[notes](https://albseb511.github.io/notes/hooks/index.html" \l "/2/5)

# Hooks

Introduced in 2018 for state management, Hooks solve a wide variety of seemingly unconnected problems in React.

## It’s hard to reuse stateful logic between components

With Hooks, you can extract stateful logic from a component so it can be tested independently and reused. Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

This can be done with custom Hook ( we will discuss this in another class )

## Complex components become hard to understand

Hooks let you split one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data), rather than forcing a split based on lifecycle methods.

This can be done with useEffect Hook\* we will cover this later

## Classes confuse both people and machines

Hooks let you use more of React’s features without classes. Conceptually, React components have always been closer to functions. Hooks embrace functions, but without sacrificing the practical spirit of React. Hooks provide access to imperative escape hatches and don’t require you to learn complex functional or reactive programming techniques.

Note: `There are no plans to remove classes from React.`

Read more [Hooks Intro](https://reactjs.org/docs/hooks-intro.html)

## Hooks at a Glance

`They let you use state and other React features without writing a class.`

## What are hooks?

Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

## When would I use a Hook?

If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component.

APIs we need to learn first

- `useState`  
- `useEffect` ( we will cover in another session )

You must have noticed that the APIs begin with a `use`, this is intentional, and it is recommended practise to keep all hooks apis starting with `use`

## useState Hook

You might have previously known functional components as “stateless components”. By introducing the ability to use React state from these, so we prefer the name “function components”.

Hooks don’t work inside classes. But you can use them instead of writing classes.

## before

Equivalent class component

```jsx

class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }

  render() {  
    return (  
      <div>  
        <p>You clicked {this.state.count} times</p>  
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>  
          Click me  
        </button>  
      </div>  
    );  
  }  
}

```

## after  
```jsx

import React, { useState } from 'react';  
function Example() {  
  // Declare a new state variable, which we'll call "count"    
  //   array destructuring  
  const [count, setCount] = useState(0);

  return (  
    <div>  
      <p>You clicked {count} times</p>  
      <button onClick={() => setCount(count + 1)}>  
        Click me  
      </button>  
    </div>  
  );  
}

```

## useState

`useState` returns a pair: the current state value and a function that lets you update it.

Declaring multiple values

```jsx

function ExampleWithManyStates() {  
  // Declare multiple state variables!  
  const [age, setAge] = useState(42);  
  const [fruit, setFruit] = useState('banana');  
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);  
  // ...  
}

```

## TIP

Declaring state variables as a pair of `[something, setSomething]` is recommended

[Docs](https://reactjs.org/docs/hooks-state.html)

For a more hook friendly documentation, the react community has maintained a hooks based docs.  
- You can find it [here](https://reactwithhooks.netlify.app/)

##  What is the difference between state and props?

props (short for “properties”) and state are both plain JavaScript objects. While both hold information that influences the output of render, they are different in one important way: props get passed to the component (similar to function parameters) whereas state is managed within the component (similar to variables declared within a function).

Here are some good resources for further reading on when to use props vs state:

[https://github.com/uberVU/react-guide/blob/master/props-vs-state.md](https://github.com/uberVU/react-guide/blob/master/props-vs-state.md)

[https://lucybain.com/blog/2016/react-state-vs-pros/](https://lucybain.com/blog/2016/react-state-vs-pros/)

## Rules of Hooks

#### Only Call Hooks at the Top Level  
Don’t call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns. By following this rule, you ensure that Hooks are called in the same order each time a component renders. That’s what allows React to correctly preserve the state of Hooks between multiple useState and useEffect calls. ( [explaination](https://reactjs.org/docs/hooks-rules.html#explanation)

#### Only Call Hooks from React Functions  
Don’t call Hooks from regular JavaScript functions. Instead, you can:

- ✅ Call Hooks from React function components.  
- ✅ Call Hooks from custom Hooks (we’ll learn about them later).

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

[Docs - Rules of hooks](https://reactjs.org/docs/hooks-rules.html)