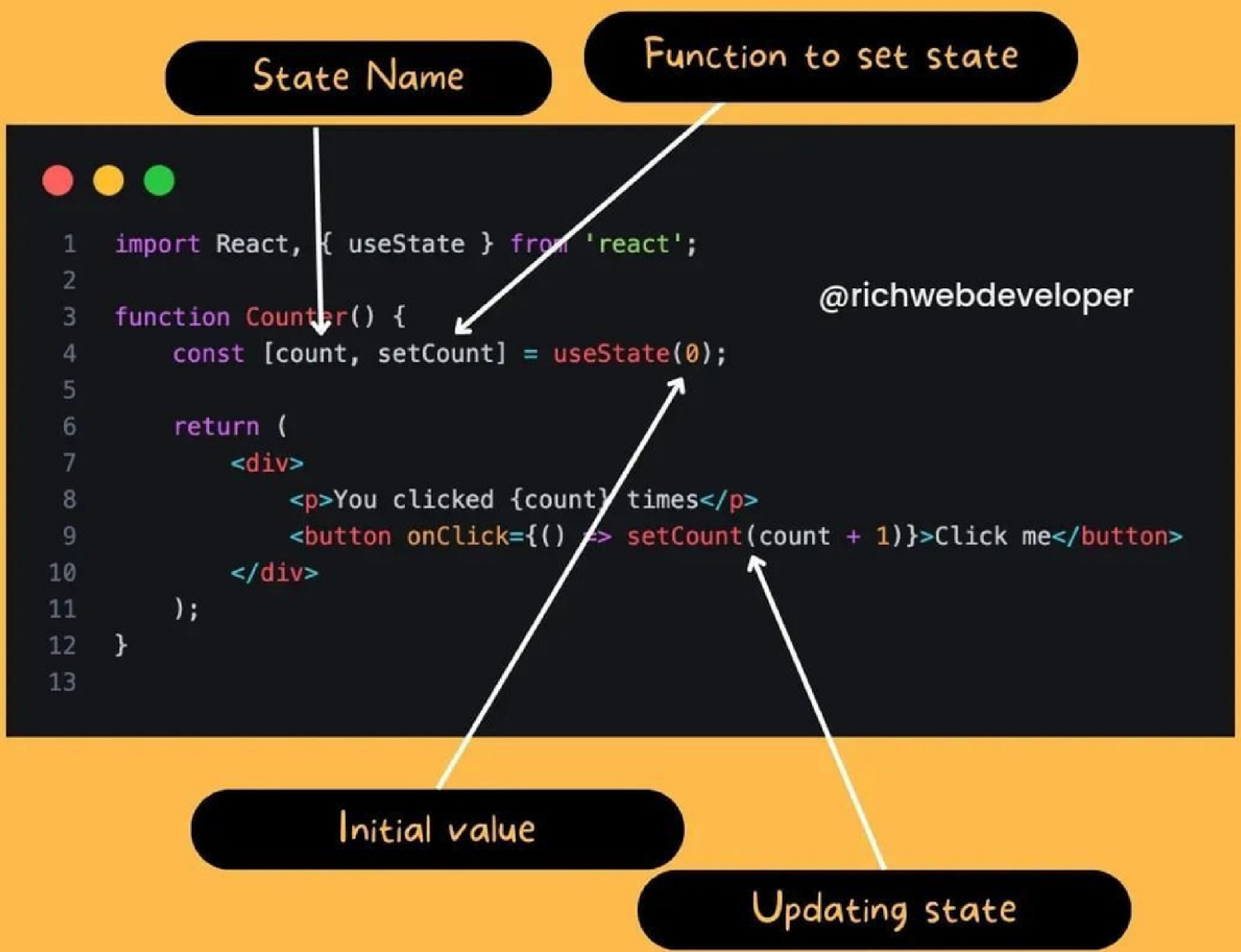
React Hooks Cheatsheet

@richwebdeveloper

1. useState

This Hook allows you to add state to functional components. The useState function returns a pair: the current state and a function that updates it.

Example:




```
1  import React, { useState } from 'react';
2
3  function Counter() {
4      const [count, setCount] = useState(0);
5
6      return (
7          <div>
8              <p>You clicked {count} times</p>
9              <button onClick={() => setCount(count + 1)}>Click me</button>
10         </div>
11     );
12 }
13
```

2. useEffect

This Hook lets you perform side effects in functional components. It's a close replacement for componentDidMount, componentDidUpdate, and componentWillUnmount in class components.

Example:

3. useContext

This Hook lets you subscribe to React context without introducing nesting. It accepts a context object and returns the current context value for that context.

Example:

4. useReducer

An alternative to useState. Accepts a reducer of type (state, action) => newState, and returns the current state paired with a dispatch method.
Example:

It is particularly useful when the state logic is complex and involves multiple sub-values, or when the next state depends on the previous one. useReducer is an alternative to useState.

While useState is great for handling independent pieces of state, useReducer excels at handling more complex, interconnected state that involves multiple changes in an atomic and predictable manner.

Example(next slide):

@richwebdeveloper


Code to update state of dispatched action
import React, { useReducer } from 'react';
const initialState = {count: 0};
function reducer(state, action) {
switch (action.type) {
case 'increment':
return {count: state.count + 1};
case 'decrement':
return {count: state.count - 1};
default:
throw new Error();
}
}
function Counter() {
const [state, dispatch] = useReducer(reducer, initialState);
return (
<>
Count: {state.count}
<button onClick={() => dispatch({type: 'decrement'})}>-</button>
<button onClick={() => dispatch({type: 'increment'})}>+</button>
</>
);
}
@richwebdeveloper
updated state value
Updating state with action

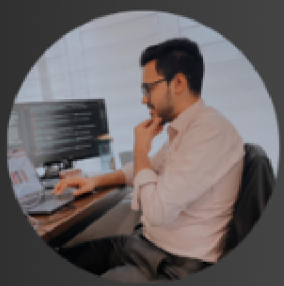
5. useRef

This Hook creates a mutable ref object whose .current property is initialized to the passed argument. It's handy for keeping any mutable value around similar to how you'd use instance fields in classes.

Example:


```
1  import React, { useRef } from 'react';
2
3  function TextInputWithFocusButton() {
4      const inputEl = useRef(null);
5      const onButtonClick = () => {
6          inputEl.current.focus();
7      };
8      return (
9          <>
10             <input ref={inputEl} type="text" />
11             <button onClick={onButtonClick}>Focus the input</button>
12         </>
13     );
14 }
15
```

DID YOU FIND IT USEFUL?



Follow @Ch Bappy