MobileDrive

Introduction to React

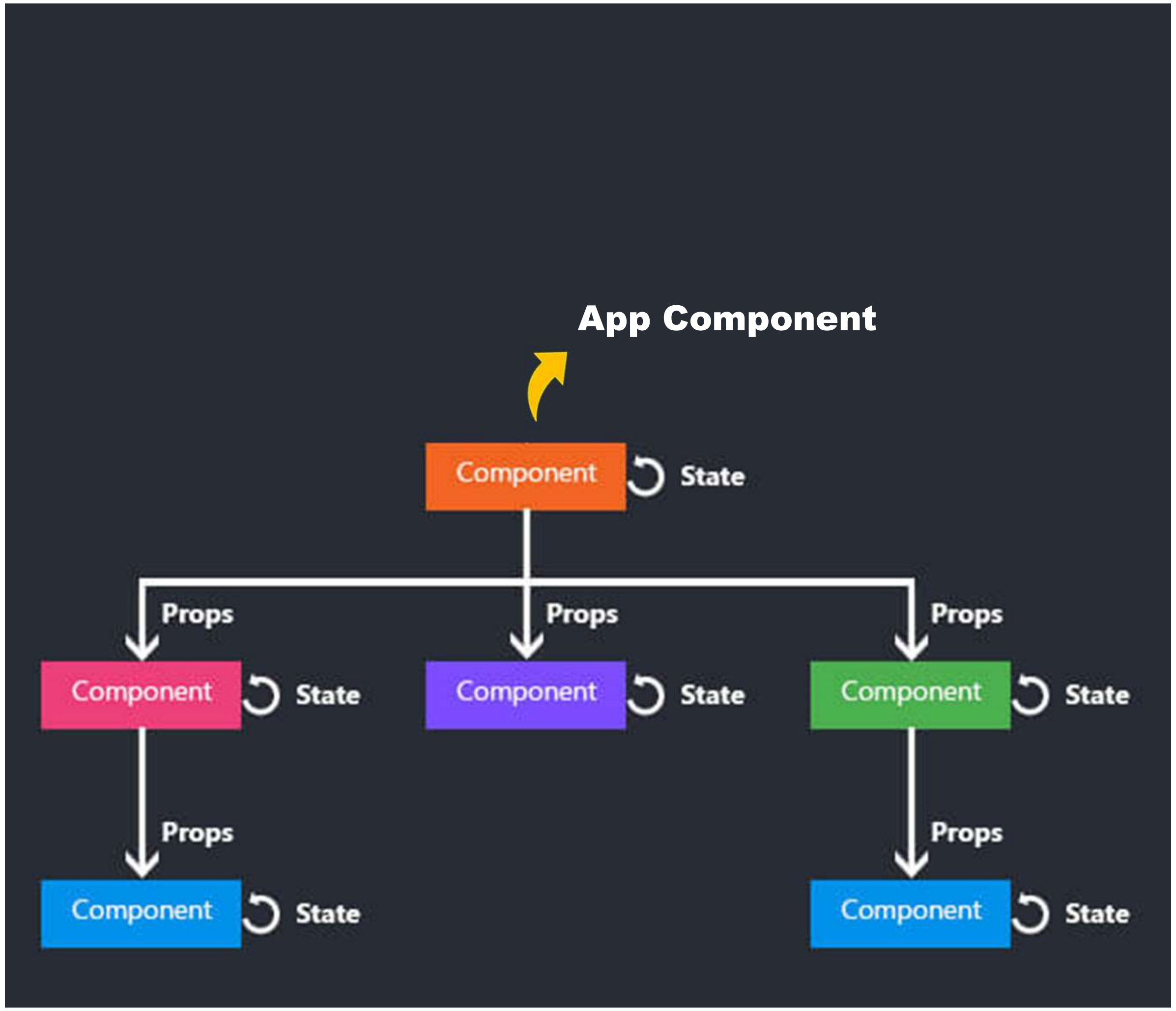
2021.12.27

Agenda

MobileDrive

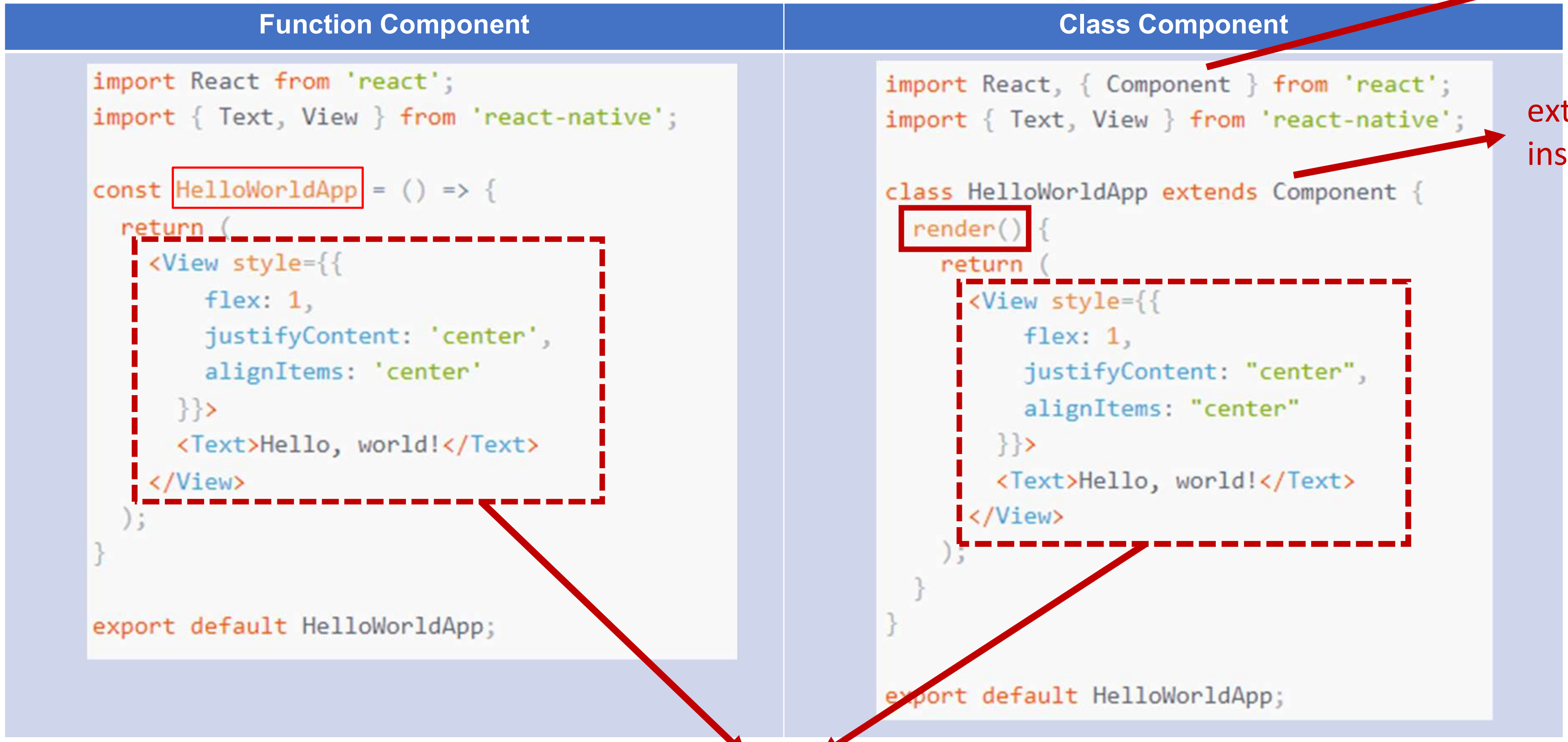
- React App Structure
- Component
- React Element & JSX
- State & Prop
- Component LifeCycle
- State & effect hooks
- Render prop, HOC, custom hooks

React Structure : Component Tree



Function Components and Class Components

- Components can be defined as **class** or **function**.
- Before React Native 0.59, only class components can use **state**.
- **Hooks** were introduced in *React Native 0.59*, allowing function component to use **state**
- Function Component name must be **CAPITLAZIED**.
- Class component must overwrite *render ()*



Function Component	Class Component
See code below	See code below

Function Component

```
import React from 'react';
import { Text, View } from 'react-native';

const HelloWorldApp = () => {
  return (
    <View style={{
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
      }}>
      <Text>Hello, world!</Text>
    </View>
  );
}

export default HelloWorldApp;
```

Class Component

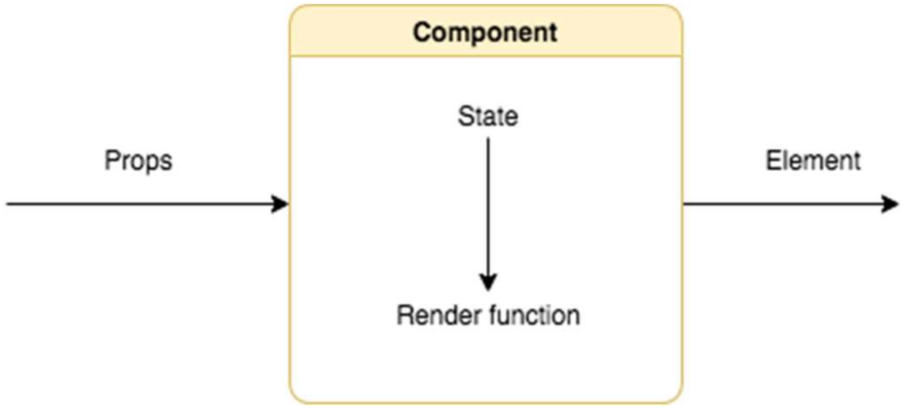
```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center"
        }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}

export default HelloWorldApp;
```

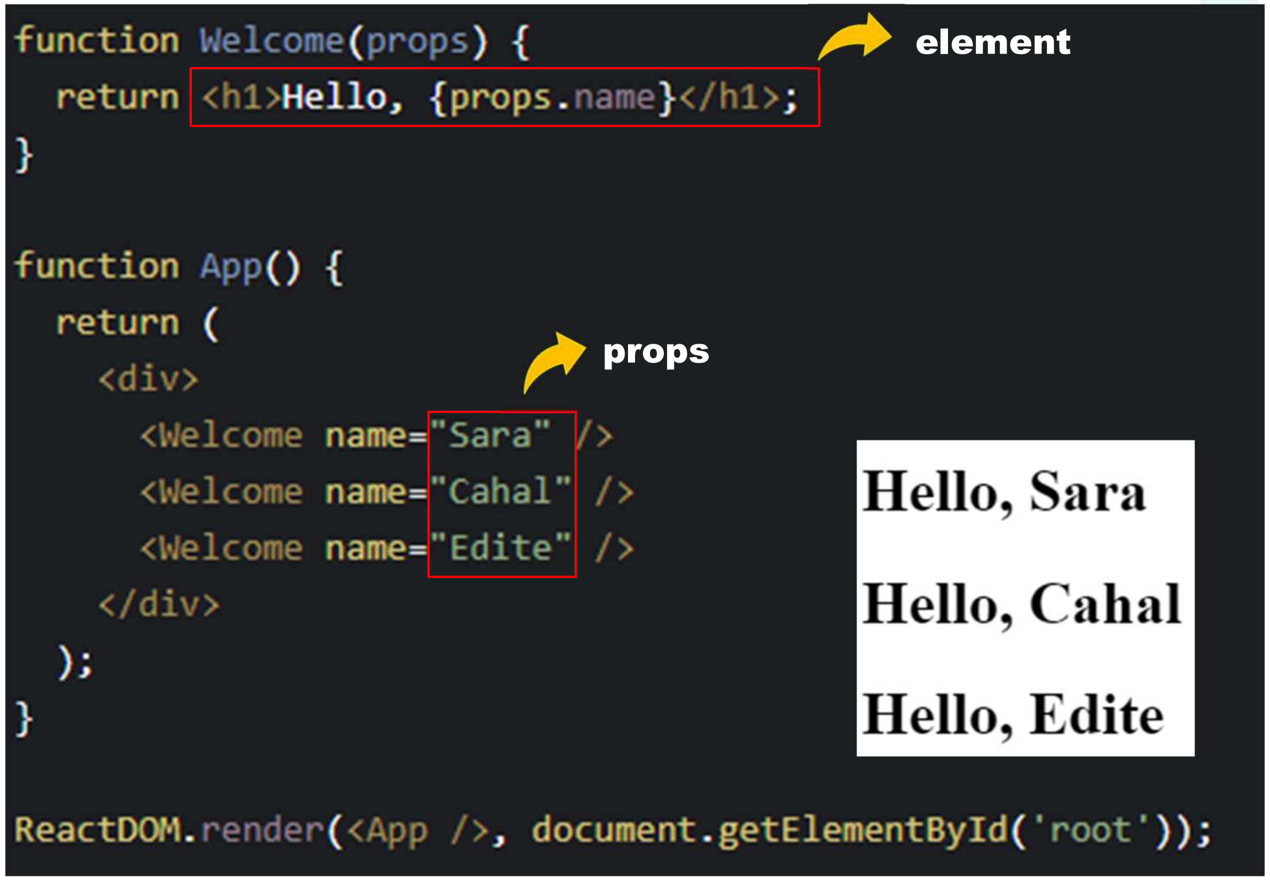
import **Component** from **React**

extending **Component** instead of as a function

Element(JSX)

Functional (Stateless) Component example





```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Hello, Sara

Hello, Cahal

Hello, Edite

React element

建立 React 應用程式最小的單位是 element。

一個 element 描述你想要在螢幕上所看到的：

```
const element = <h1>Hello, world</h1>;
```

React element

JSX	React.createElement()

JSX

```jsx
const element = (
  <h1 className="greeting">
    Hello, World!
  </h1>
);
```

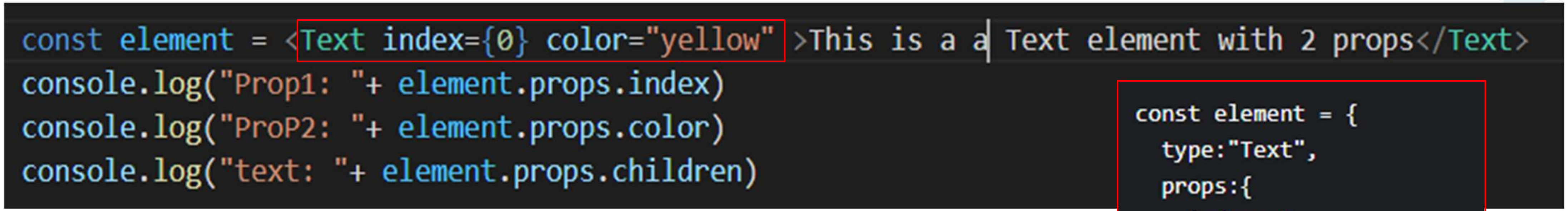
React.createElement()

```js
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, World!'
);
```

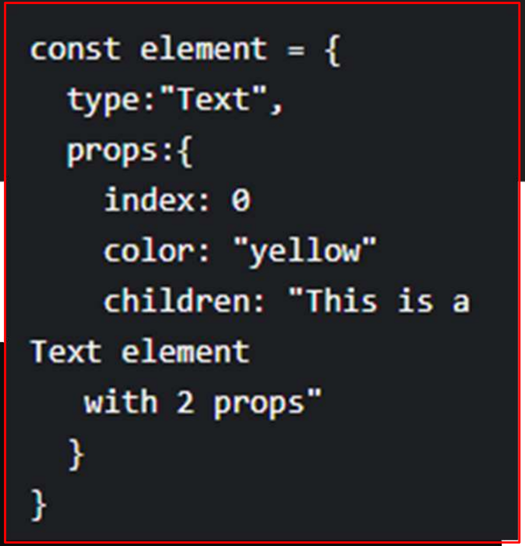
```js
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

```js
React.createElement(
  type,
  [props],
  [...children]
)
```

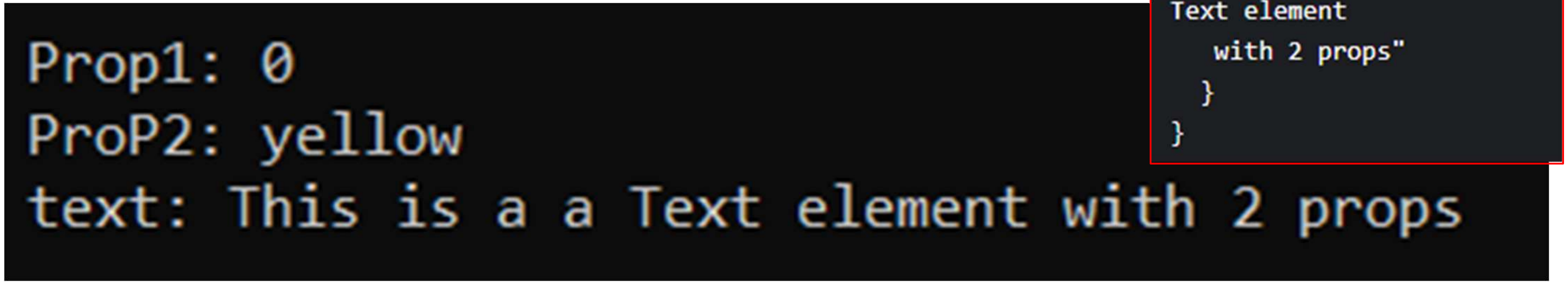
JSX : attribute



```
const element = <Text index={0} color="yellow" >This is a a Text element with 2 props</Text>
console.log("Prop1: "+ element.props.index)
console.log("ProP2: "+ element.props.color)
console.log("text: "+ element.props.children)
```

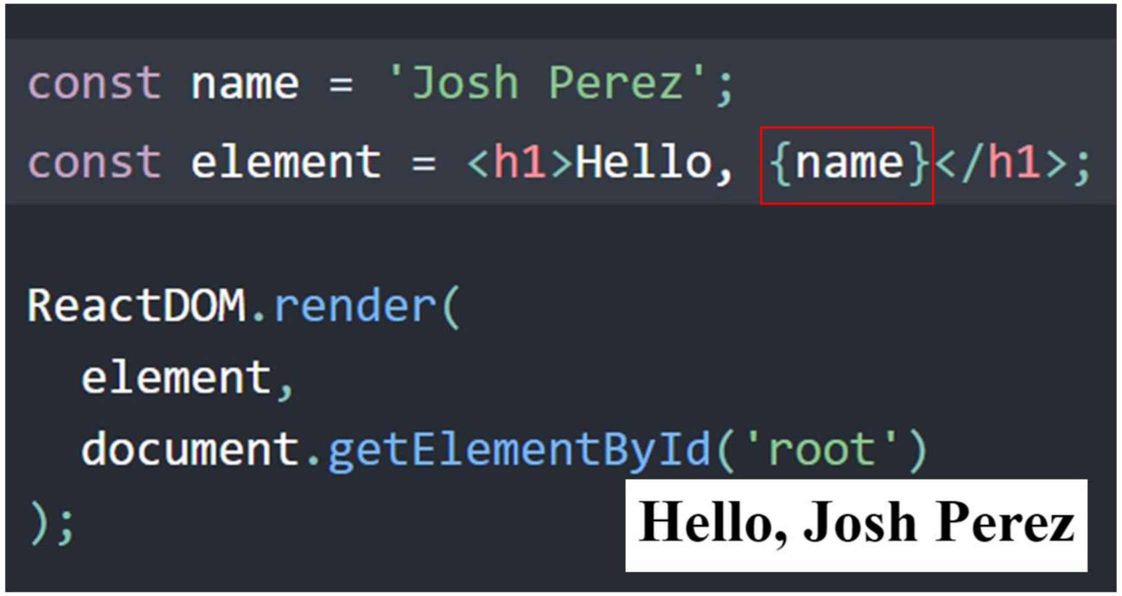


```
const element = {
  type:"Text",
  props:{
    index: 0
    color: "yellow"
    children: "This is a
Text element
   with 2 props"
  }
}
```



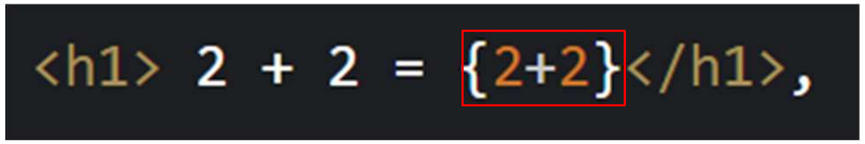
```
Prop1: 0
ProP2: yellow
text: This is a a Text element with 2 props
```

JSX: using expression



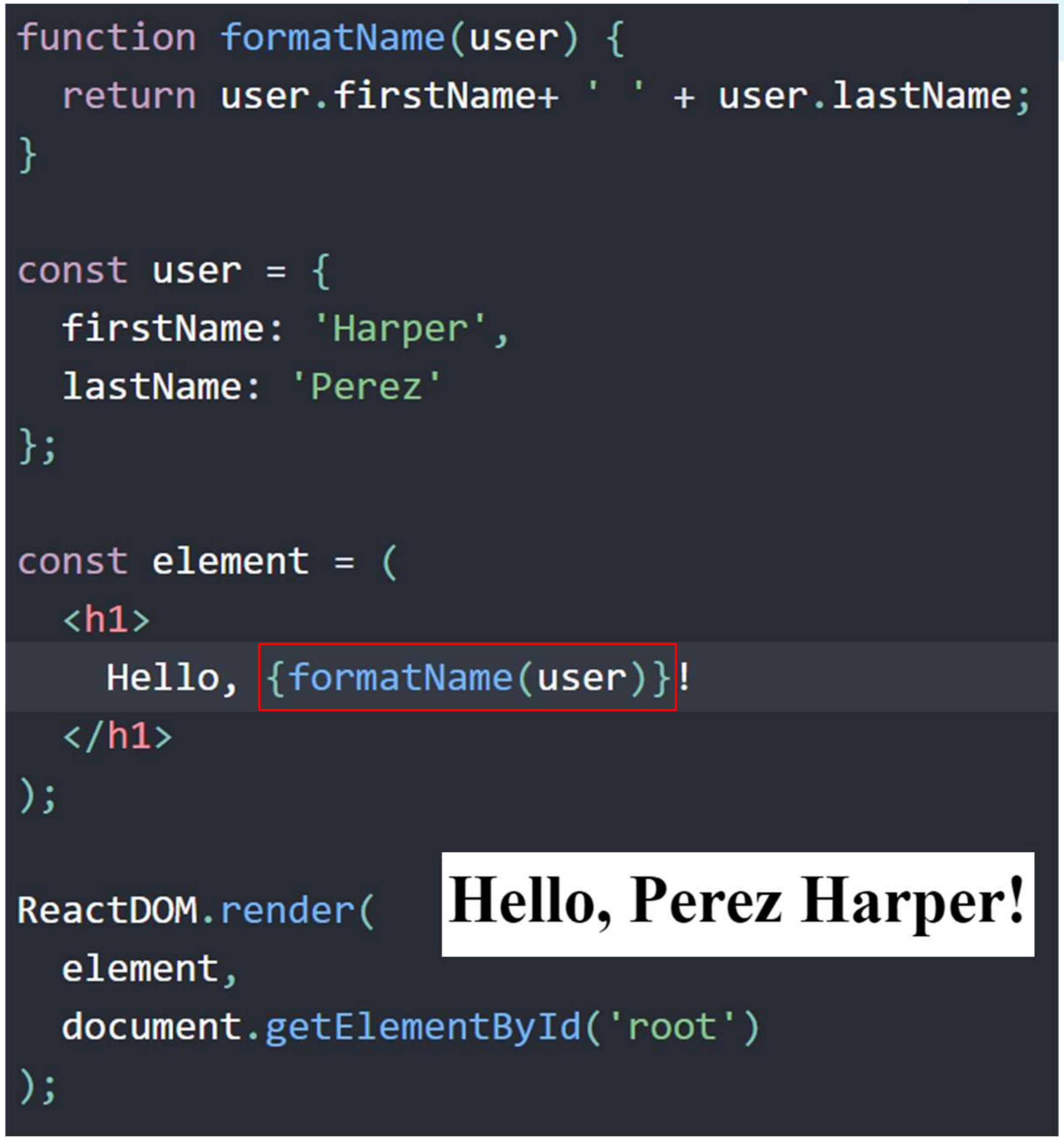
```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Hello, Josh Perez



```
<h1> 2 + 2 = {2+2}</h1>,
```

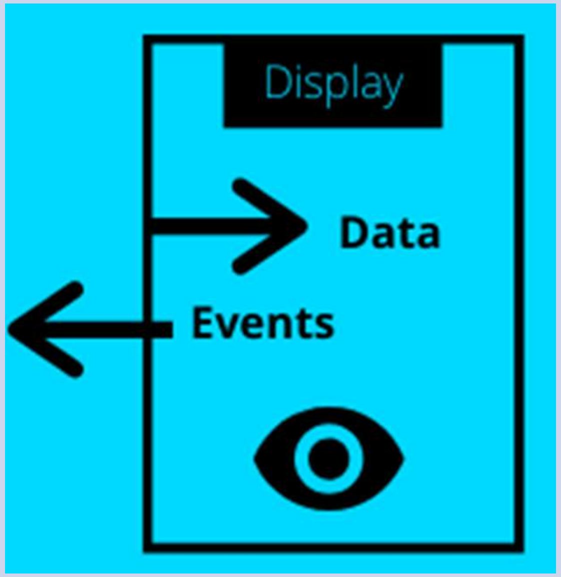
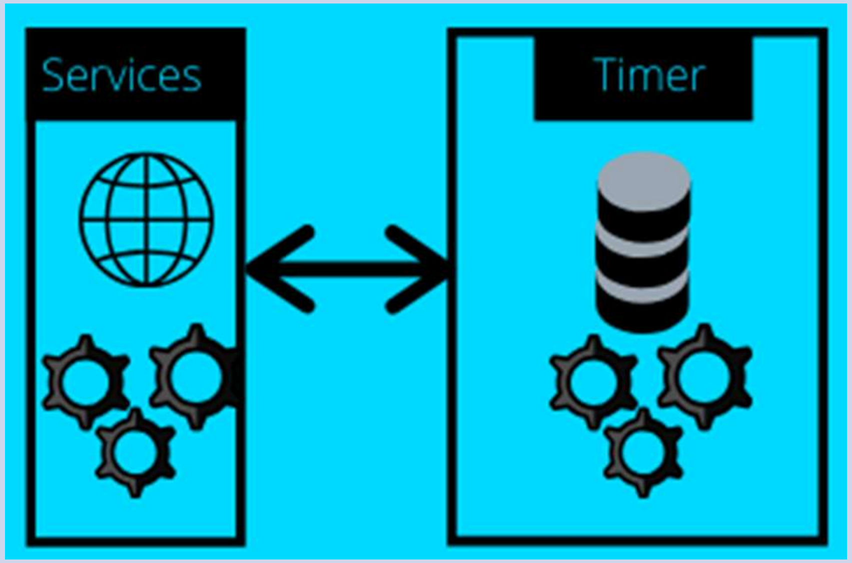
2 + 2 = 4



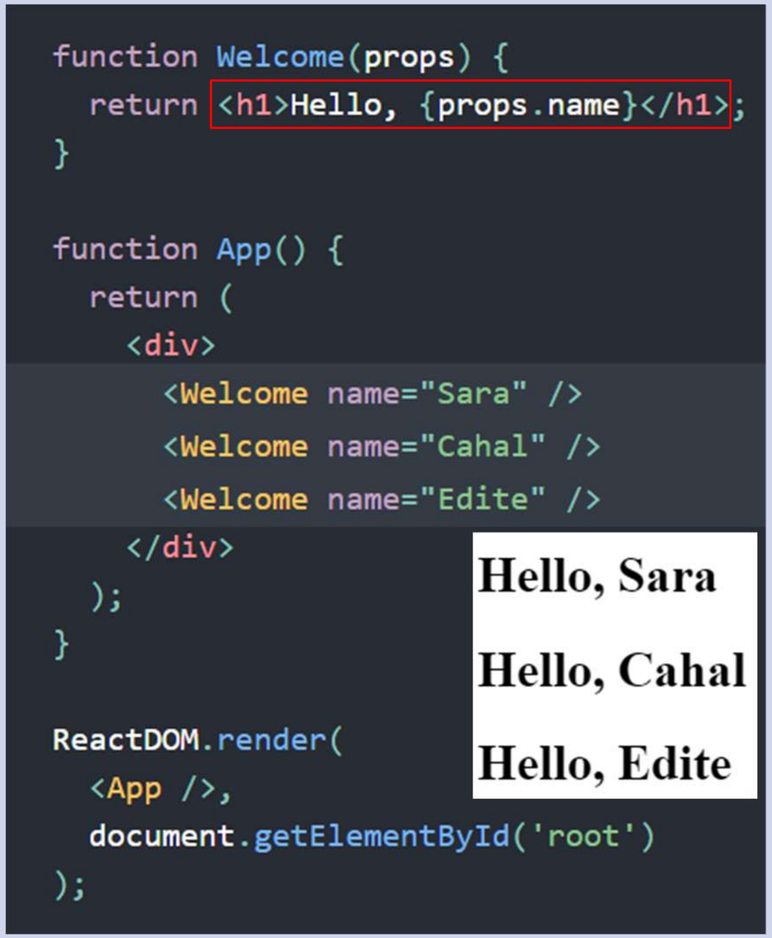
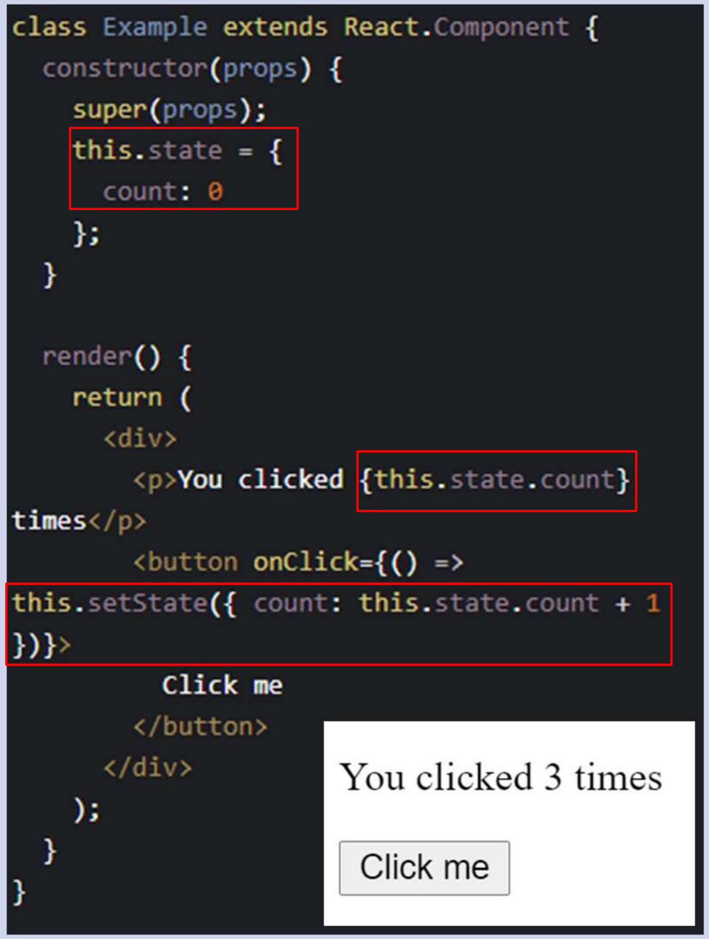
```
function formatName(user) {
  return user.firstName+ ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Hello, Perez Harper!

Stateless vs Stateful Component

Stateless	Stateful
	

Stateless vs Stateful Component

Stateless	Stateful
	

Stateless

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

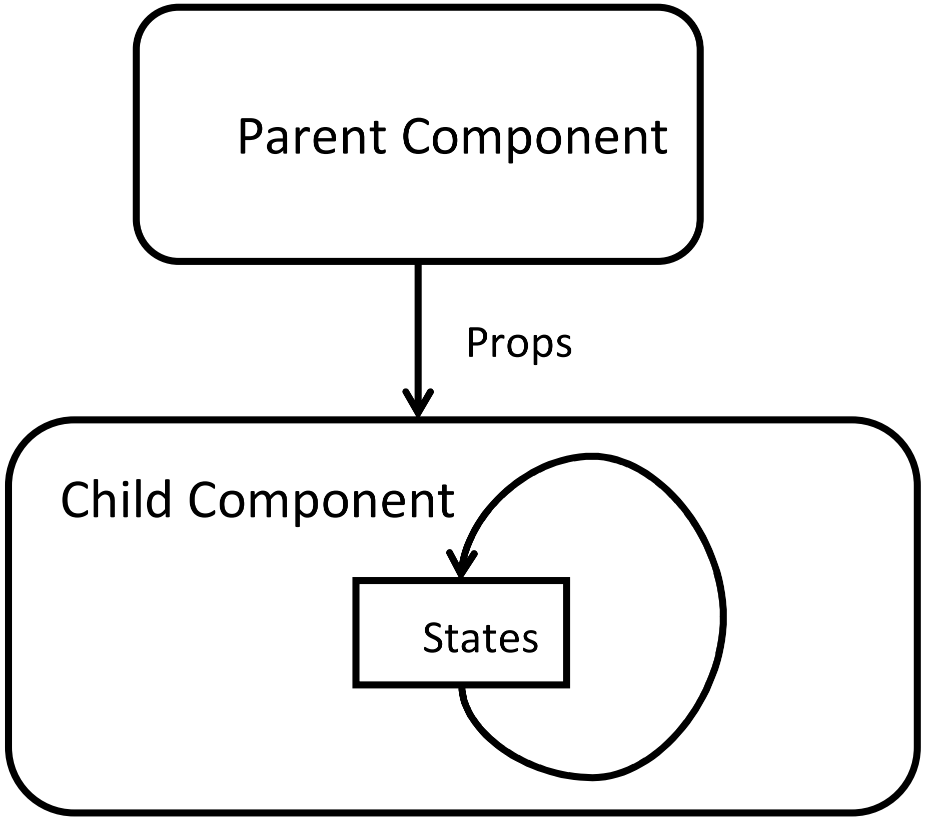
```
Hello, Sara

Hello, Cahal

Hello, Edite
```

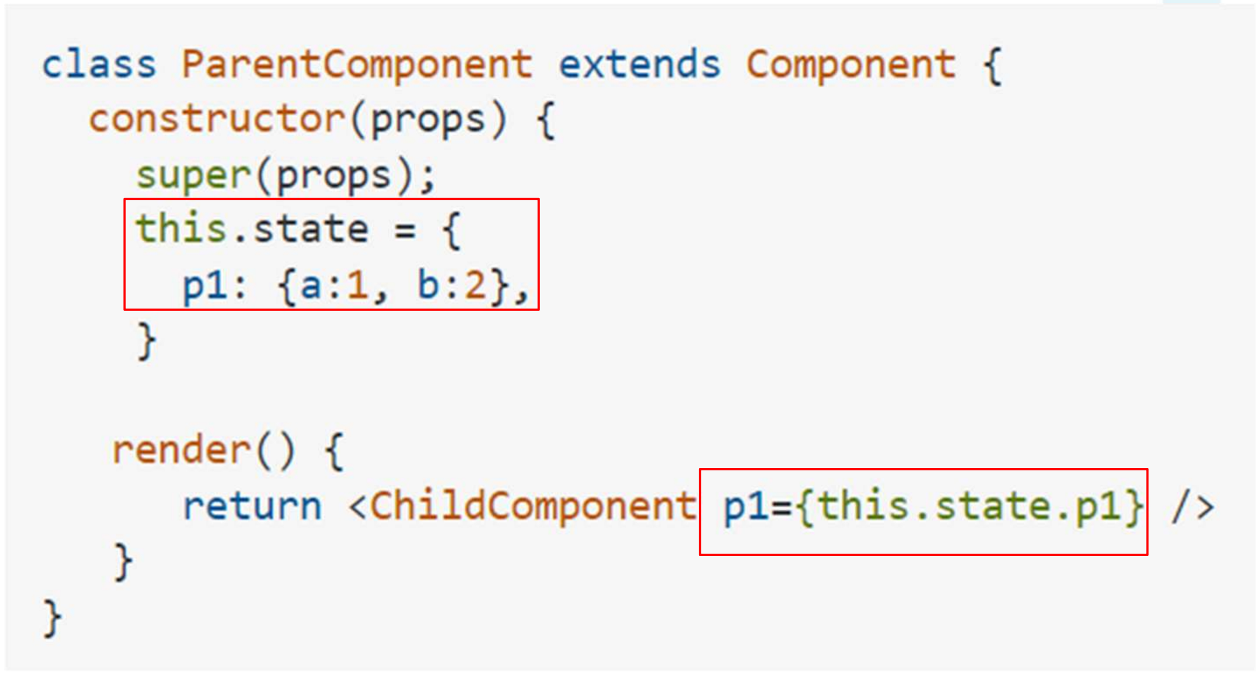
Stateful

```
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>
    );
  }
}
```

```
You clicked 3 times

Click me
```

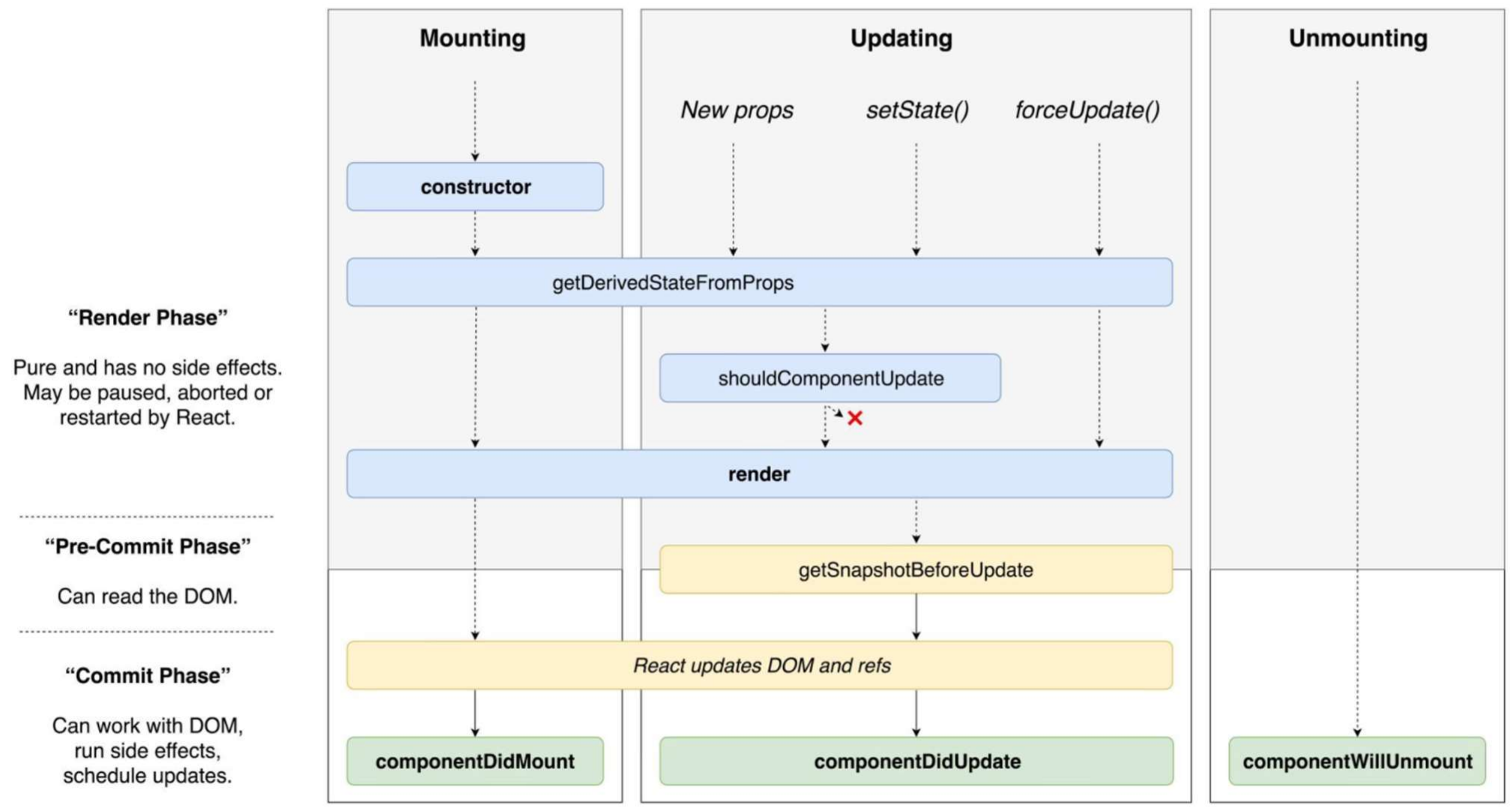
Prop is read-only





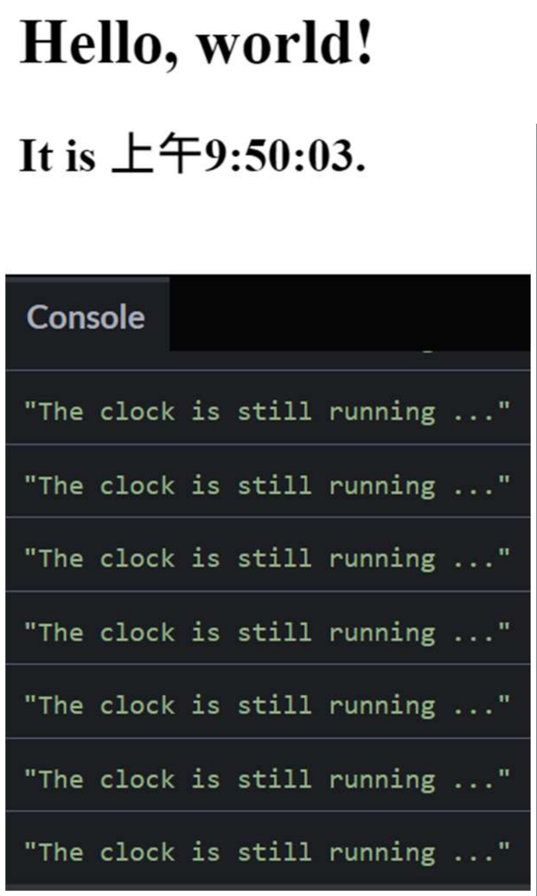
```
class ParentComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      p1: {a:1, b:2},
    }

   render() {
      return <ChildComponent p1={this.state.p1} />
   }
}
```

Component Life cycle



https://www.w3schools.com/react/react_lifecycle.asp

Performing Operations in life-cycle methods

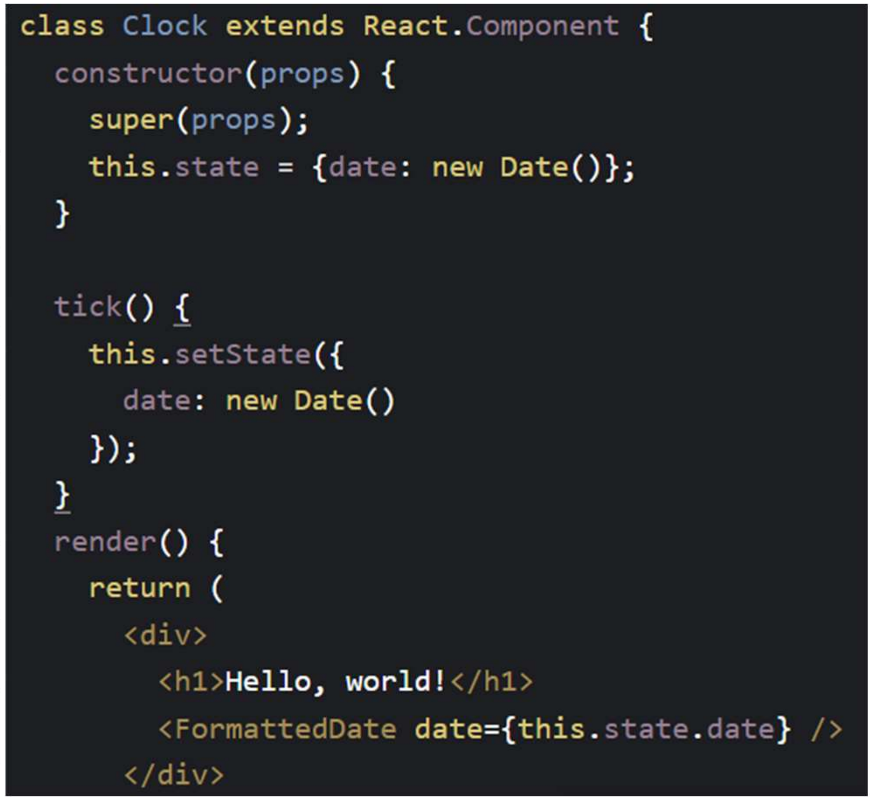


Hello, world!

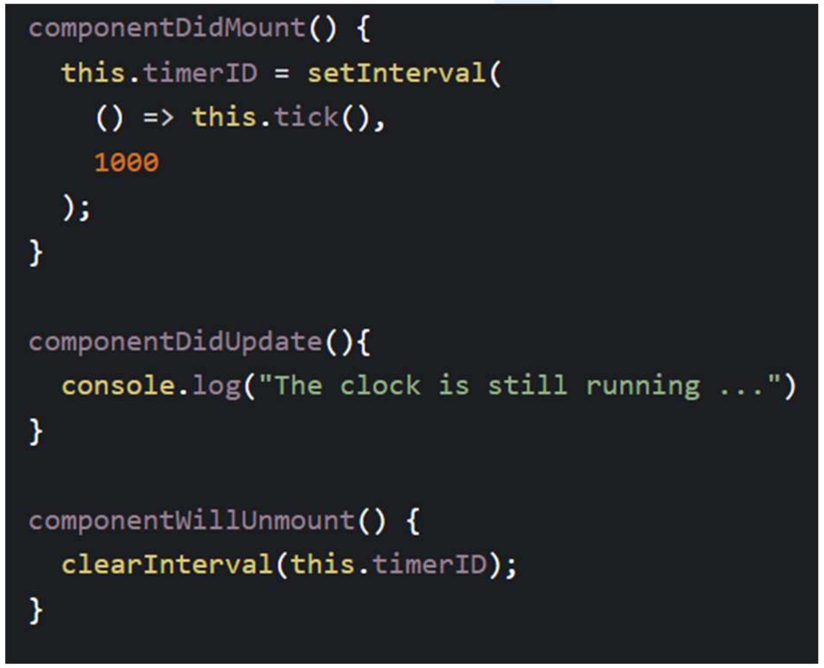
It is 上午9:50:03.

Console

```
"The clock is still running ..."
"The clock is still running ..."
"The clock is still running ..."
"The clock is still running ..."
"The clock is still running ..."
"The clock is still running ..."
"The clock is still running ..."
```



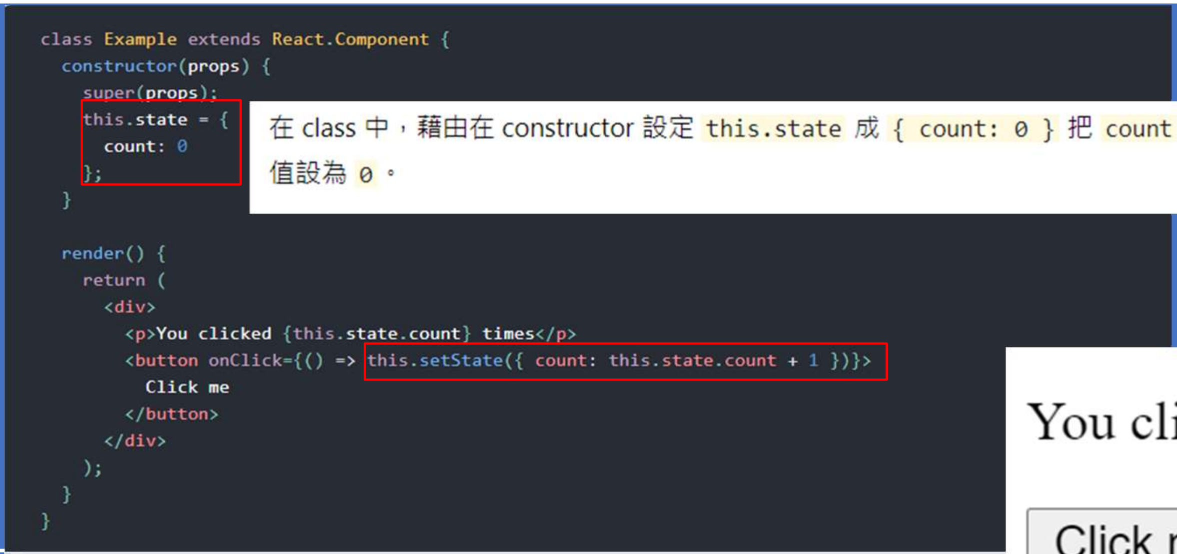
```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <FormattedDate date={this.state.date} />
      </div>
```



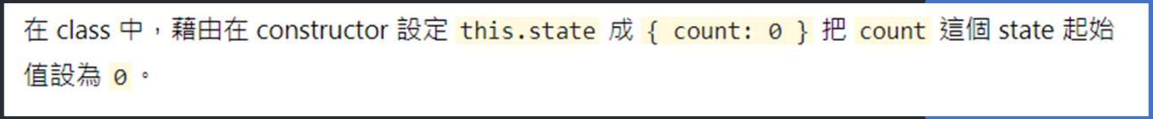
```
componentDidMount() {
  this.timerID = setInterval(
    () => this.tick(),
    1000
  );
}

componentDidUpdate(){
  console.log("The clock is still running ...")
}

componentWillUnmount() {
  clearInterval(this.timerID);
}
```

State Hook

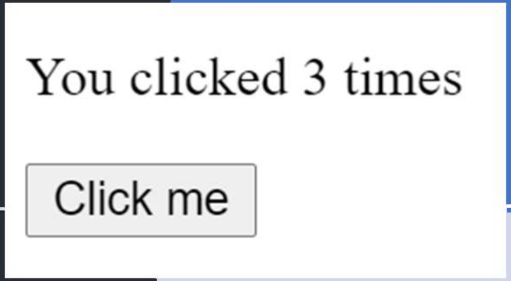
Use Class



```
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>
    );
  }
}
```

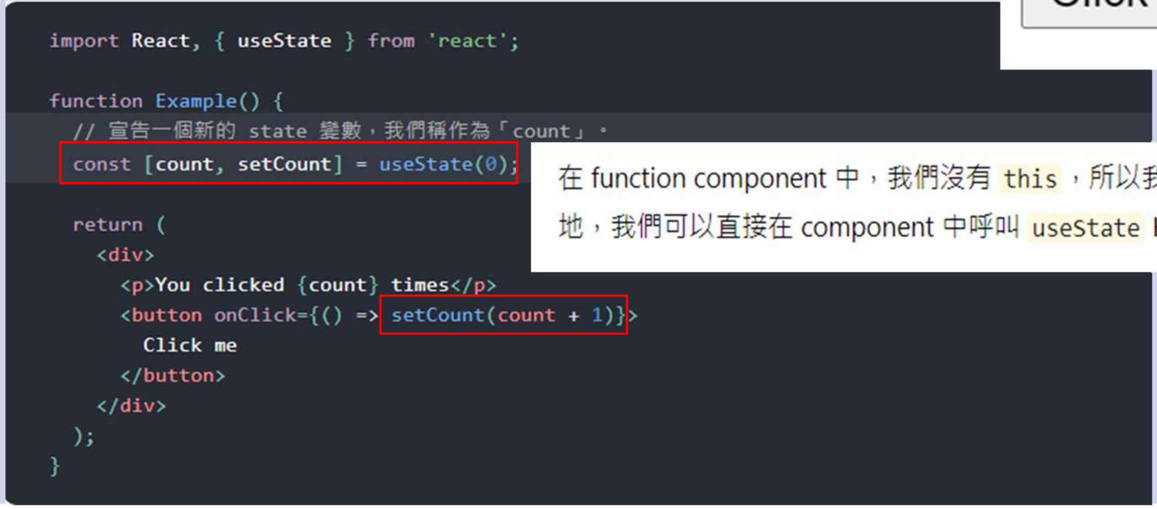


在 class 中，藉由在 constructor 設定 `this.state` 成 `{ count: 0 }` 把 `count` 這個 state 起始值設為 `0`。

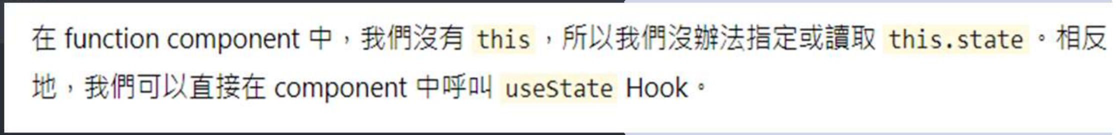


You clicked 3 times

Use Hook

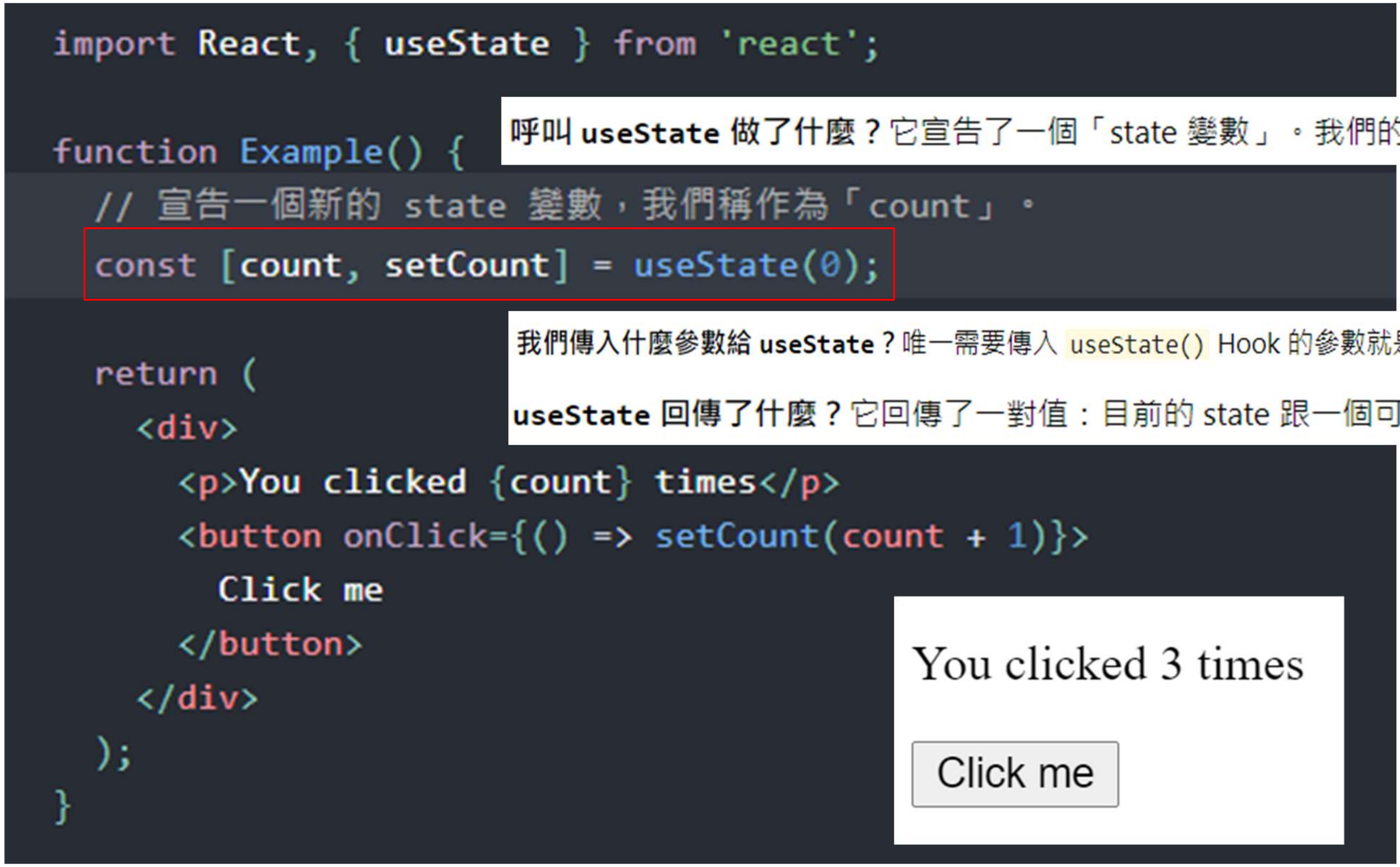


```
import React, { useState } from 'react';

function Example() {
  // 宣告一個新的 state 變數，我們稱作為「count」。
  const [count, setCount] = useState(0);

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



在 function component 中，我們沒有 `this`，所以我們沒辦法指定或讀取 `this.state`。相反地，我們可以直接在 component 中呼叫 `useState` Hook。

State Hooks

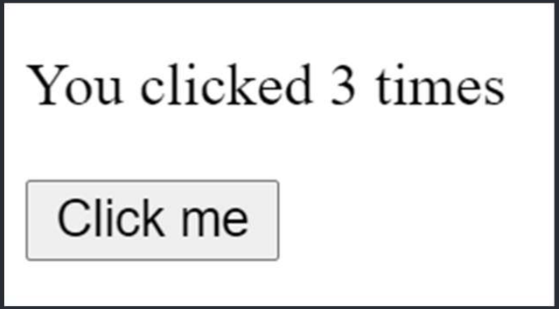


```
import React, { useState } from 'react';

function Example() {
  // 宣告一個新的 state 變數，我們稱作為「count」。
  const [count, setCount] = useState(0);

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

呼叫 **useState** 做了什麼？它宣告了一個「state 變數」。我們的變數叫做 `count`

我們傳入什麼參數給 **useState**？唯一需要傳入 `useState()` Hook 的參數就是 state 的起始

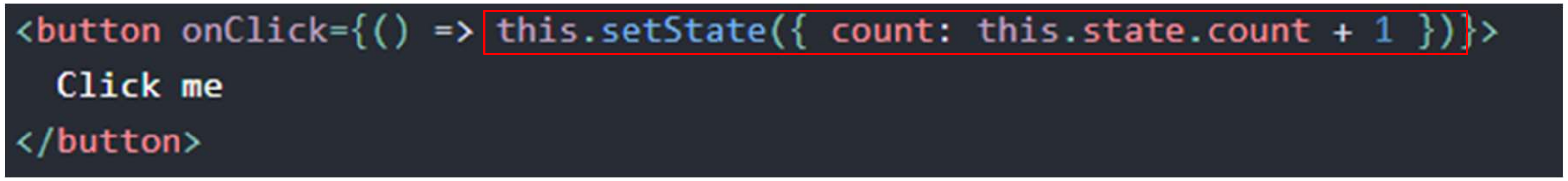
useState 回傳了什麼？它回傳了一對值：目前的 state 跟一個可以更新 state 的 function



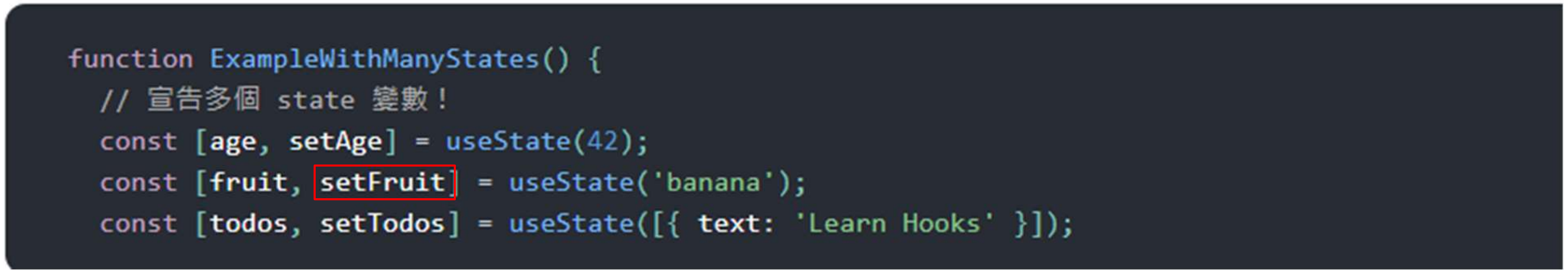
You clicked 3 times

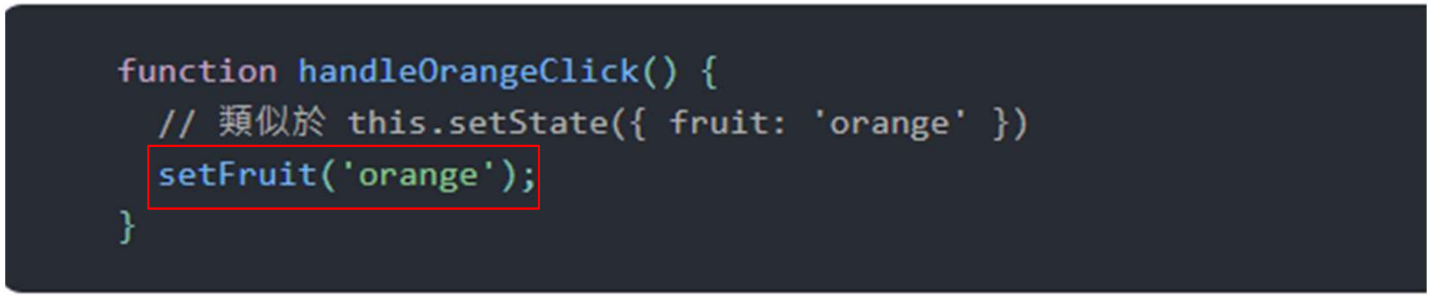
State Hooks

MobileDrive

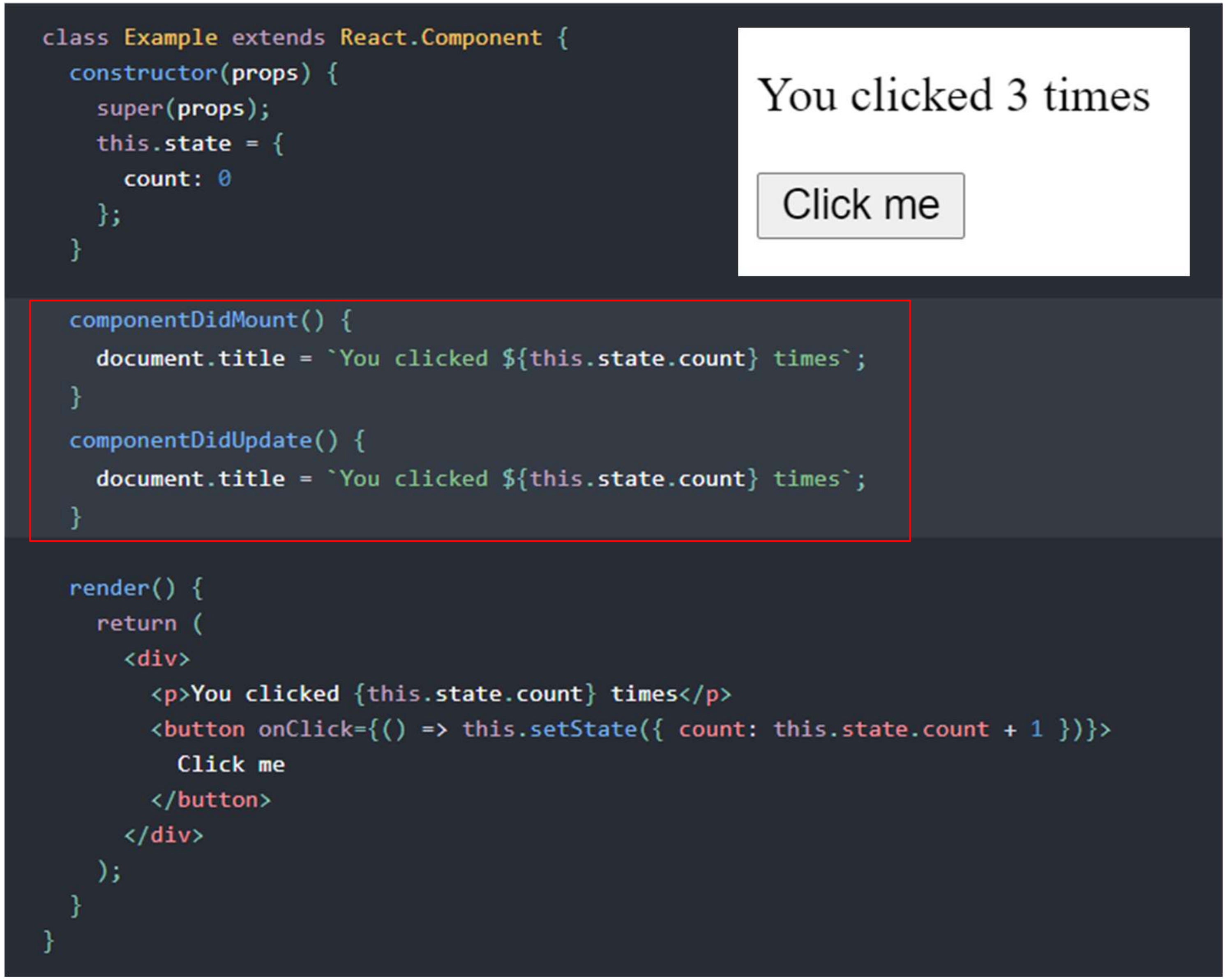

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>
  Click me
</button>
```

把 state 變數宣告成一對 `[something, setSomething]` 同時也很便利，因為如果想要使用超過一個 state 變數，這能讓我們對不同的 state 變數有不同的命名


```
function ExampleWithManyStates() {
  // 宣告多個 state 變數！
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```


```
function handleOrangeClick() {
  // 類似於 this.setState({ fruit: 'orange' })
  setFruit('orange');
}
```

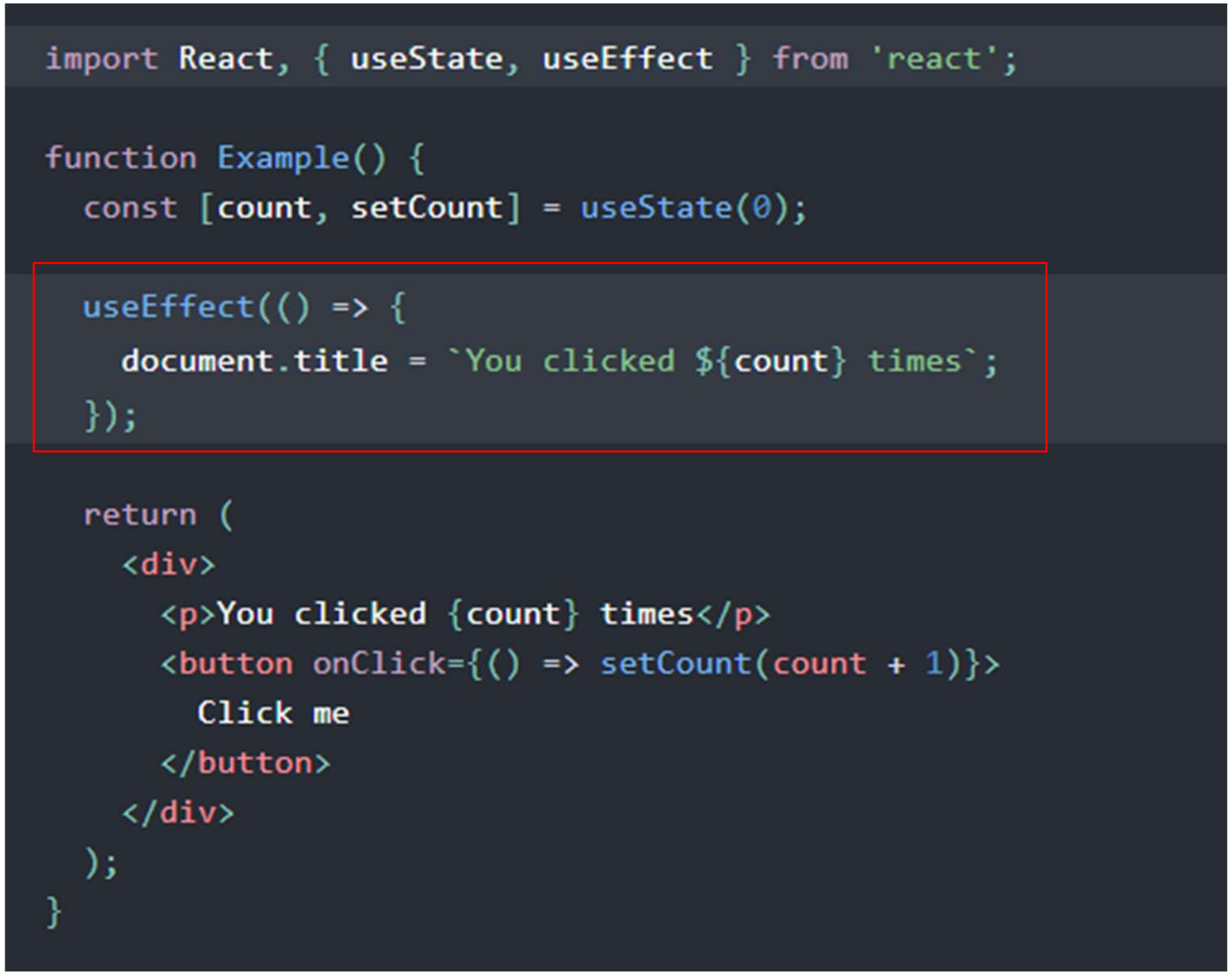
Effect Hook



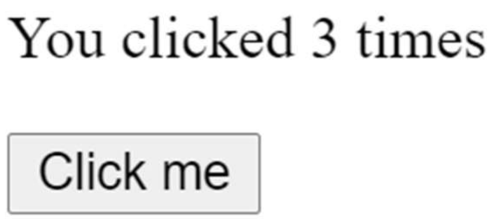
```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

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

You clicked 3 times

Effect Hook



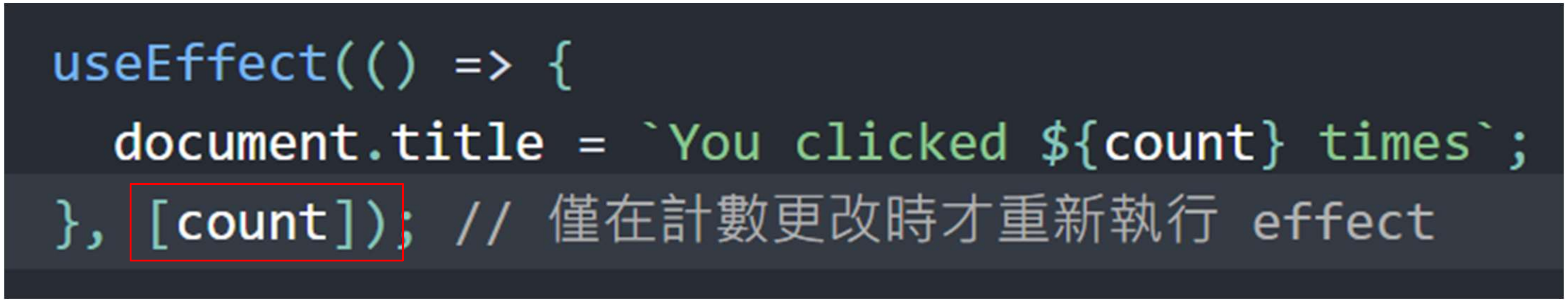
```
import React, { useState, useEffect } from 'react';

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

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



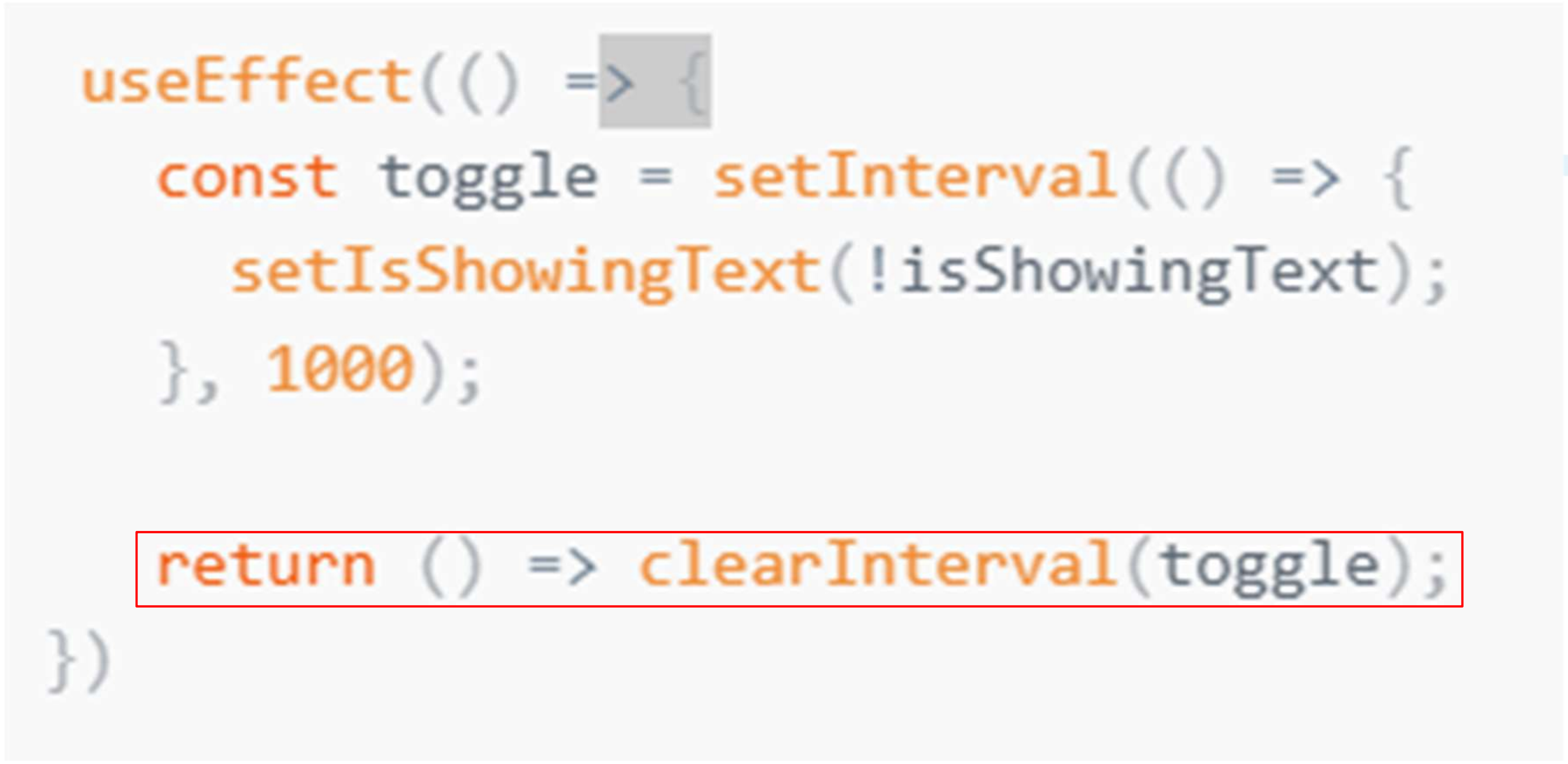
```
You clicked 3 times
```

每次 **render** 後都會執行 **useEffect**

Effect Hook


```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]); // 僅在計數更改時才重新執行 effect
```

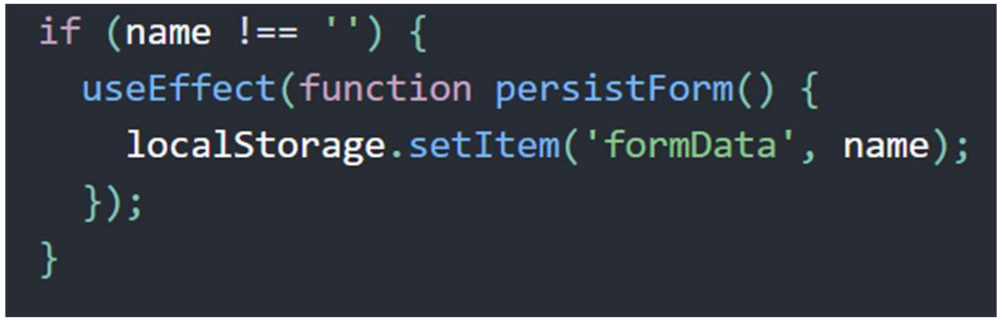
Clear effect


```
useEffect(() => {
  const toggle = setInterval(() => {
    setIsShowingText(!isShowingText);
  }, 1000);

  return () => clearInterval(toggle);
})
```

Rules of using hooks

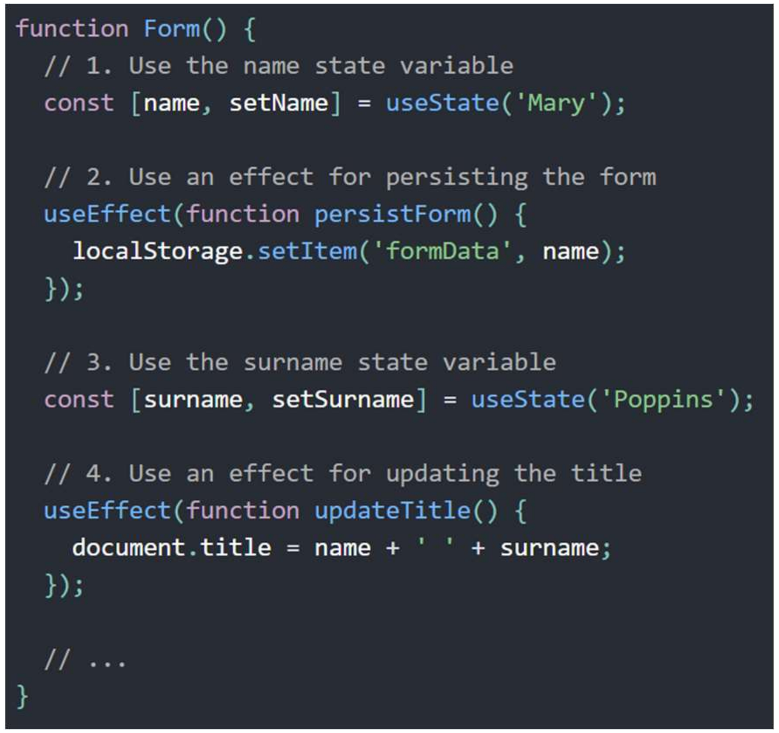
- **ONLY** use hooks *function component*, or *custom hooks*
- **ONLY** use hooks at the *top level*, **NOT** in loops, conditions, or nested function

✕



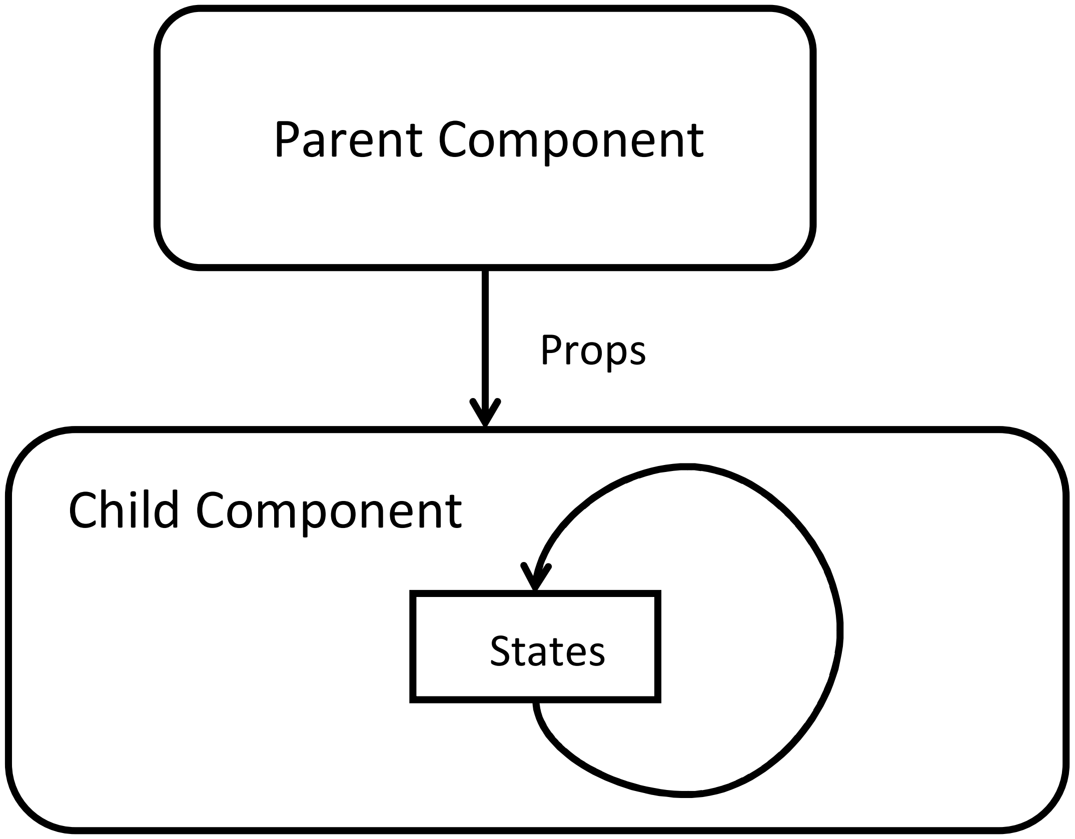
```
if (name !== '') {
  useEffect(function persistForm() {
    localStorage.setItem('formData', name);
  });
}
```

✓



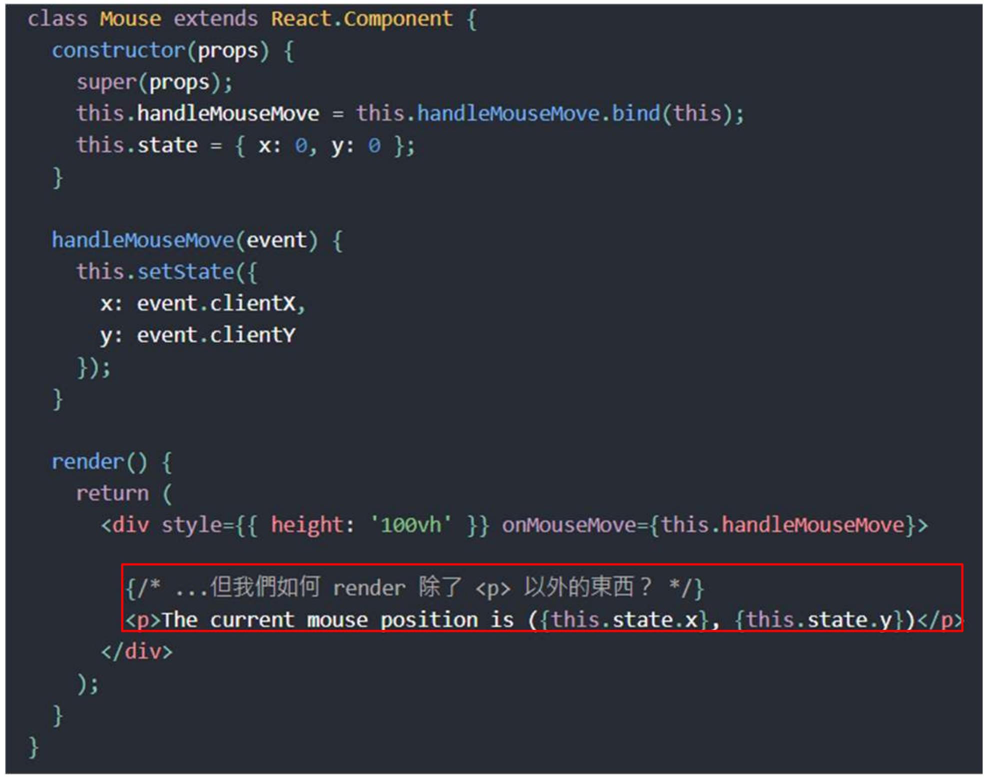
```
function Form() {
  // 1. Use the name state variable
  const [name, setName] = useState('Mary');

  // 2. Use an effect for persisting the form
  useEffect(function persistForm() {
    localStorage.setItem('formData', name);
  });

  // 3. Use the surname state variable
  const [surname, setSurname] = useState('Poppins');

  // 4. Use an effect for updating the title
  useEffect(function updateTitle() {
    document.title = name + ' ' + surname;
  });

  // ...
}
```

State, Prop, example with hook

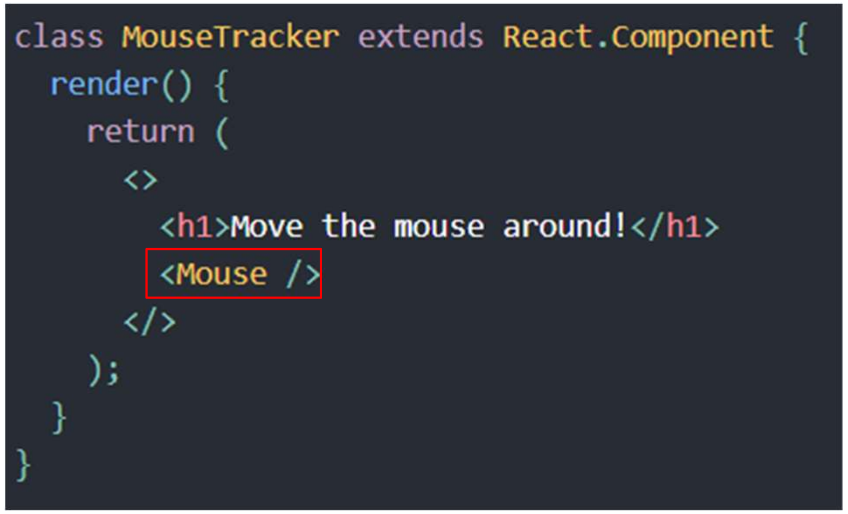


Demo link

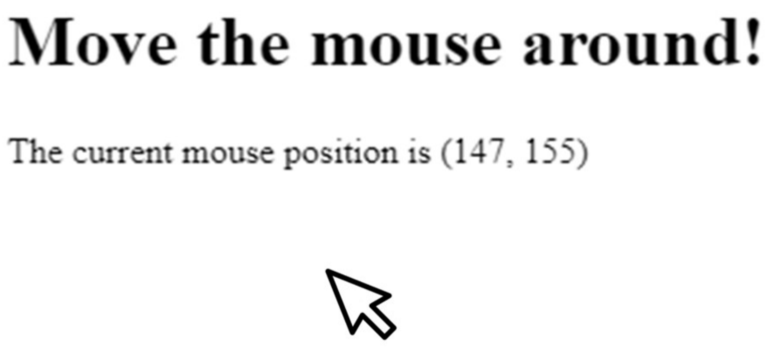
Sharing stateful logic



```
class Mouse extends React.Component {
  constructor(props) {
    super(props);
    this.handleMouseMove = this.handleMouseMove.bind(this);
    this.state = { x: 0, y: 0 };
  }

  handleMouseMove(event) {
    this.setState({
      x: event.clientX,
      y: event.clientY
    });
  }

  render() {
    return (
      <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>

        {/* ...但我們如何 render 除了 <p> 以外的東西？ */}
        <p>The current mouse position is ({this.state.x}, {this.state.y})</p>
      </div>
    );
  }
}
```

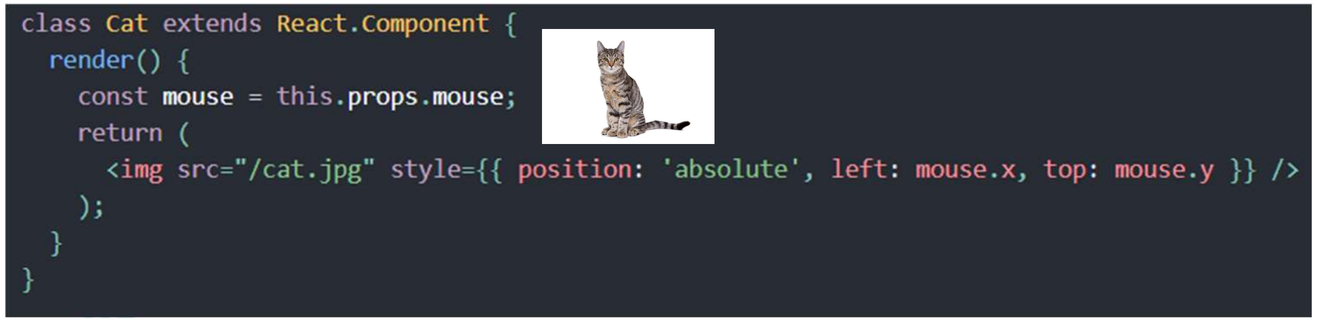


```
class MouseTracker extends React.Component {
  render() {
    return (
      <>
        <h1>Move the mouse around!</h1>
        <Mouse />
      </>
    );
  }
}
```



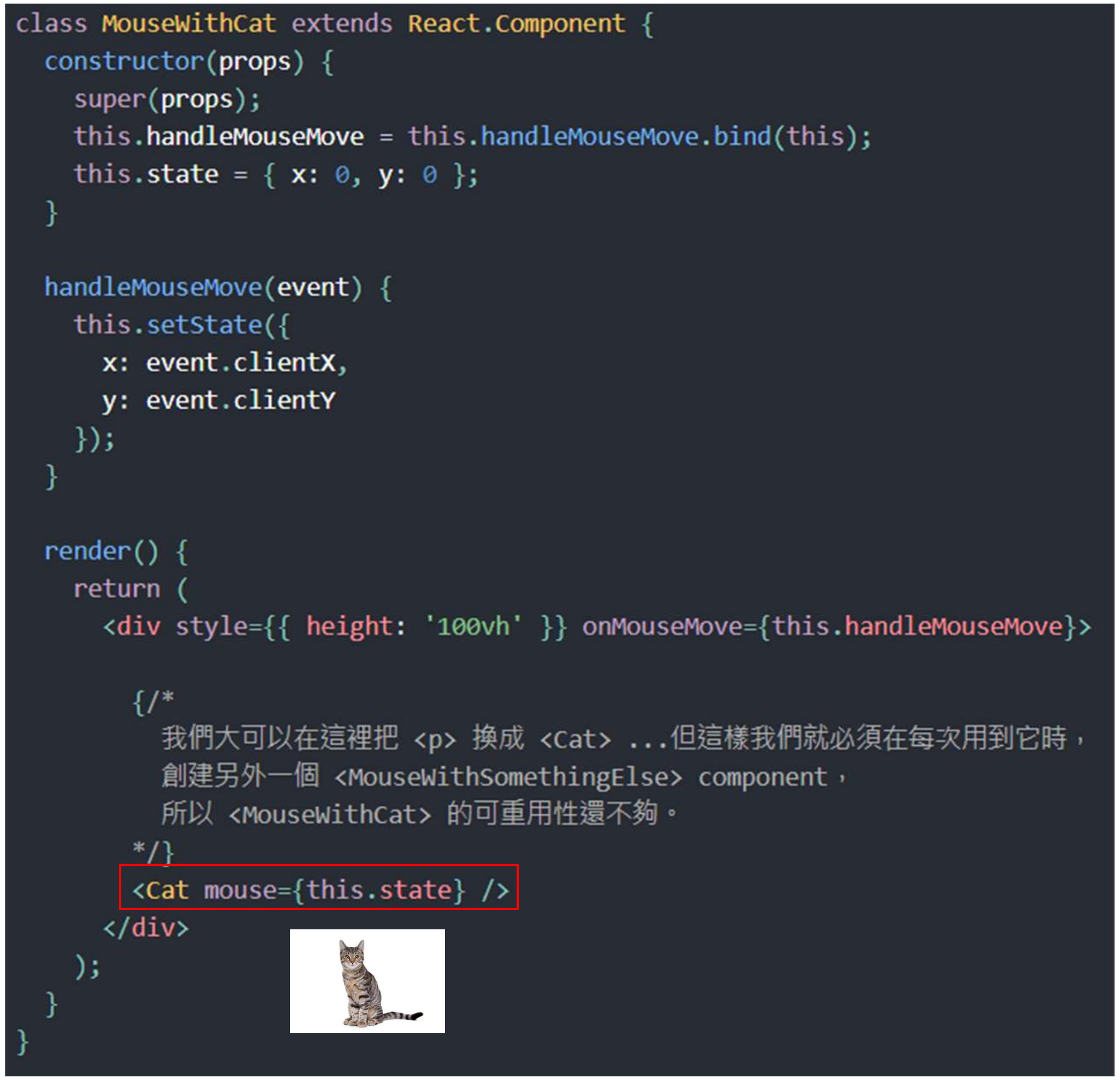
Move the mouse around!

The current mouse position is (147, 155)

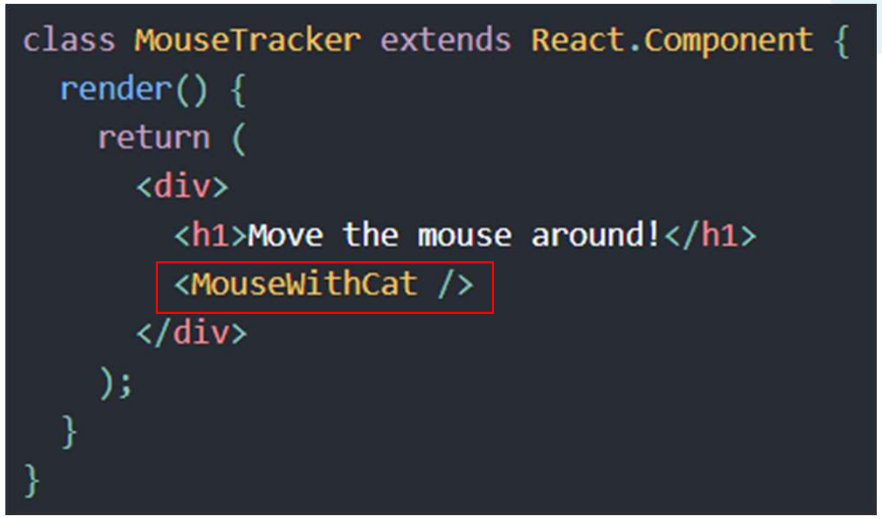


```
class Cat extends React.Component {
  render() {
    const mouse = this.props.mouse;
    return (
      <img src="/cat.jpg" style={{ position: 'absolute', left: mouse.x, top: mouse.y }} />
    );
  }
}
```

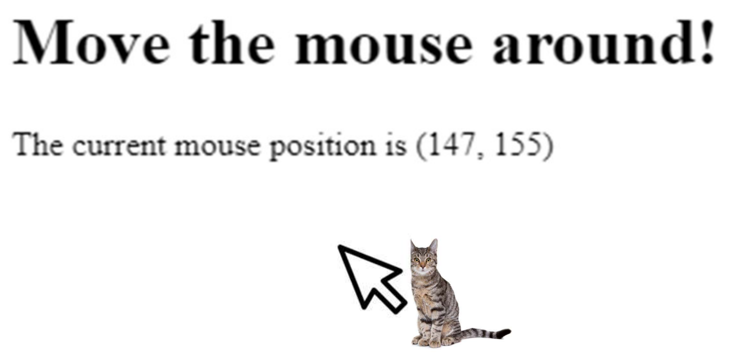
Sharing stateful logic: fist attempt



```
class MouseWithCat extends React.Component {
  constructor(props) {
    super(props);
    this.handleMouseMove = this.handleMouseMove.bind(this);
    this.state = { x: 0, y: 0 };
  }

  handleMouseMove(event) {
    this.setState({
      x: event.clientX,
      y: event.clientY
    });
  }

  render() {
    return (
      <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>

        {/*
          我們大可以在這裡把 <p> 換成 <Cat> ...但這樣我們就必須在每次用到它時，
          創建另外一個 <MouseWithSomethingElse> component，
          所以 <MouseWithCat> 的可重用性還不夠。
        */}
        <Cat mouse={this.state} />
      </div>
    );
  }
}
```



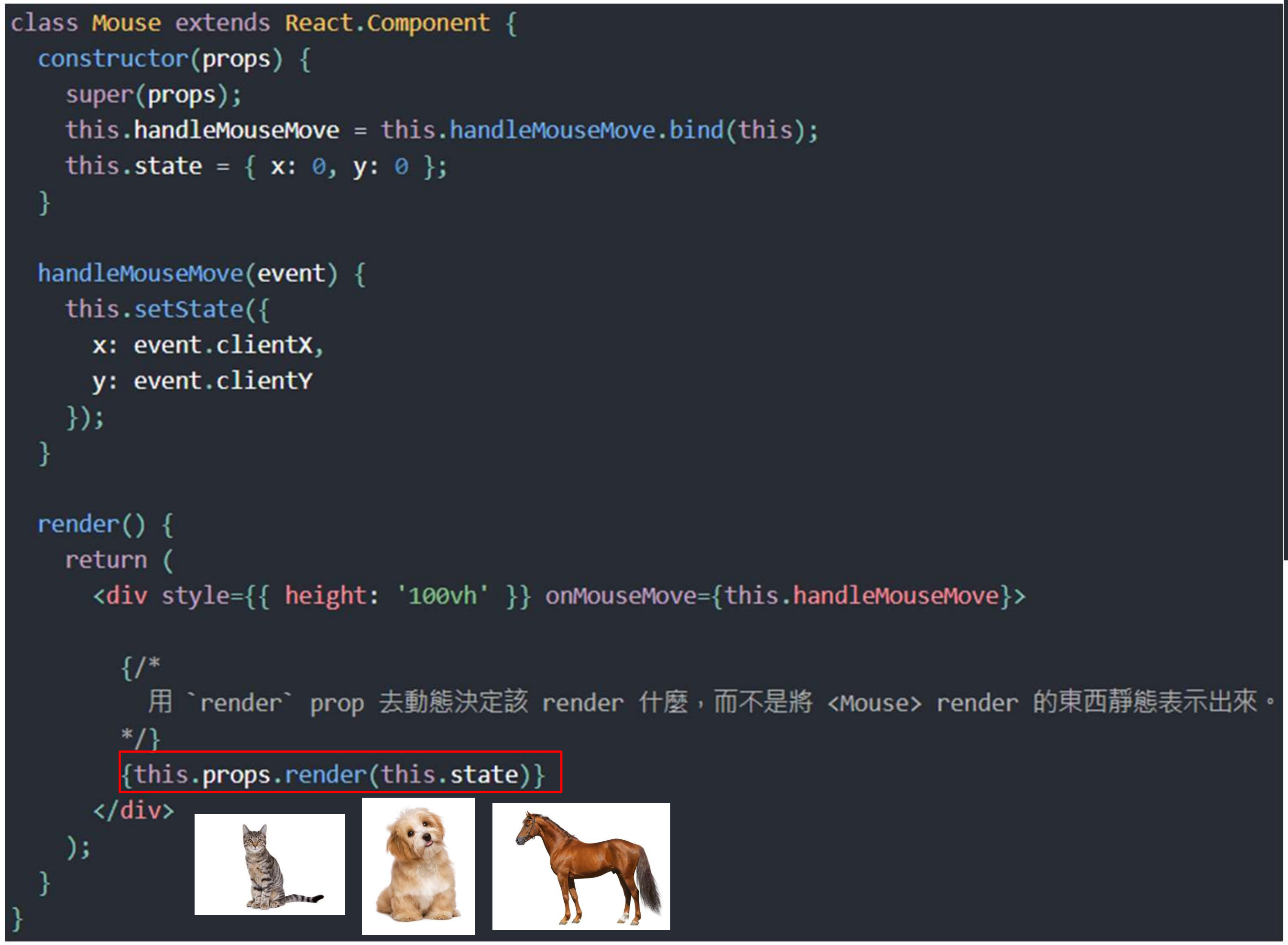
```
class MouseTracker extends React.Component {
  render() {
    return (
      <div>
        <h1>Move the mouse around!</h1>
        <MouseWithCat />
      </div>
    );
  }
}
```



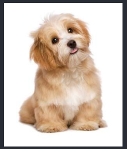
Move the mouse around!

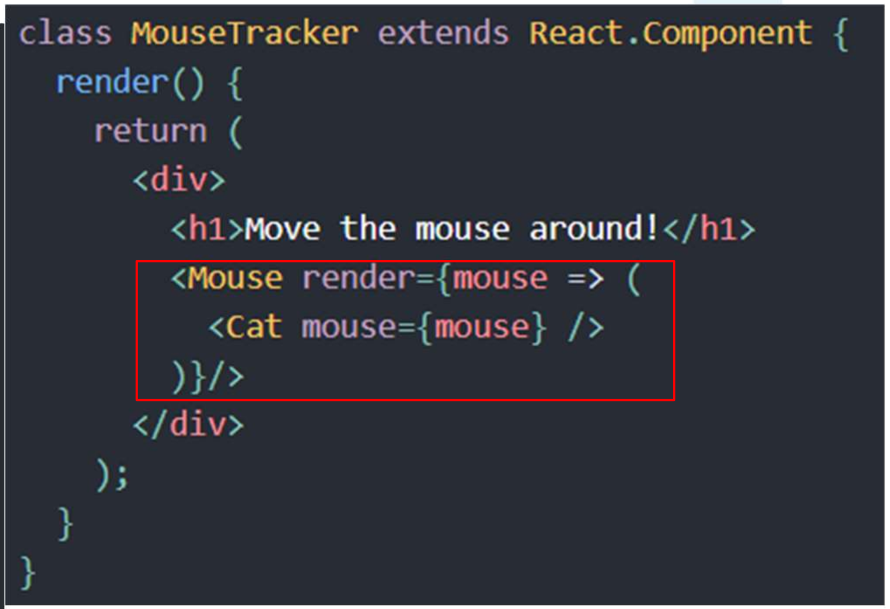
The current mouse position is (147, 155)

Render prop

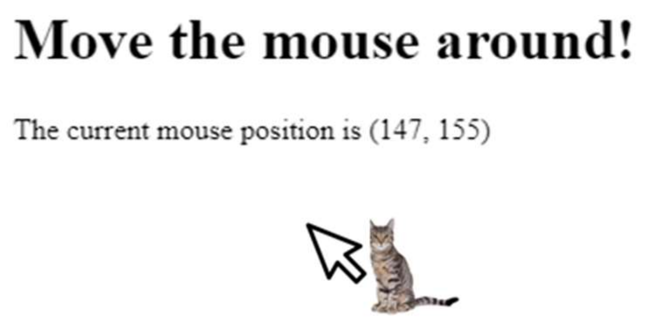


```
class Mouse extends React.Component {
  constructor(props) {
    super(props);
    this.handleMouseMove = this.handleMouseMove.bind(this);
    this.state = { x: 0, y: 0 };
  }

  handleMouseMove(event) {
    this.setState({
      x: event.clientX,
      y: event.clientY
    });
  }

  render() {
    return (
      <div style={{ height: '100vh' }} onMouseMove={this.handleMouseMove}>

        {/*
          用 `render` prop 去動態決定該 render 什麼，而不是將 <Mouse> render 的東西靜態表示出來。
        */}
        {this.props.render(this.state)}
      </div>
    );
  }
}
```





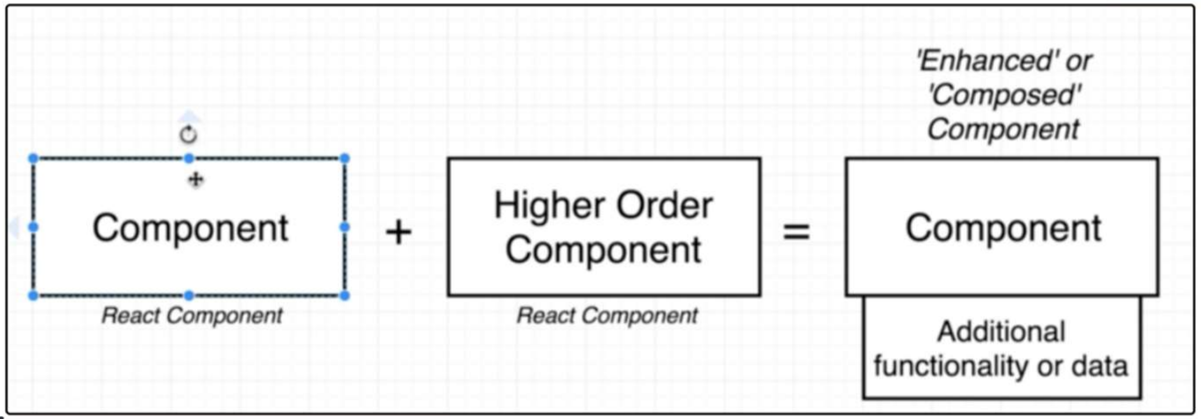
```
class MouseTracker extends React.Component {
  render() {
    return (
      <div>
        <h1>Move the mouse around!</h1>
        <Mouse render={mouse => (
          <Cat mouse={mouse} />
        )}/>
      </div>
    );
  }
}
```

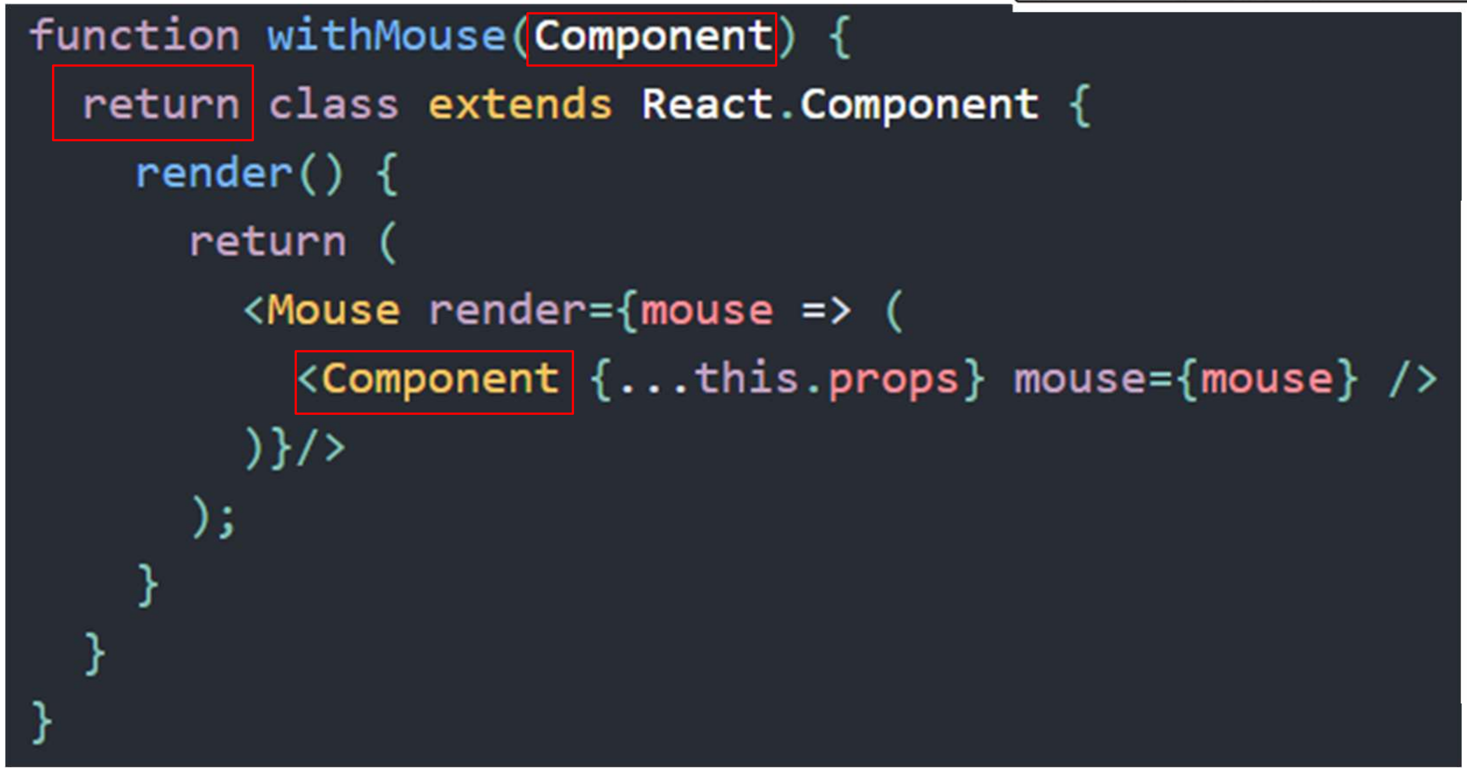


Move the mouse around!

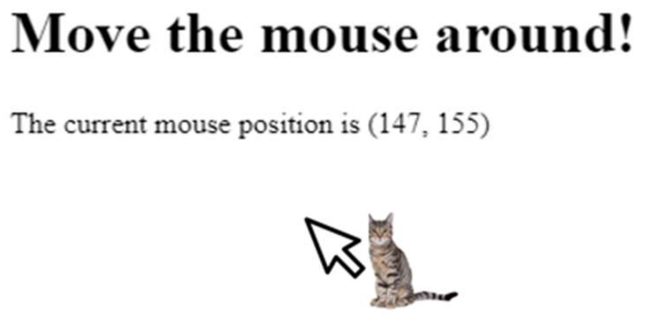
The current mouse position is (147, 155)

Higher order component





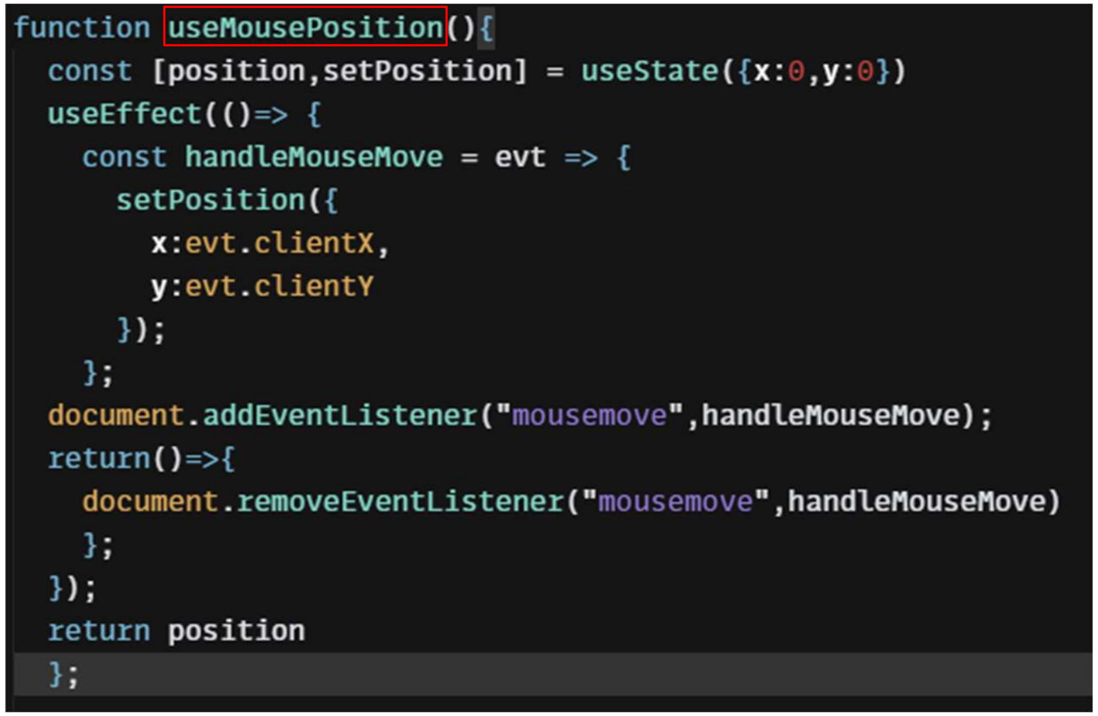
```
function withMouse(Component) {
  return class extends React.Component {
    render() {
      return (
        <Mouse render={mouse => (
          <Component {...this.props} mouse={mouse} />
        )}/>
      );
    }
  }
}
```



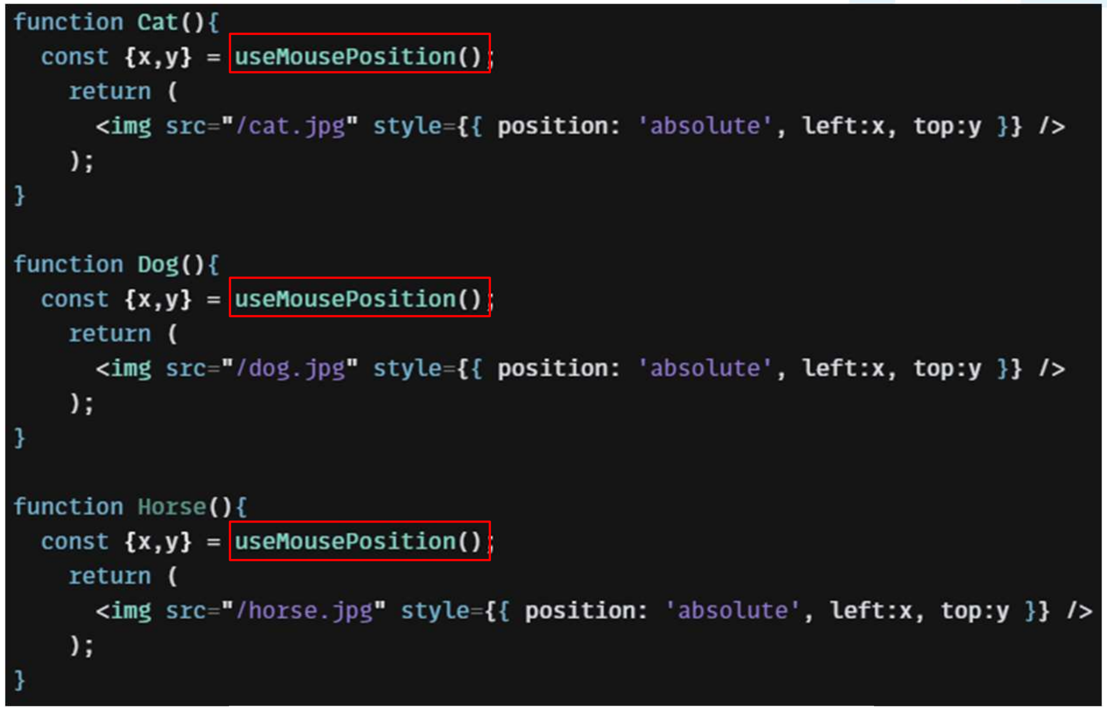
Move the mouse around!

The current mouse position is (147, 155)

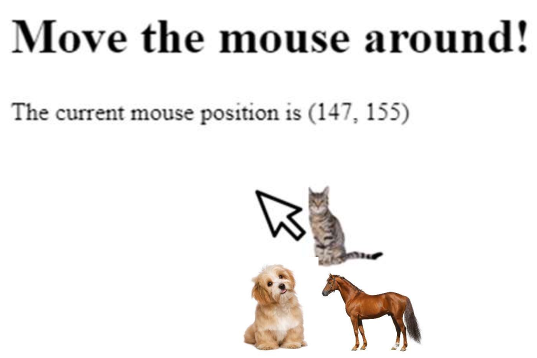
Custom Hook



```
function useMousePosition(){
  const [position,setPosition] = useState({x:0,y:0})
  useEffect(()=> {
    const handleMouseMove = evt => {
      setPosition({
        x:evt.clientX,
        y:evt.clientY
      });
    };
  document.addEventListener("mousemove",handleMouseMove);
  return()=>{
    document.removeEventListener("mousemove",handleMouseMove)
    };
  });
  return position
  };
```



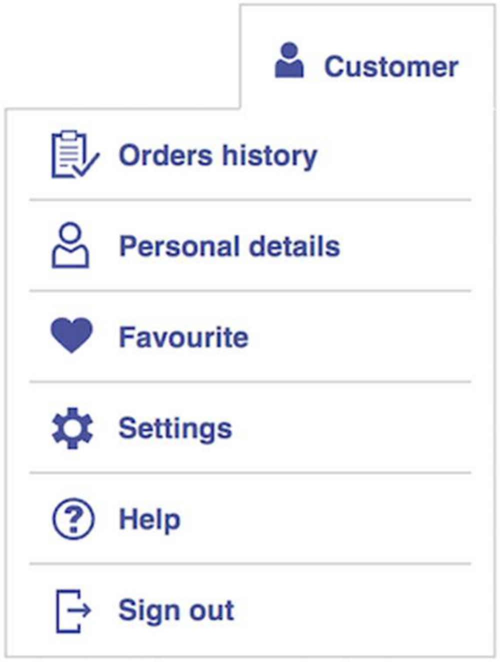
```
function Cat(){
  const {x,y} = useMousePosition();
    return (
      <img src="/cat.jpg" style={{ position: 'absolute', left:x, top:y }} />
    );
}

function Dog(){
  const {x,y} = useMousePosition();
    return (
      <img src="/dog.jpg" style={{ position: 'absolute', left:x, top:y }} />
    );
}

function Horse(){
  const {x,y} = useMousePosition();
    return (
      <img src="/horse.jpg" style={{ position: 'absolute', left:x, top:y }} />
    );
}
```

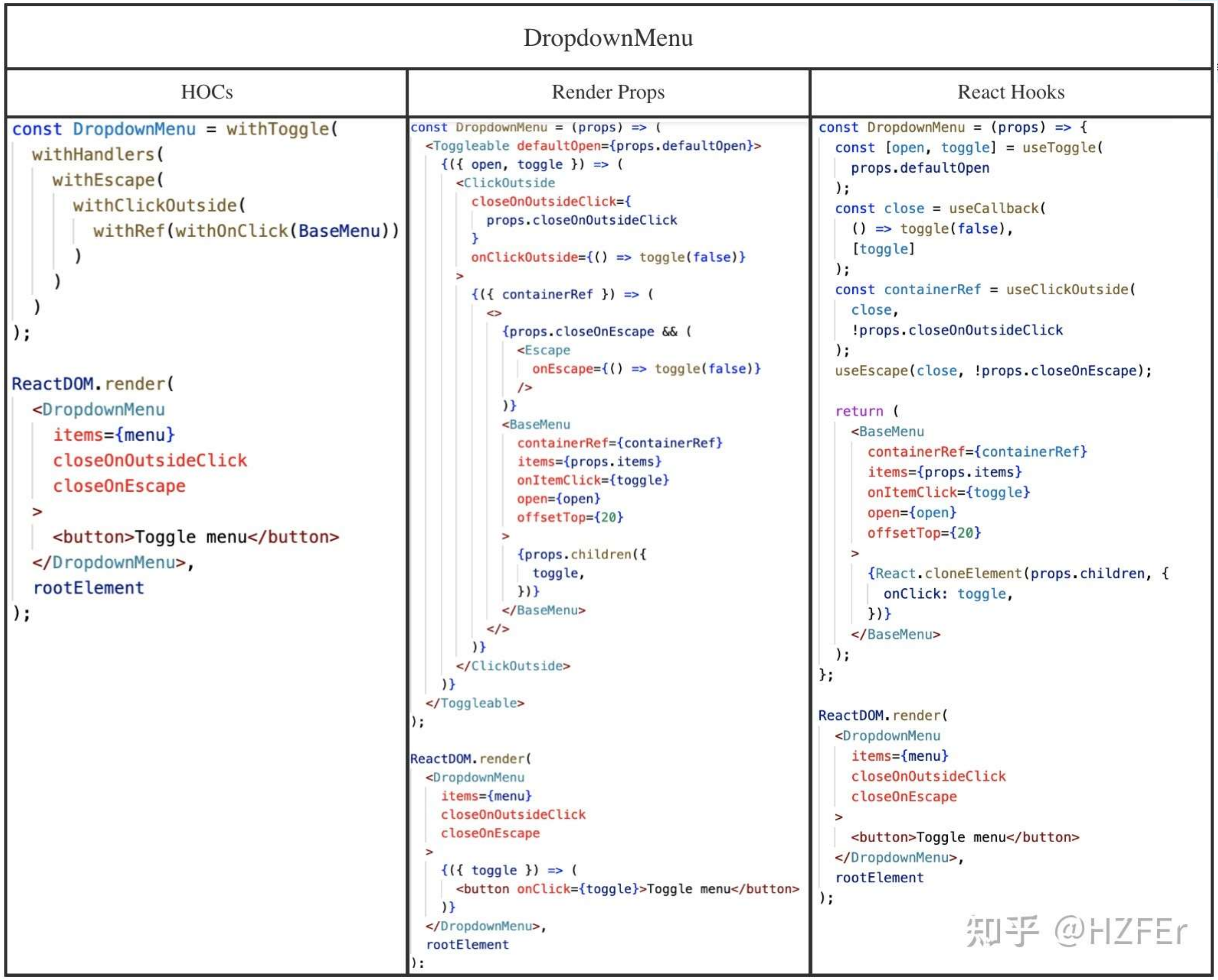


Move the mouse around!

The current mouse position is (147, 155)

Render vs HOC vs Hook





DropdownMenu

HOCs

```
const DropdownMenu = withToggle(
  withHandlers(
    withEscape(
      withClickOutside(
        withRef(withOnClick(BaseMenu))
      )
    )
  )
);

ReactDOM.render(
  <DropdownMenu
    items={menu}
    closeOnOutsideClick
    closeOnEscape
  >
    <button>Toggle menu</button>
  </DropdownMenu>,
  rootElement
);
```

Render Props

```
const DropdownMenu = (props) => (
  <Toggleable defaultOpen={props.defaultOpen}>
    {({ open, toggle }) => (
      <ClickOutside
        closeOnOutsideClick={
          props.closeOnOutsideClick
        }
        onClickOutside={() => toggle(false)}
      >
        {({ containerRef }) => (
          <>
            {props.closeOnEscape && (
              <Escape
                onEscape={() => toggle(false)}
              />
            )}
            <BaseMenu
              containerRef={containerRef}
              items={props.items}
              onItemClick={toggle}
              open={open}
              offsetTop={20}
            >
              {props.children({
                toggle,
              })}
            </BaseMenu>
          </>
        )}
      </ClickOutside>
    )}
  </Toggleable>
);

ReactDOM.render(
  <DropdownMenu
    items={menu}
    closeOnOutsideClick
    closeOnEscape
  >
    {({ toggle }) => (
      <button onClick={toggle}>Toggle menu</button>
    )}
  </DropdownMenu>,
  rootElement
);
```

React Hooks

```
const DropdownMenu = (props) => {
  const [open, toggle] = useToggle(
    props.defaultOpen
  );
  const close = useCallback(
    () => toggle(false),
    [toggle]
  );
  const containerRef = useClickOutside(
    close,
    !props.closeOnOutsideClick
  );
  useEscape(close, !props.closeOnEscape);

  return (
    <BaseMenu
      containerRef={containerRef}
      items={props.items}
      onItemClick={toggle}
      open={open}
      offsetTop={20}
    >
      {React.cloneElement(props.children, {
        onClick: toggle,
      })}
    </BaseMenu>
  );
};

ReactDOM.render(
  <DropdownMenu
    items={menu}
    closeOnOutsideClick
    closeOnEscape
  >
    <button>Toggle menu</button>
  </DropdownMenu>,
  rootElement
);
```

Render vs HOC vs Hook

	Higher Order Component	Render Prop	Hook
Readibility	Poor	Fair	Good
Reusability	Good	Fair	Good
Customization and Usage	Poor	Good	Good
Debugging	Poor	Good	Good
Testability	Good	Good	Fair
Performance	Good	Good	Fair

Reference