

# the Master Course

{C0DENATION}

# Working with State & Methods

{C0DENATION}

# Learning Objectives

**To understand what state is and how we can work with it.**

**To be able to use the useState hook**

# React.js

What does the word **state** mean to you?

# React.js

## State

As React is a JavaScript library for creating **user interfaces** when we talk about state in react, we are talking about the state of the user interface.

**There is a starting state, which you will describe in code.**

**When you buy a new car, it has a starting state. It's shiny. It's clean, etc.**

**When your app first loads, it too will have a starting state. It's up to you what this is.**

## State can be changed.

If you crash your brand new car, you will change the state of it. Maybe it's now got some scratches. Or missing a wheel.

Your app's user interface will change its state as users interact with it. They could like a photo, or leave a comment etc.

**We always need to keep track of the current state of things.**

**HMRC will always want to be kept updated with information about your current salary, so they know how much tax you should be paying. Once you update them, they will change their records.**

**Similarly, Your app needs to keep track of the current state of things, so it knows how it should display the user interface.**



**State is used/stored inside **class**  
**components** as an object or as a **hook****

**But we should use state with care!**

**Imagine if you had lots of different  
components all tracking their own state  
but not talking to each other. Things  
could get very messy.**

**It's good practice to keep your state in your root component (as much as you can at least) and then pass state around through props.**

**As we build more and more complicated apps, this may not always be possible.**

# React.js

**State** is very special. Behind the scenes, React is keeping an eye on it. Every time React detects the state has been changed, this triggers a **re-render (reload)** of components which have changed.

# React.js

**So every time the state changes.  
The render method is executed and  
any changes made to the UI are  
shown.**

```
class App extends React.Component {  
  
  render() {  
    return (  
      <div>  
        <h1>I'm a class component</h1>  
        <Person />  
        <Person />  
        <Person />  
      </div>  
    );  
  }  
}
```



```
class App extends React.Component {  
  state = {  
    persons: [  
      {name: "Dan", age: 34},  
      {name: "Ben", age: 21}  
    ]  
  }  
  
  render() {  
    return (  
      <div>  
        <h1>I'm a class component</h1>  
        <Person />  
        <Person />  
      </div>  
    );  
  }  
}
```

**In the state I have a persons property which holds an array (made up of two objects).**



```
class App extends React.Component {  
  state = {  
    persons: [  
      {name: "Dan", age: 34},  
      {name: "Ben", age: 21}  
    ]  
  }  
  
  render() {  
    return (  
      <div>  
        <h1>I'm a class component</h1>  
        <Person />  
        <Person />  
      </div>  
    );  
  }  
}
```

To refer to the state  
property we use

**this.state**

because we are  
accessing this object  
inside the class itself.



```
class App extends React.Component {  
  state = {  
    persons: [  
      {name: "Dan", age: 34},  
      {name: "Ben", age: 21}  
    ]  
  }  
}
```

```
  render() {  
    return (  
      <div>  
        <h1>I'm a class component</h1>  
        <Person />  
        <Person />  
      </div>  
    );  
  }  
}
```

So to access the persons  
property from the state  
object we would write  
**this.state.persons[1].age**

What would we have to  
write to access Ben's  
age?



# React.js

**Functional components are sometimes known as stateless components.**

**Class components are sometimes known as stateful components.**

# React.js

But this all got switched up with the introduction of Hooks.

# React.js

**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.**

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

```
const App = () => {
```

```
  const [count, setCount] = useState(0)
```

```
  return (
```

```
    <div>
```

```
      <h1>{count}</h1>
```

```
    </div>
```

```
  )
```

```
}
```

**We access the useState  
hook that is imported  
from React**

```
const [count, setCount] = useState(0)
```

**The useState function returns a pair of values which we restructure with [ ].**

**The first is a value, the second a function to update the first.**

**Hooks have a very basic setup and are a ridiculously quick way of adding state functionality instead of having to deal with class components**

# React.js

**Completely opt-in.** You can try Hooks in a few components without rewriting any existing code. But you don't have to learn or use Hooks right now if you don't want to.

**100% backwards-compatible.** Hooks don't contain any breaking changes.

**Think of the state as a kind of store, which holds the current state of the application.**

**When the app loads, it will be rendered with the initial state (that you coded in), but as users interact with your app, there state can change.**

# React.js

**As well as having a state property inside our component, we can also have different methods (functions) which can do things to our user interface.**



# React.js

**Let's look at a simple method which will log the string "hello world" to the console when we click on something.**

# React.js

We can change the state of our components **using methods**. Let's have a look how we might do that.


# React.js

**You can pass state and methods via props just like you used props previously. Let's have a look.**

```
const App = () => {
  const [persons, setPersons] = useState([
    {name: "Leon"},
    {name: "Jordan"}
  ])

  return (
    <div>
      <Person name={persons[0].name} />
      <Person name={persons[1].name} />
    </div>
  )
}

const Person = (props) => {
  return (
    <h1>{props.name}</h1>
  )
}
```



```
const App = () => {
  const [persons, setPersons] = useState([
    { name: "Leon" },
    { name: "Jordan" },
  ]);

  const handleClick = () => {
    console.log("Clickity click");
  };

  return (
    <div>
      <Person name={persons[0].name} clickMe={handleClick} />
      <Person name={persons[1].name} clickMe={handleClick} />
    </div>
  );
};

const Person = (props) => {
  return <h1 onClick={props.clickMe}>{props.name}</h1>;
};
```



# React.js

**Sometimes you might want to pass a value to your function, to do something with. In that case, we handle our methods slightly differently.**

```
const App = () => {  
  const [persons, setPersons] = useState([  
    { name: "Leon" },  
    { name: "Jordan" },  
  ]);
```

```
  const handleClick = (value) => {  
    console.log(value);  
  };
```

```
  return (  
    <div>  
      <Person name={persons[0].name} clickMe={handleClick} />  
      <Person name={persons[1].name} clickMe={handleClick} />  
    </div>  
  );  
};
```

```
const Person = (props) => {  
  return <h1 onClick={() => props.clickMe("Hi there")}>{props.name}</h1>;  
};
```

It doesn't matter where the function is used, if we want to pass it a value we must use it as an arrow function.

We call this an anonymous function.

# React.js

**We are now going to look at rendering dynamic content. Basically rendering a component, when a certain condition is met.**



# React.js

**We can render content conditionally in a few different ways.**

**? ( ) : ( )**

**Ternary  
Operator**

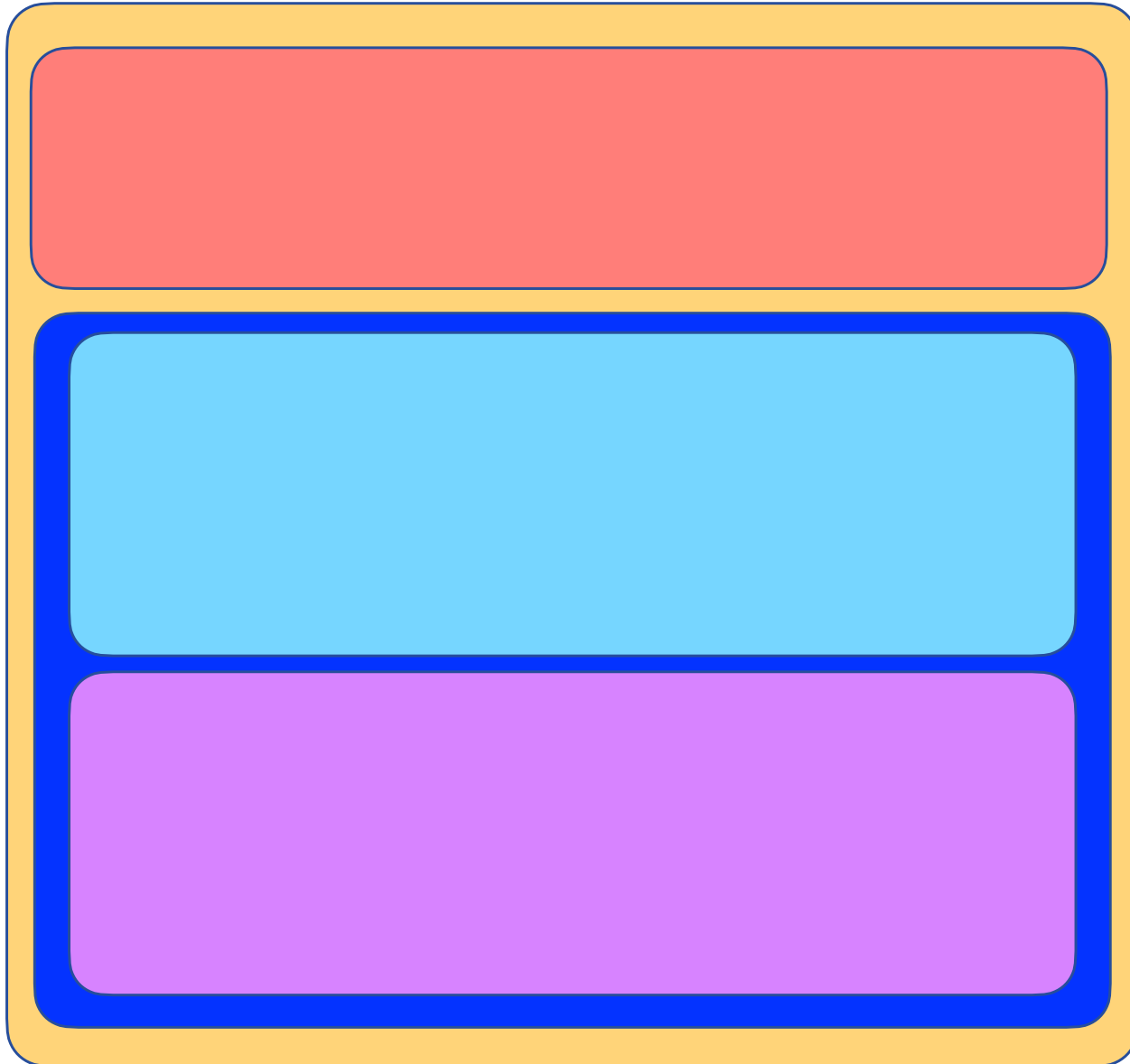
**&&**

**Short Circuit  
Evaluation**

**if statement**

# React.js

**Another (longer) way we COULD render things conditionally would be to use if/else statements inside our render method. I'll live code you an example. It's usually much cleaner to use ternaries or short circuit evaluation.**



# Class Components

Properties and methods go here

**Render( ) method**  
JS logic can go here.

return statement

# Functional Components

Hooks and methods go here  
JS logic can go here.

return statement



## Challenge

Create a h1 element which shows a number (stored in the state) initially at zero. Create two methods and two buttons. The first button should increase the number by 1 onClick, the other should decrease the number by 1.

# Revisiting Learning Objectives

**To understand what state is and how we can work with it.**

**To be able to include methods in class components.**