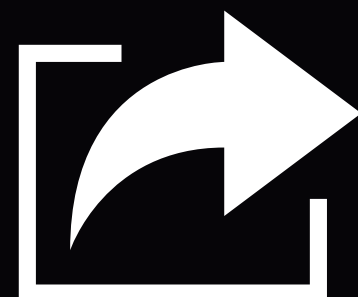
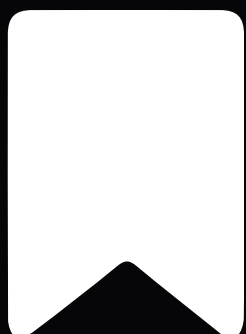
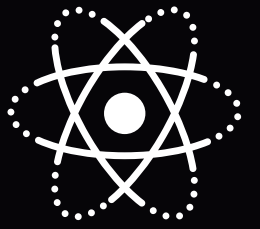


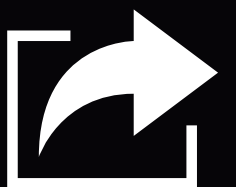
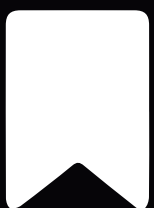
React useState Hook

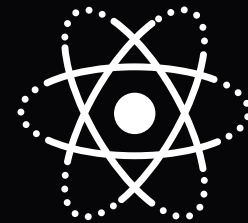




> What is the useState Hook?

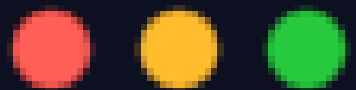
The useState hook is a built-in React function that allows functional components to manage state. Before hooks were introduced, state management was only possible in class components. However, with useState, we can now manage state directly within functional components, leading to cleaner and more modular code.



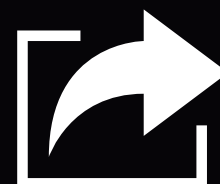


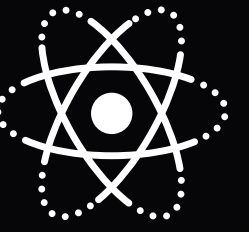
> Syntax Overview

Here's the basic syntax for using the useState hook:

A small graphic of a terminal window with three colored circles (red, yellow, green) in the top-left corner.

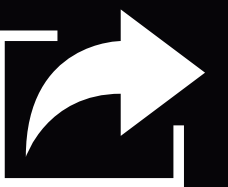
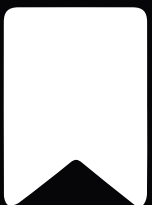
```
const [state, setState] =  
  useState(initialValue);
```

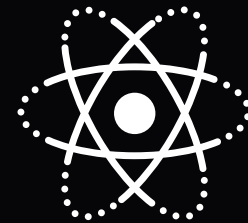




> Syntax Overview

- **state:** This is the current state value.
- **setState:** This is a function that allows you to update the state.
- **initialValue:** This is the initial value of the state, which can be of any data type (number, string, object, array, etc.).





> Syntax Overview

Example in Action

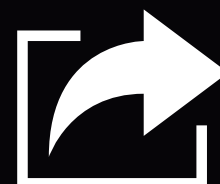


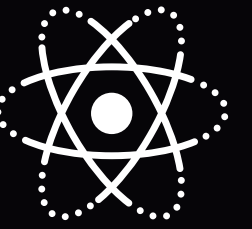
```
import React, { useState } from
'react';

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

  return (
    <div>
      <p>Current Count: {count}</p>
      <button onClick={() =>
setCount(count + 1)}>Increment</button>
    </div>
  );
}

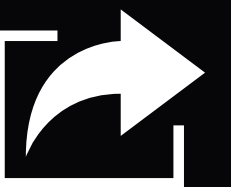
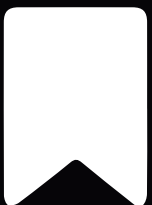
export default Counter;
```

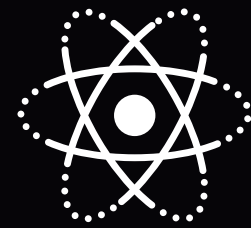




> Syntax Overview

- In this example:
- We declare a state variable count with an initial value of 0.
- The setCount function is used to update the count state whenever the button is clicked.
- The component re-renders automatically when the state changes, updating the displayed count.





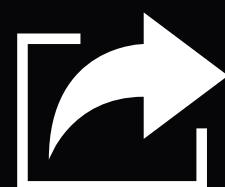
> Why useState is Important ??

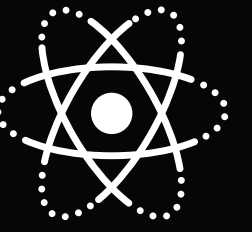
1. Reactivity

- The useState hook enables your components to be reactive. Whenever the state changes, React automatically re-renders the component, ensuring that the UI stays in sync with the underlying data.

2. Simplicity and Readability

- By using useState in functional components, you can avoid the verbosity and complexity associated with class components. This leads to more concise and readable code, making it easier for both you and your team to maintain.

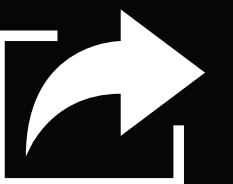
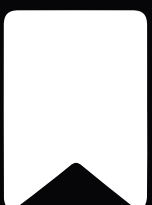


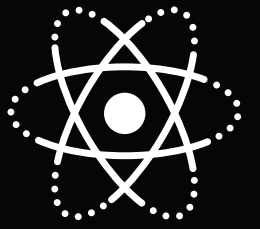


> Why useState is Important ??

3. Modularity

Each call to `useState` creates an isolated piece of state. This modular approach allows you to manage multiple state variables independently, leading to better-organized code.





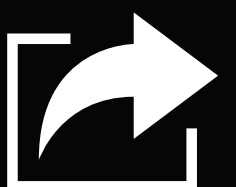
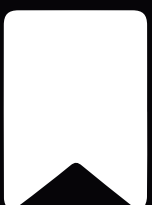
> **Best Practices for Using useState**

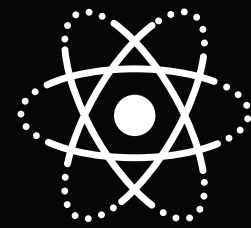
1. Initialize State Thoughtfully

Always provide an initial state that makes sense for your application. This could be a number, a string, an empty array, or even null. Thoughtful initialization helps prevent unexpected behavior in your application.

2. Use Multiple useState Hooks for Different Pieces of State

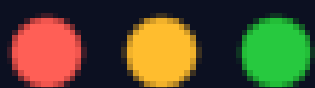
Rather than using a single state object to manage multiple state values, it's often cleaner to use multiple useState hooks. This makes your code more readable and easier to manage.



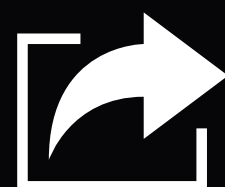


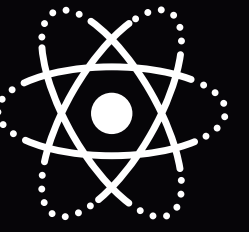
> Best Practices for Using useState

For example:



```
const [firstName, setFirstName] =  
  useState('');  
const [lastName, setLastName] =  
  useState('');  
const [age, setAge] = useState(0);
```

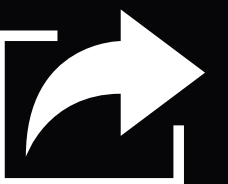
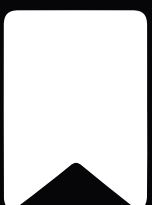


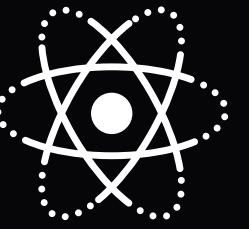


> Best Practices for Using useState

3. Avoid Overusing useState

While useState is a powerful tool, it's important to use it judiciously. Combine it with other hooks like useEffect for managing side effects or useContext for global state management, to build a well-rounded and efficient application.

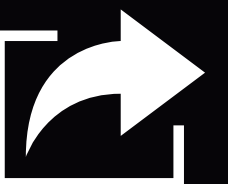
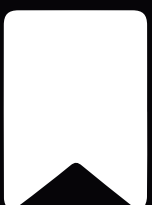


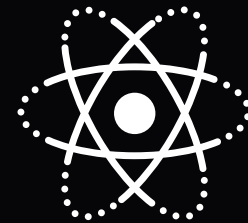


> Conclusion

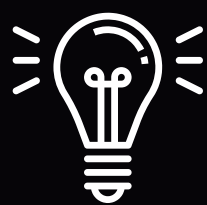
Mastering the `useState` hook is a crucial step in becoming proficient with React. It provides a simple yet powerful way to manage state within functional components, making your applications more dynamic and interactive.

Today's deep dive into `useState` has solidified my understanding of state management in React. I'm excited to continue building upon this knowledge as I progress further in my coding journey.





Hassan Bin Wagar
Follow me
to know more!



Trick

