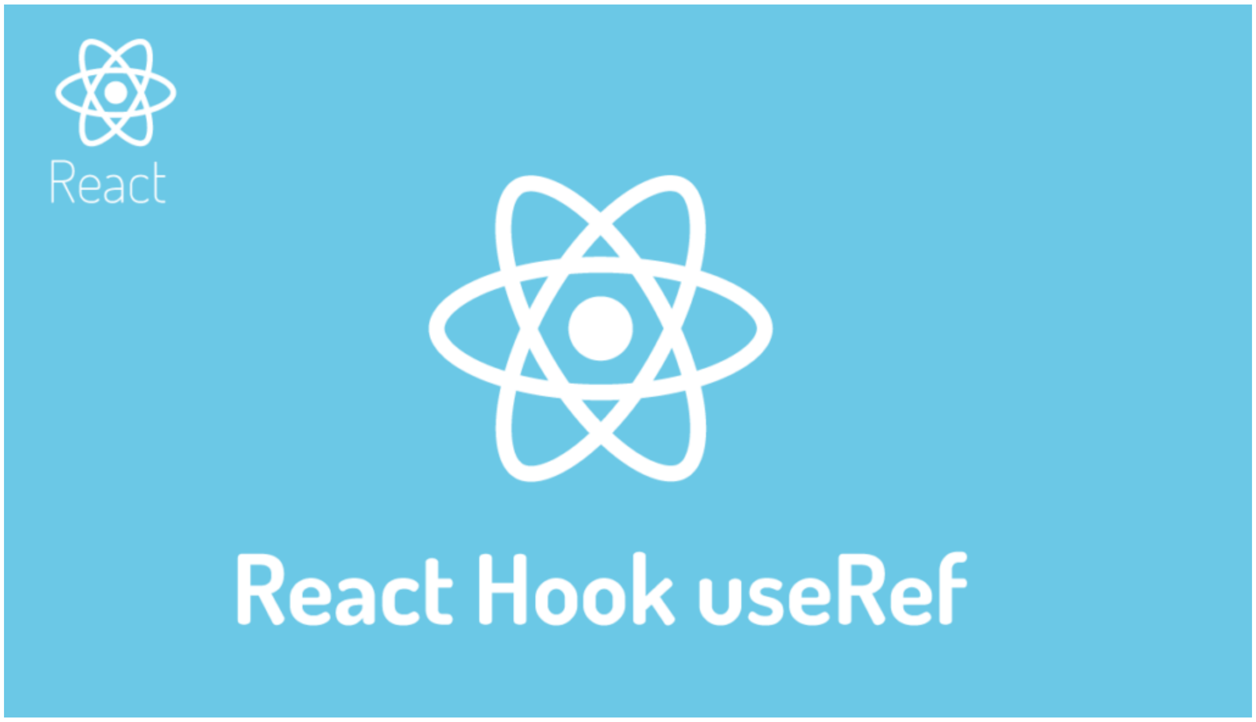
React useRef Hook



What is useRef hook in React?

In React useRef hook is a function that provides us with the -

- functionality for persisting values between renders
- can be used to store a mutable value that does not cause a re-render when updated
- and can be used to access a DOM element directly.

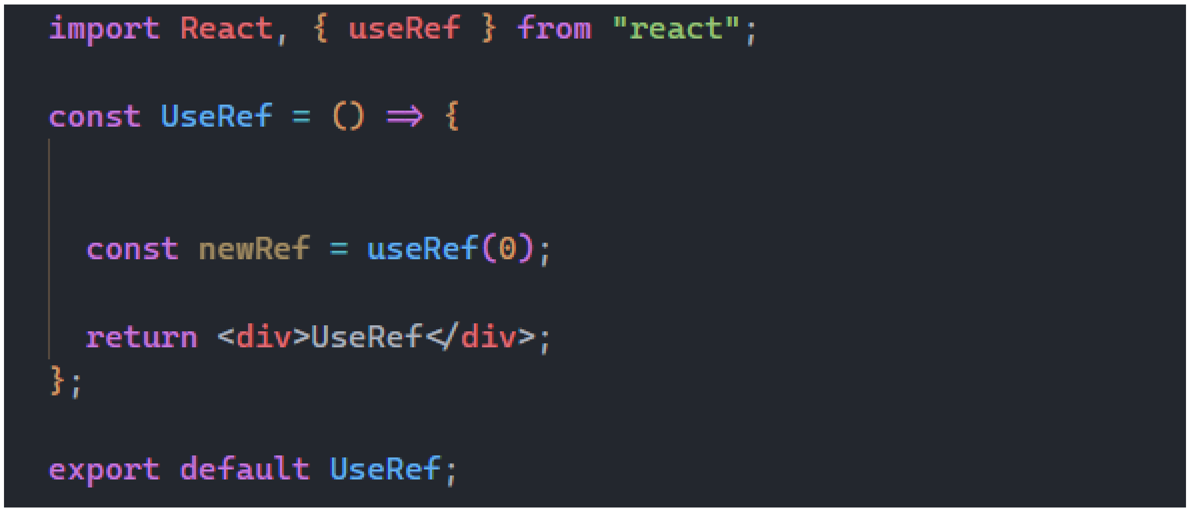
The useRef hook is majorly used for making mutable states and directly accessing the DOM element of the HTML.

The useRef hook consists of a method to access the content within it.

- .current

The .current method provides us with the current value stored in the useRef hook.

The syntax for the useRef Hook -

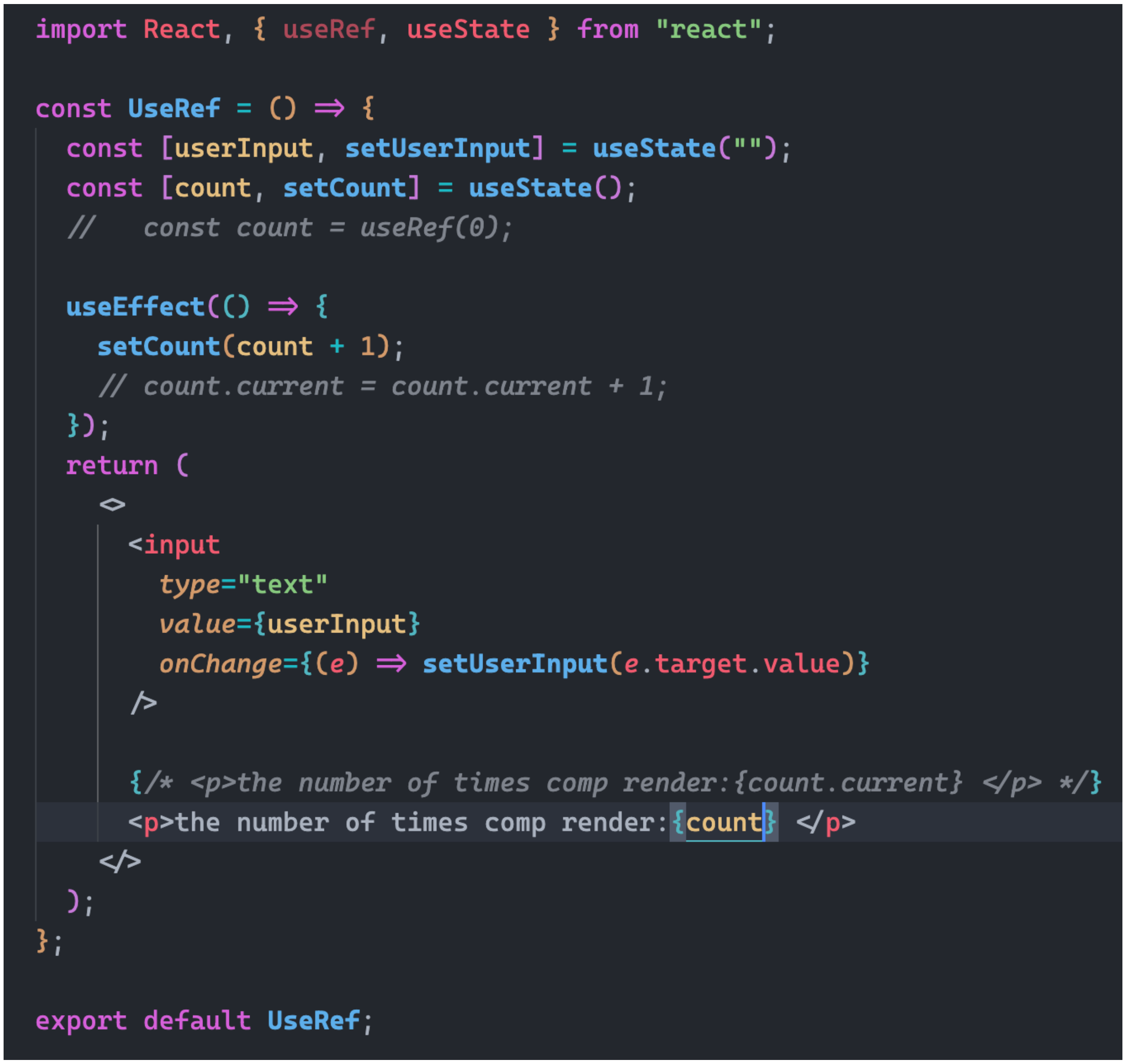


```
import React, { useRef } from "react";

const UseRef = () => {


  const newRef = useRef(0);

  return <div>UseRef</div>;
};

export default UseRef;
```

useRef for mutable states.

The useRef is also similar to the useState hook but with a catch. The catch is that when we use useRef hook for any state updation the component does not re-render itself.

For example -
If we use useState for state management, then every time when the state is being changed the component in which it is being re-rendered is.
But with useRef the component does not re-render itself, it does not perform any reloading



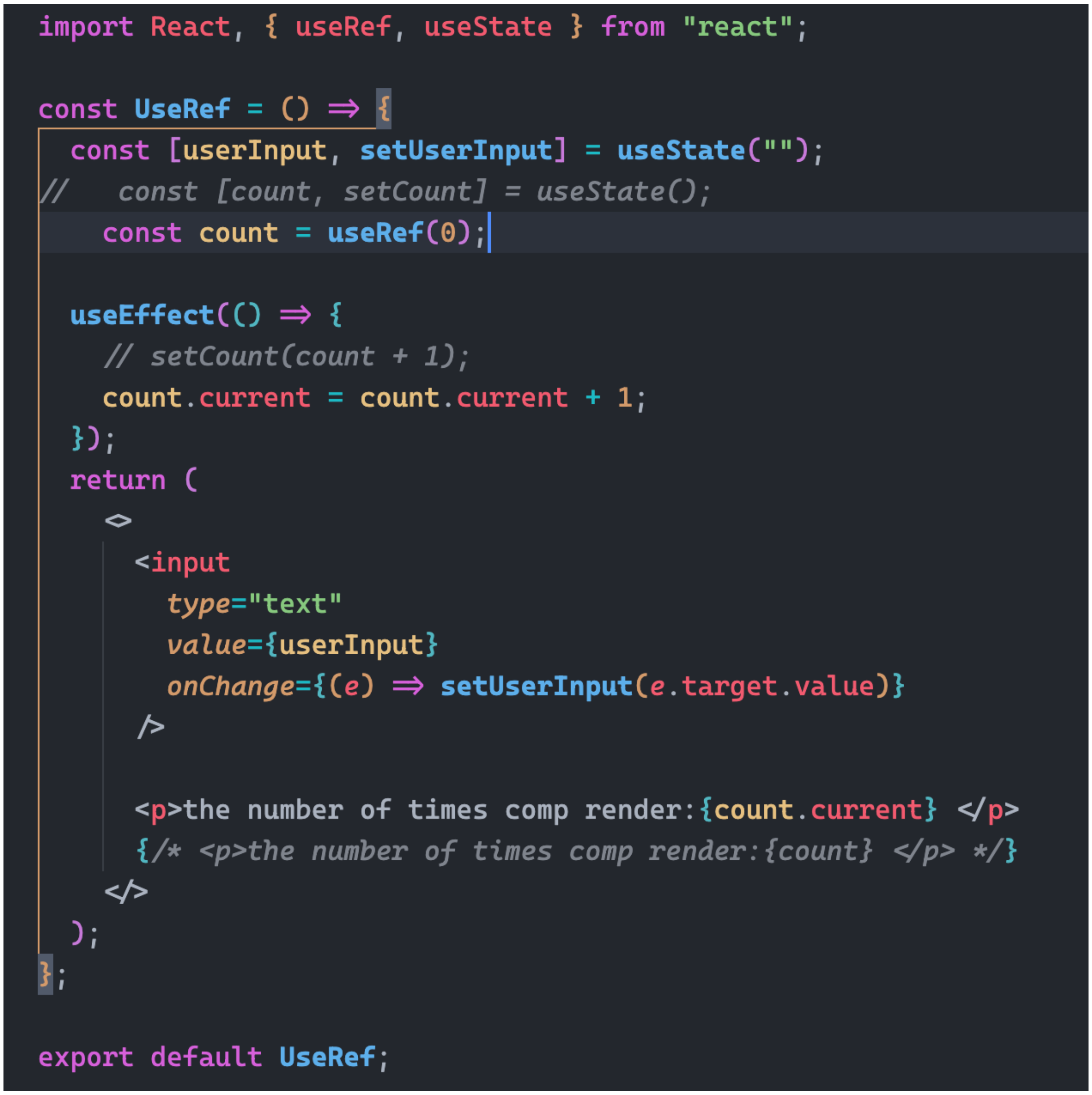
```jsx
import React, { useRef, useState } from "react";

const UseRef = () => {
  const [userInput, setUserInput] = useState("");
  const [count, setCount] = useState();
  //   const count = useRef(0);

  useEffect(() => {
    setCount(count + 1);
    // count.current = count.current + 1;
  });
  return (
    <>
      <input
        type="text"
        value={userInput}
        onChange={(e) => setUserInput(e.target.value)}
      />

      {/* <p>the number of times comp render:{count.current} </p> */}
      <p>the number of times comp render:{count} </p>
    </>
  );
};

export default UseRef;
```

In the above example, we are trying to get the number of times the component renders.

But when we run this piece of code it results in an infinite loop type of situation, that is because setUserInput function renders the component, and every time it sets the count to one, that how the loop goes on.

To tackle this situation we use useRef as it does not re-render the component.

For example -



```
import React, { useRef, useState } from "react";

const UseRef = () => {
  const [userInput, setUserInput] = useState("");
//   const [count, setCount] = useState();
    const count = useRef(0);

  useEffect(() => {
    // setCount(count + 1);
    count.current = count.current + 1;
  });
  return (
    <>
      <input
        type="text"
        value={userInput}
        onChange={(e) => setUserInput(e.target.value)}
      />

      <p>the number of times comp render:{count.current} </p>
      {/* <p>the number of times comp render:{count} </p> */}
    </>
  );
};

export default UseRef;
```

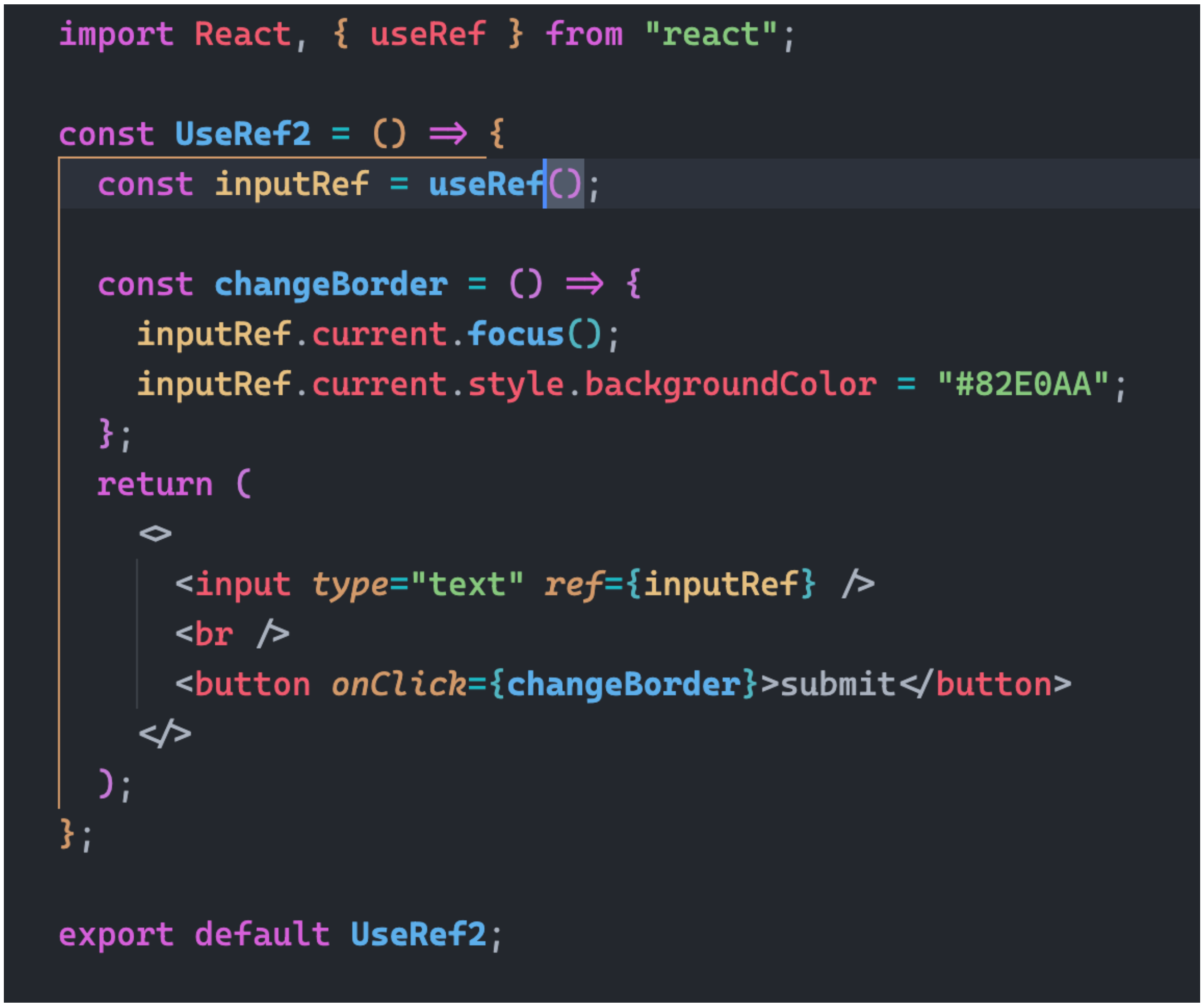
Now if we see this example we have used useRef for storing the render count and as we know useRef does not re-render the component on the change of value so we can now count the number of renders whenever there is a change in state.

useRef for accessing the DOM element directly

useRef hooks provide us the power to manipulate the DOM elements and properties directly.
We don't have to rely on other functions or procedures to access to the DOM.

useRef hook provides us the direct reference to the element, and once we have the reference then we can perform changes according to our needs.

Let us see it with an example -



```
import React, { useRef } from "react";

const UseRef2 = () => {
  const inputRef = useRef();

  const changeBorder = () => {
    inputRef.current.focus();
    inputRef.current.style.backgroundColor = "#82E0AA";
  };
  return (
    <>
      <input type="text" ref={inputRef} />
      <br />
      <button onClick={changeBorder}>submit</button>
    </>
  );
};

export default UseRef2;
```

Here we have given the reference to the input tag with the useRef instance named “inputRef”.

Now we can access the DOM properties of the input tag with the new useRef instance that we created.

As we can see that we are able to perform activities such as focus or background color change with click events.

Similarly, we can handle more DOM properties of any HTML element on any event handler, with the use of useRef.