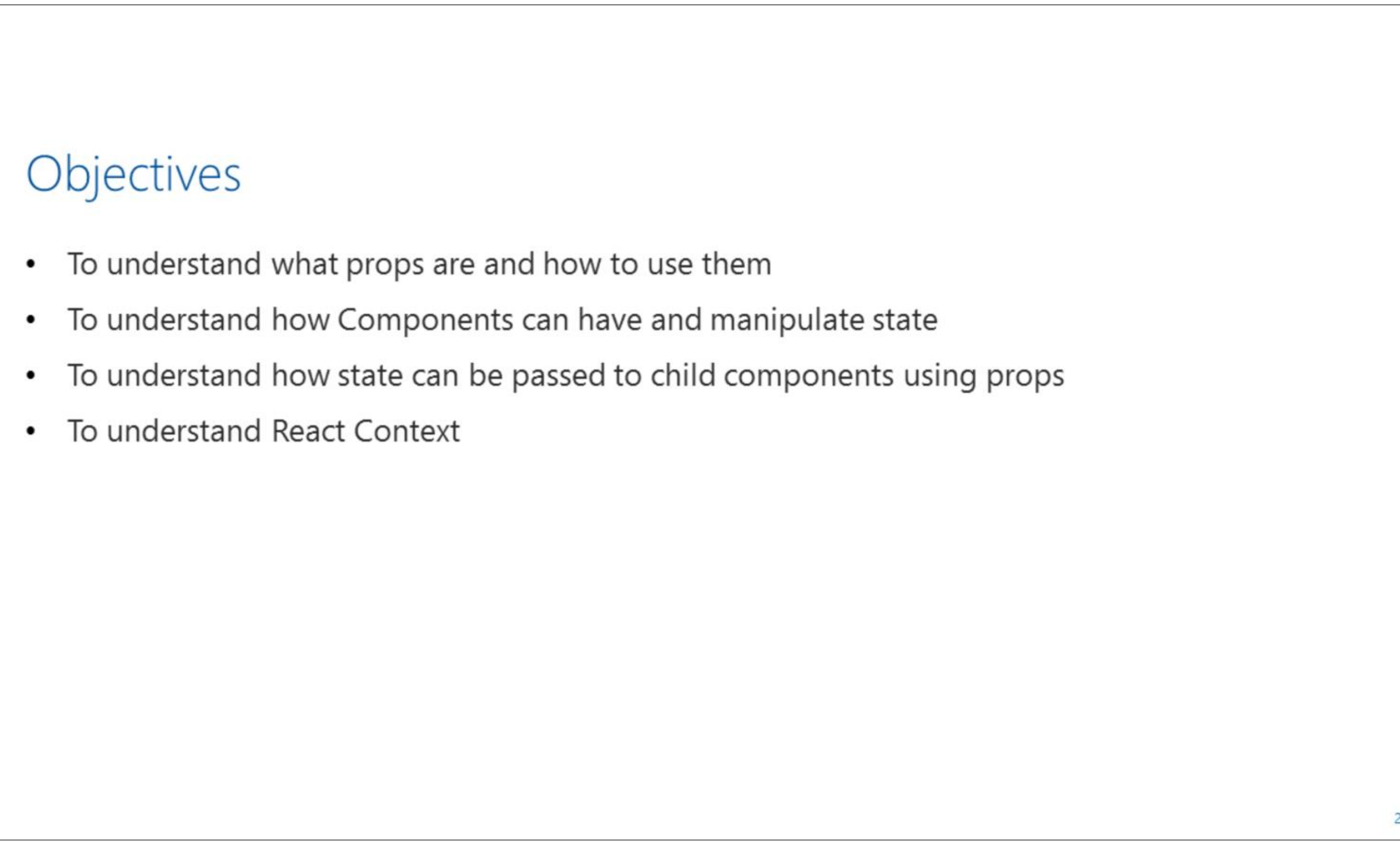
Using props and state - Adding Properties to Components

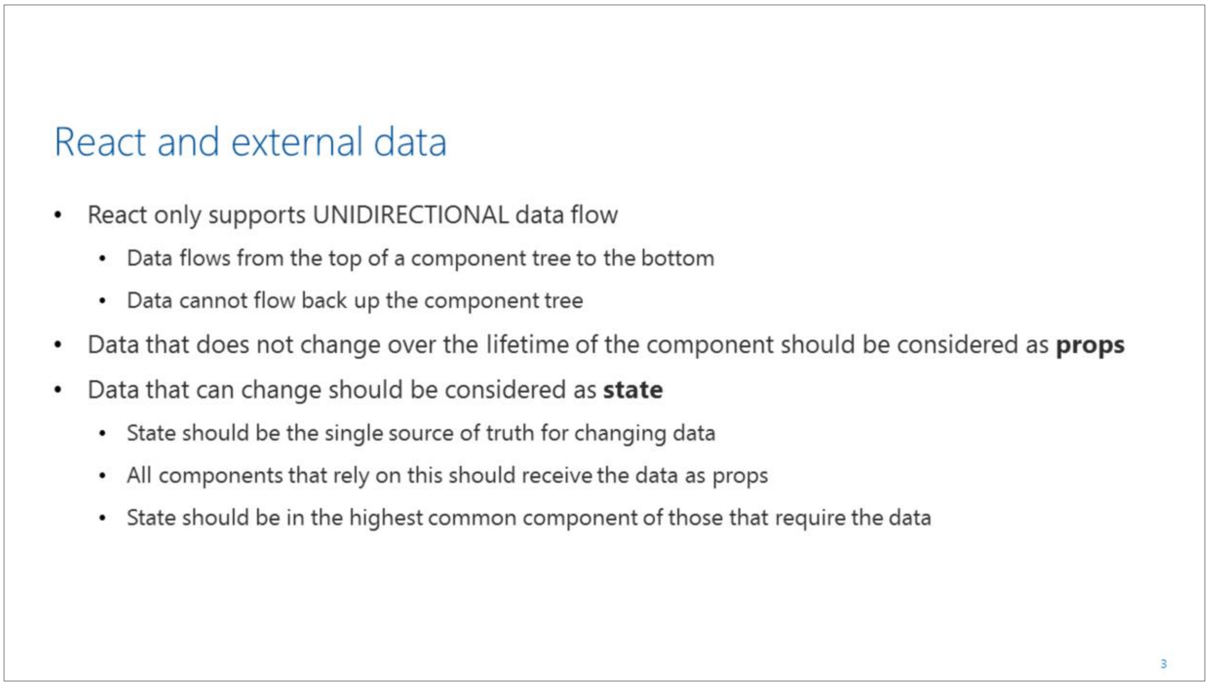
DEVELOPING APPLICATIONS USING REACTJS

QA



Objectives

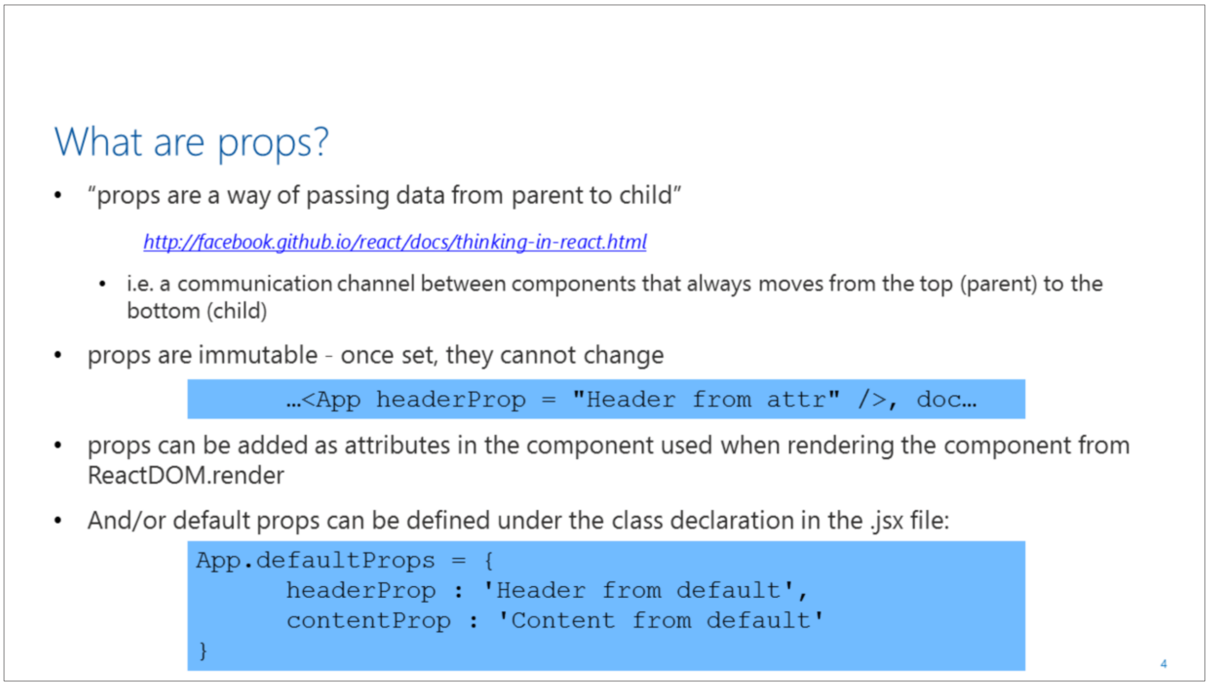
- To understand what props are and how to use them
- To understand how Components can have and manipulate state
- To understand how state can be passed to child components using props
- To understand React Context



React and external data

- React only supports UNIDIRECTIONAL data flow
  - Data flows from the top of a component tree to the bottom
  - Data cannot flow back up the component tree
- Data that does not change over the lifetime of the component should be considered as **props**
- Data that can change should be considered as **state**
  - State should be the single source of truth for changing data
  - All components that rely on this should receive the data as props
  - State should be in the highest common component of those that require the data

3



What are props?

- "props are a way of passing data from parent to child"

  http://facebook.github.io/react/docs/thinking-in-react.html

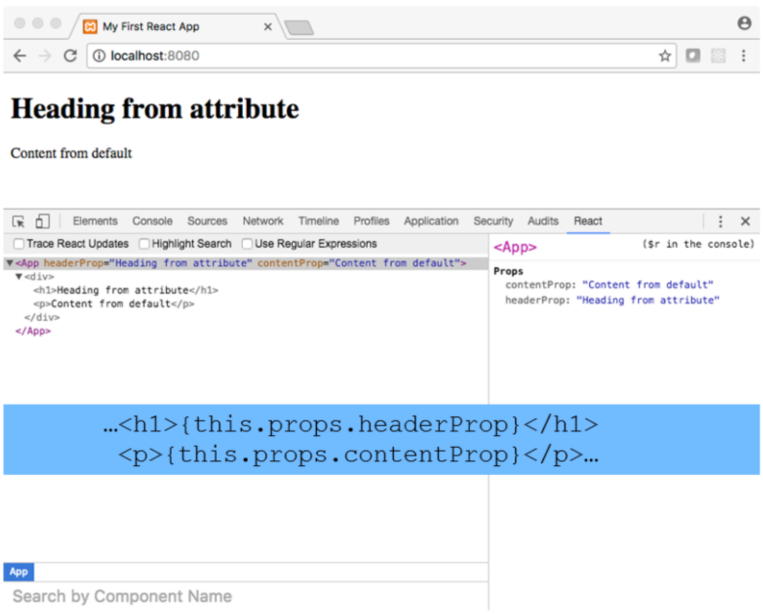
  - i.e. a communication channel between components that always moves from the top (parent) to the bottom (child)
- props are immutable - once set, they cannot change

```
…<App headerProp = "Header from attr" />, doc…
```

- props can be added as attributes in the component used when rendering the component from ReactDOM.render
- And/or default props can be defined under the class declaration in the .jsx file:

```
App.defaultProps = {
      headerProp : 'Header from default',
      contentProp : 'Content from default'
}
```

USING PROPS IN COMPONENTS


My First React App

localhost:8080

Heading from attribute

Content from default

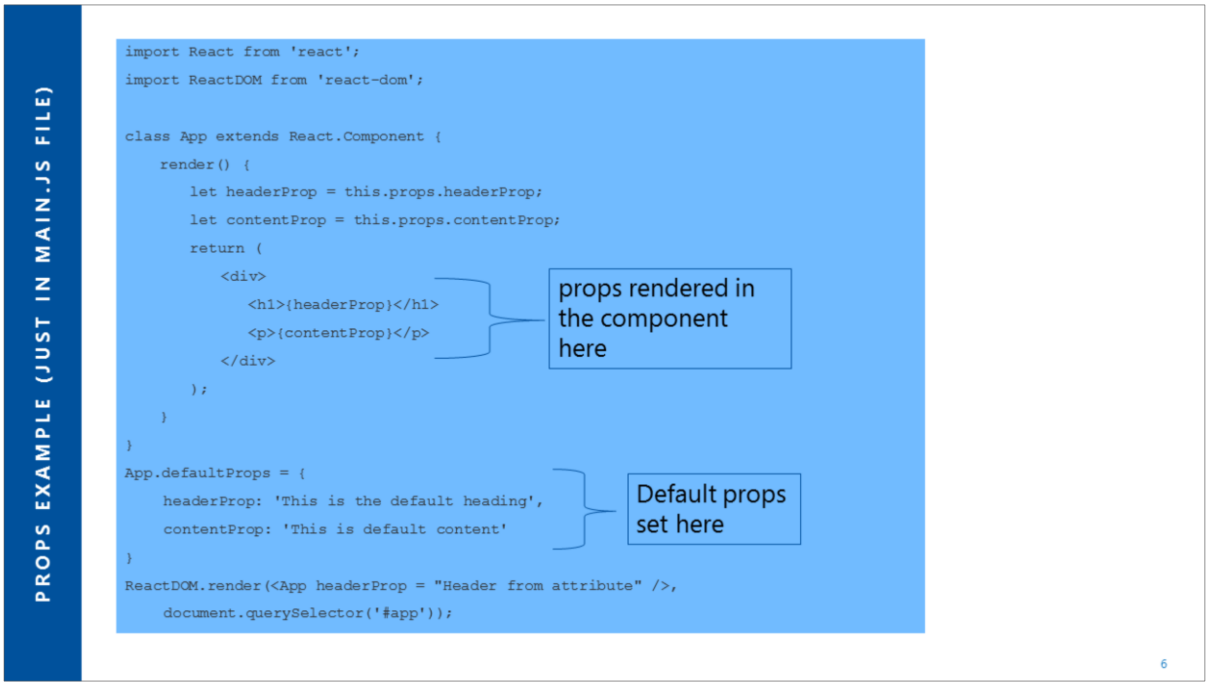
```
☐ Trace React Updates  ☐ Highlight Search  ☐ Use Regular Expressions
▼<App headerProp="Heading from attribute" contentProp="Content from default">
  ▼<div>
     <h1>Heading from attribute</h1>
     <p>Content from default</p>
   </div>
 </App>
```

```
<App>                    ($r in the console)
Props
  contentProp: "Content from default"
  headerProp: "Heading from attribute"
```

```
…<h1>{this.props.headerProp}</h1>
  <p>{this.props.contentProp}</p>…
```

props rendered to the browser through the component return – either the default, or overriding value (if supplied)

5



PROPS EXAMPLE (JUST IN MAIN.JS FILE)

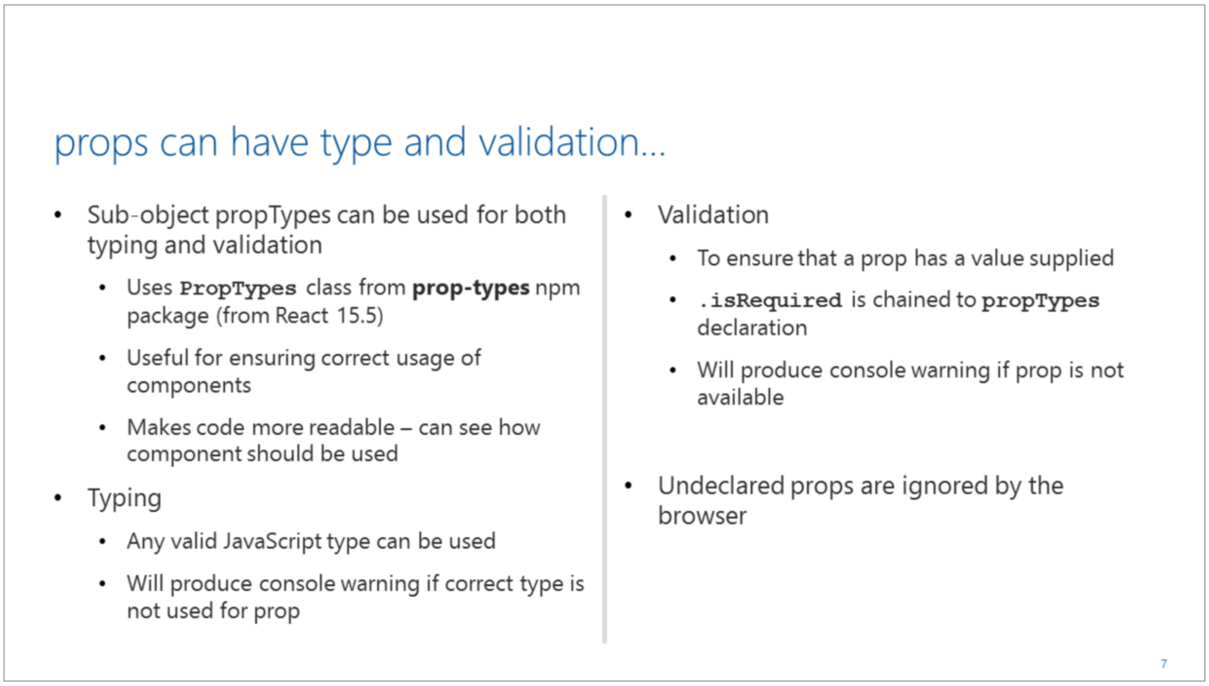
```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
    render() {
        let headerProp = this.props.headerProp;
        let contentProp = this.props.contentProp;
        return (
            <div>
               <h1>{headerProp}</h1>
               <p>{contentProp}</p>
            </div>
        );
    }
}
App.defaultProps = {
    headerProp: 'This is the default heading',
    contentProp: 'This is default content'
}
ReactDOM.render(<App headerProp = "Header from attribute" />,
    document.querySelector('#app'));
```

props rendered in the component here

Default props set here

6

This component will be rendered as expected with the header being displayed from the overriding attribute setting and the content being rendered from the default.

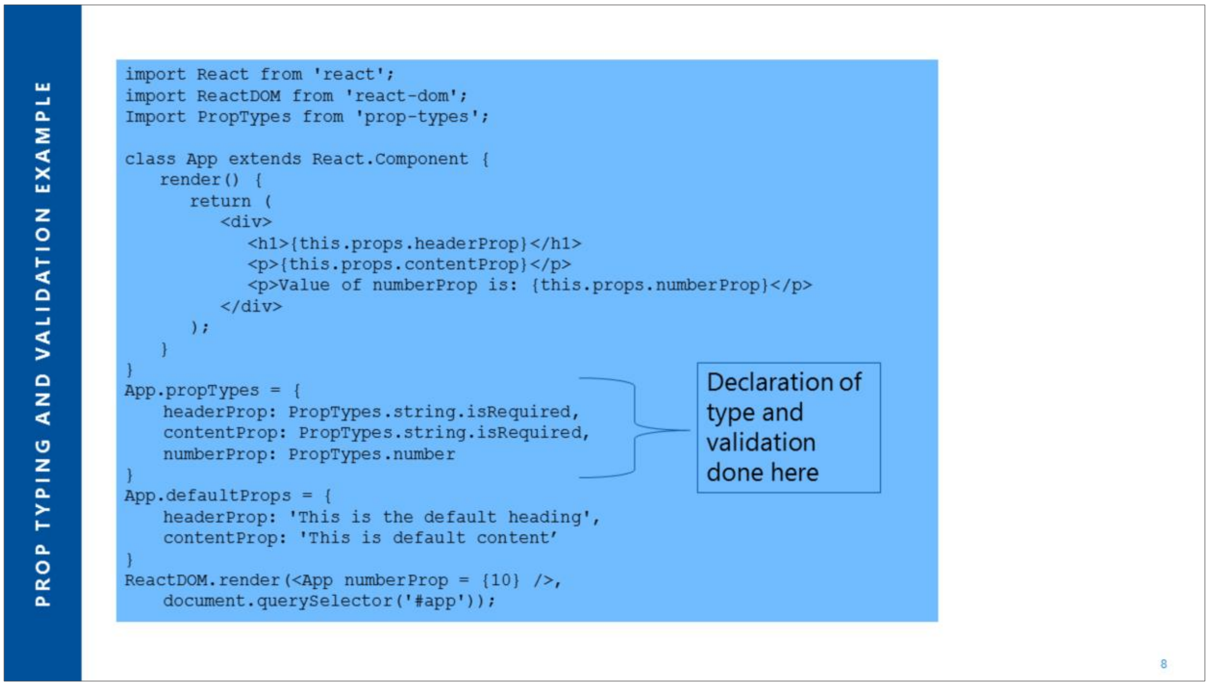


props can have type and validation…

- Sub-object propTypes can be used for both typing and validation
  - Uses `PropTypes` class from **prop-types** npm package (from React 15.5)
  - Useful for ensuring correct usage of components
  - Makes code more readable – can see how component should be used
- Typing
  - Any valid JavaScript type can be used
  - Will produce console warning if correct type is not used for prop
- Validation
  - To ensure that a prop has a value supplied
  - `.isRequired` is chained to `propTypes` declaration
  - Will produce console warning if prop is not available
- Undeclared props are ignored by the browser

7

To avoid the bloating of the React package, React.PropTypes (along with React,createClass) was removed from it when v15.5 was released. They are now sourced from the prop-types npm packge.



PROP TYPING AND VALIDATION EXAMPLE

```
import React from 'react';
import ReactDOM from 'react-dom';
Import PropTypes from 'prop-types';

class App extends React.Component {
    render() {
       return (
          <div>
             <h1>{this.props.headerProp}</h1>
             <p>{this.props.contentProp}</p>
             <p>Value of numberProp is: {this.props.numberProp}</p>
          </div>
       );
    }
}
App.propTypes = {
    headerProp: PropTypes.string.isRequired,
    contentProp: PropTypes.string.isRequired,
    numberProp: PropTypes.number
}
App.defaultProps = {
    headerProp: 'This is the default heading',
    contentProp: 'This is default content’
}
ReactDOM.render(<App numberProp = {10} />,
    document.querySelector('#app'));
```

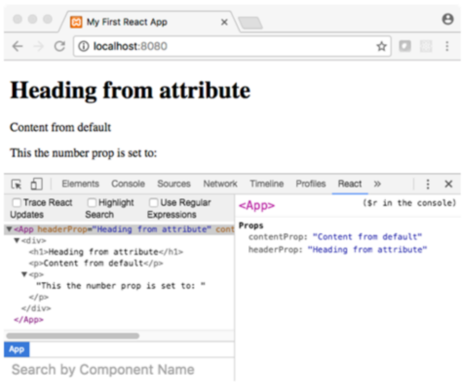
Declaration of type and validation done here

8

In this example, the number prop has to be a number if it is supplied, else there will be a console warning. Therefore, this component will be rendered as expected with the header and content being displayed from the default and numberProp being evaluated to 10 through the attributes.

prop type and validation

- Unset prop is ignored by the browser with no errors



Heading from attribute

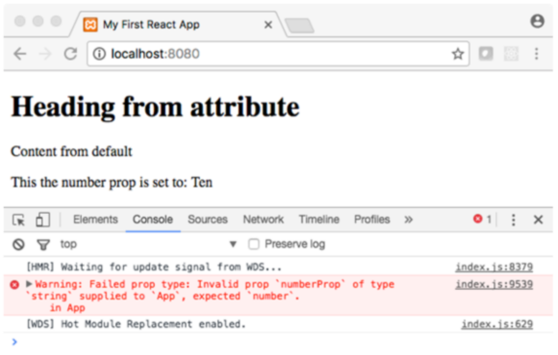
Content from default

This the number prop is set to:

```
▼<App headerProp="Heading from attribute" cont
  ▼<div>
     <h1>Heading from attribute</h1>
     <p>Content from default</p>
    ▼<p>
       "This the number prop is set to: "
     </p>
   </div>
 </App>
```

```
<App>                    ($r in the console)
Props
  contentProp: "Content from default"
  headerProp: "Heading from attribute"
```

- Incorrect type used in prop produces console warning but is still rendered



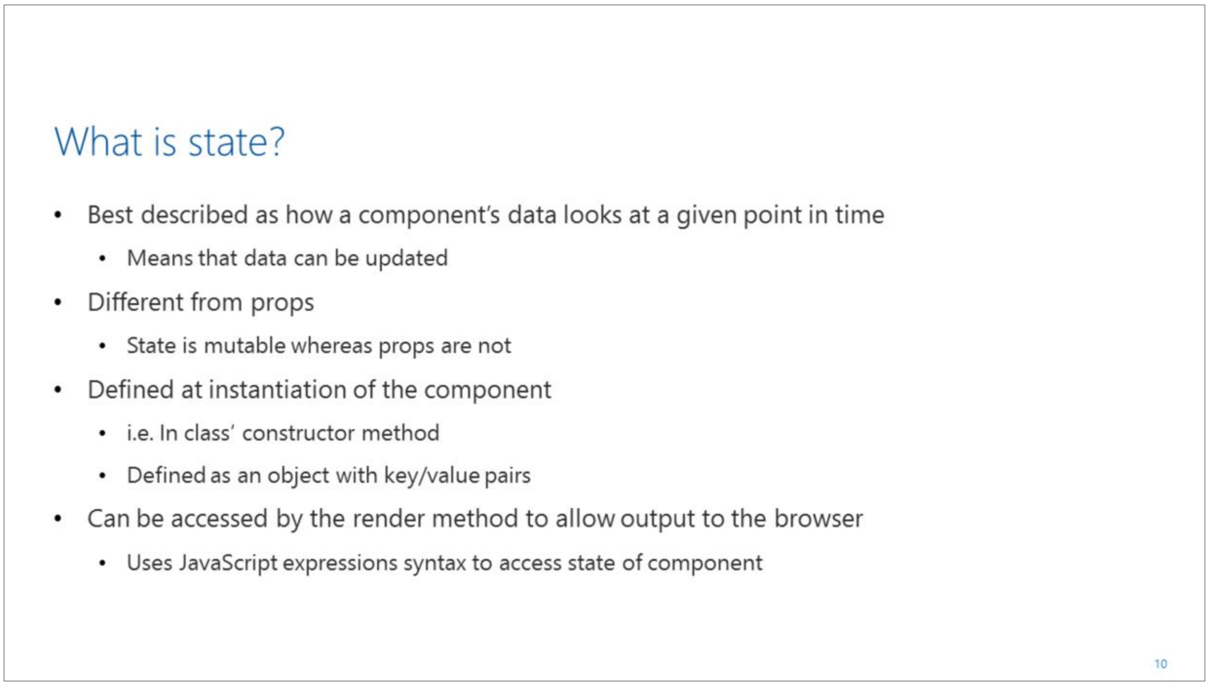
Heading from attribute

Content from default

This the number prop is set to: Ten

```
[HMR] Waiting for update signal from WDS...                         index.js:8379
▶Warning: Failed prop type: Invalid prop `numberProp` of type       index.js:9539
`string` supplied to `App`, expected `number`.
    in App
[WDS] Hot Module Replacement enabled.                               index.js:629
>
```

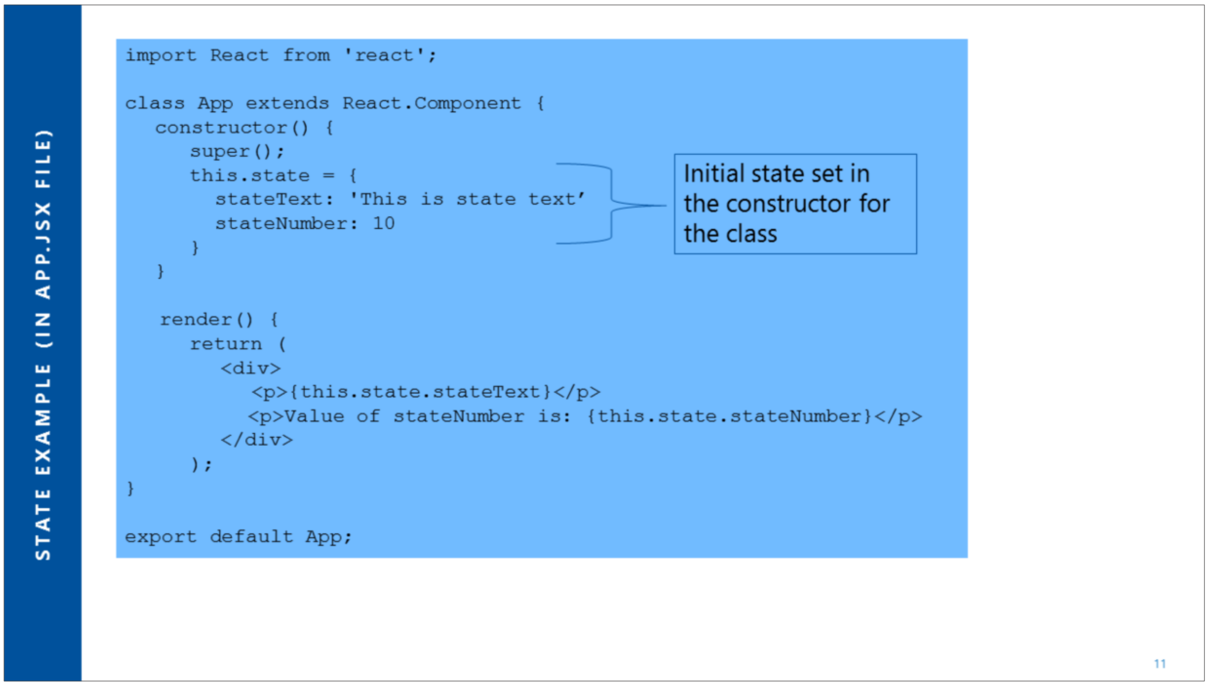
9



What is state?

- Best described as how a component's data looks at a given point in time
  - Means that data can be updated
- Different from props
  - State is mutable whereas props are not
- Defined at instantiation of the component
  - i.e. In class' constructor method
  - Defined as an object with key/value pairs
- Can be accessed by the render method to allow output to the browser
  - Uses JavaScript expressions syntax to access state of component

10



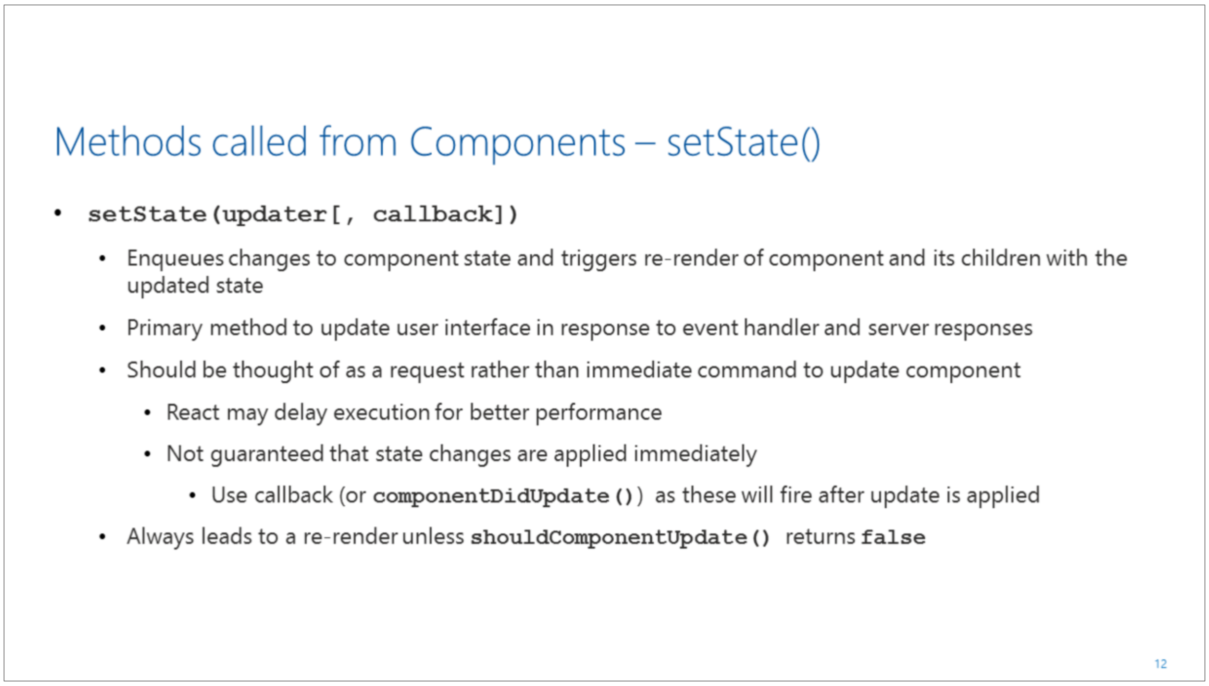
STATE EXAMPLE (IN APP.JSX FILE)

```
import React from 'react';

class App extends React.Component {
   constructor() {
      super();
      this.state = {
        stateText: 'This is state text'
        stateNumber: 10
      }
   }

   render() {
      return (
         <div>
            <p>{this.state.stateText}</p>
            <p>Value of stateNumber is: {this.state.stateNumber}</p>
         </div>
      );
}

export default App;
```

Initial state set in the constructor for the class

11

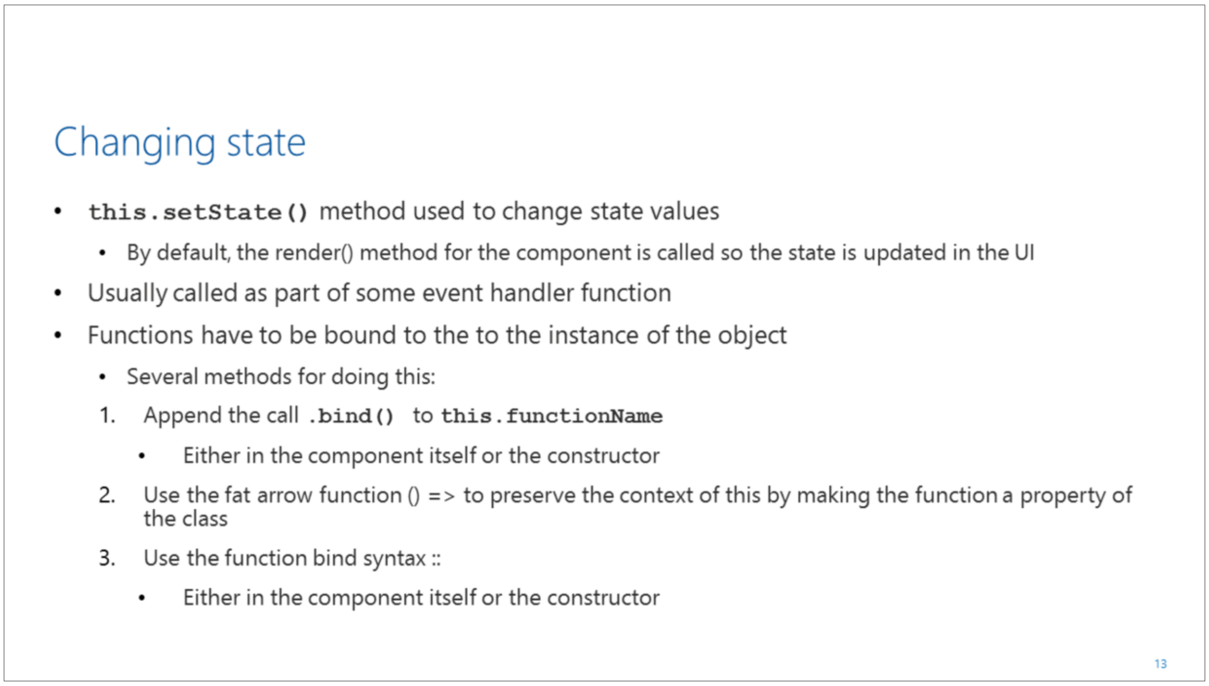
As with props, any state that is not declared in the constructor and then used in the render function will be ignored by the browser.



Methods called from Components – setState()

- **`setState(updater[, callback])`**
  - Enqueues changes to component state and triggers re-render of component and its children with the updated state
  - Primary method to update user interface in response to event handler and server responses
  - Should be thought of as a request rather than immediate command to update component
    - React may delay execution for better performance
    - Not guaranteed that state changes are applied immediately
      - Use callback (or **`componentDidUpdate()`**) as these will fire after update is applied
  - Always leads to a re-render unless **`shouldComponentUpdate()`** returns **`false`**

12



Changing state

- `this.setState()` method used to change state values
  - By default, the render() method for the component is called so the state is updated in the UI
- Usually called as part of some event handler function
- Functions have to be bound to the to the instance of the object
  - Several methods for doing this:
  1. Append the call `.bind()` to `this.functionName`
     - Either in the component itself or the constructor
  2. Use the fat arrow function () => to preserve the context of this by making the function a property of the class
  3. Use the function bind syntax ::
     - Either in the component itself or the constructor

13

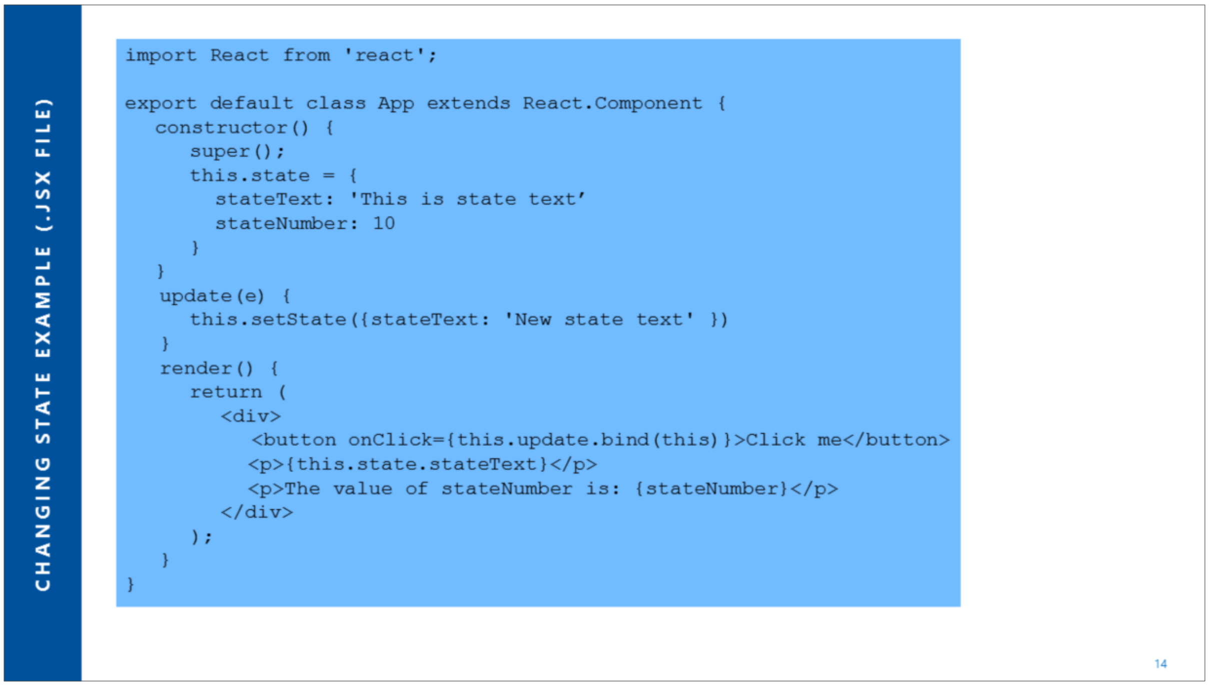
1. Appending the call bind() to this.functionName:
   a. <button onClick={this.functionName.bind(this)}>Click</button>
   // Used in the component's render function
   b. this.functionName = this.functionName.bind(this)
   // Used as part of the constructor
2. Using the fat arrow function:

   functionName = () => this.functionName();
   // Used as a class variable or as part of constructor

3.Using the bind function syntax

   a. this.functionName = ::this.functionName;
   // Used as part of the constructor
   b. <button onClick={::this.functionName}>Click</button>

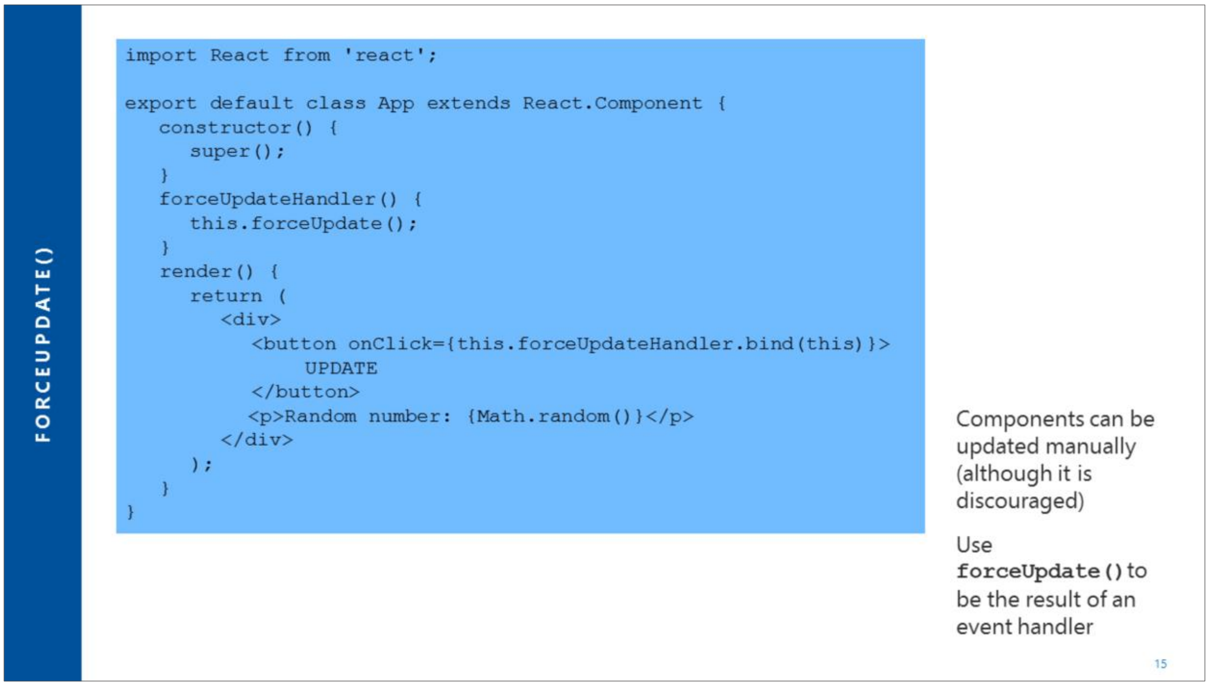
From here on in, the example given in 1b will be used as it is the most explicit and reliable at the time of the course being authored.



CHANGING STATE EXAMPLE (.JSX FILE)

```
import React from 'react';

export default class App extends React.Component {
   constructor() {
      super();
      this.state = {
         stateText: 'This is state text’
         stateNumber: 10
      }
   }
   update(e) {
      this.setState({stateText: 'New state text' })
   }
   render() {
      return (
         <div>
            <button onClick={this.update.bind(this)}>Click me</button>
            <p>{this.state.stateText}</p>
            <p>The value of stateNumber is: {stateNumber}</p>
         </div>
      );
   }
}
```

14



FORCEUPDATE()

```
import React from 'react';

export default class App extends React.Component {
   constructor() {
      super();
   }
   forceUpdateHandler() {
      this.forceUpdate();
   }
   render() {
      return (
         <div>
            <button onClick={this.forceUpdateHandler.bind(this)}>
                UPDATE
            </button>
            <p>Random number: {Math.random()}</p>
         </div>
      );
   }
}
```

Components can be updated manually (although it is discouraged)

Use **`forceUpdate()`** to be the result of an event handler

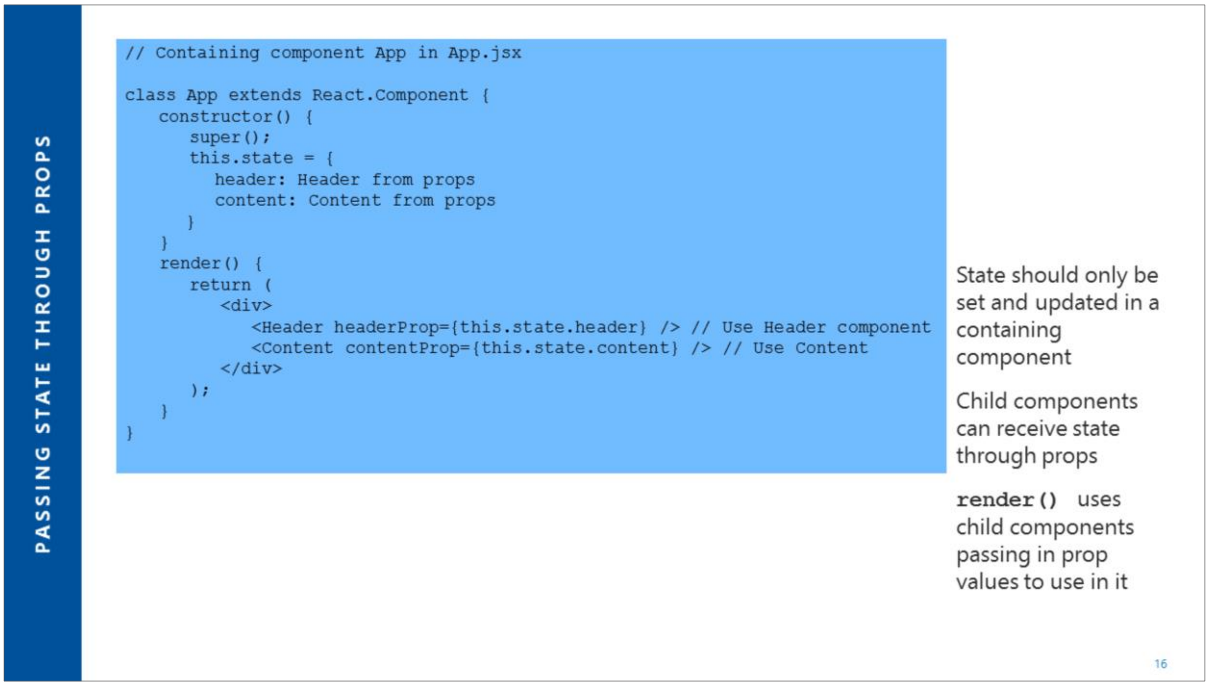
15

forceUpdate() is a React function that causes a component to be re-rendered (if the DOM mark-up has changed). Its use is discouraged, even by the API for the function, which states:

Normally you should try to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

https://facebook.github.io/react/docs/react-component.html#forceupdate

It is sometimes unavoidable and this example forces an update on the Component, which in turn generates a new random number to be displayed, as the Math.random() function is called when the component is re-rendered. However, this example could have been written using a randomNumber state and using a function to update the state rather than re-rendering the whole component.



PASSING STATE THROUGH PROPS

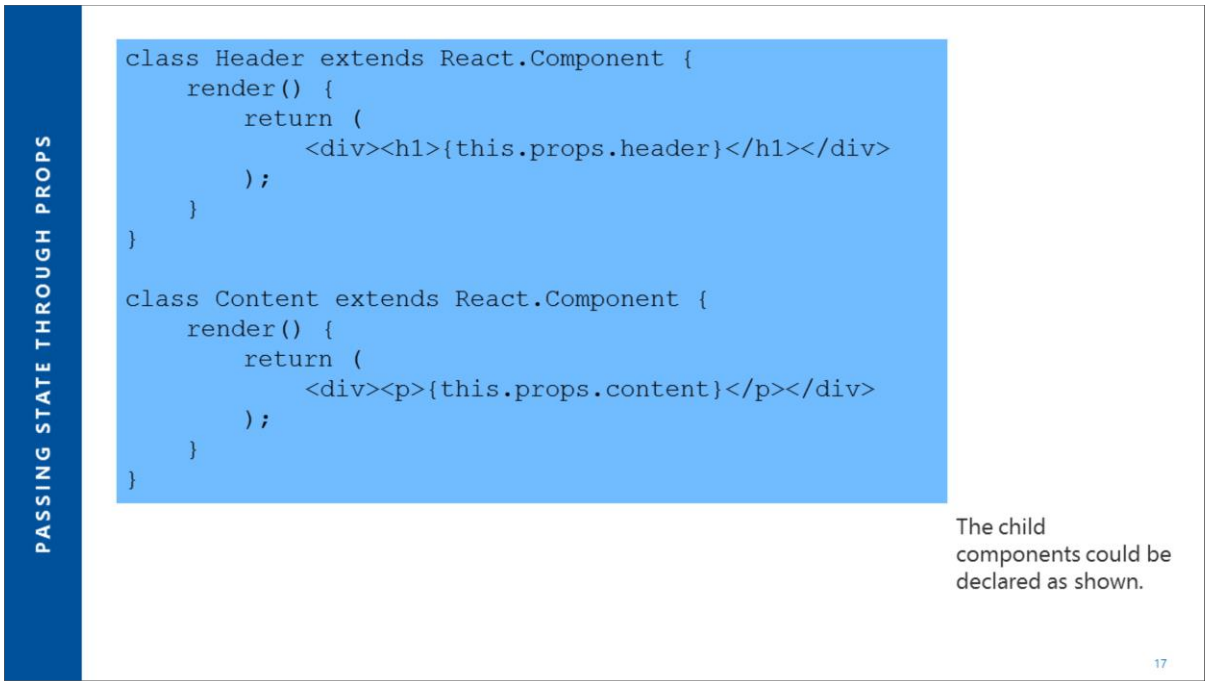
```
// Containing component App in App.jsx

class App extends React.Component {
   constructor() {
      super();
      this.state = {
         header: Header from props
         content: Content from props
      }
   }
   render() {
      return (
         <div>
            <Header headerProp={this.state.header} /> // Use Header component
            <Content contentProp={this.state.content} /> // Use Content
         </div>
      );
   }
}
```

State should only be set and updated in a containing component

Child components can receive state through props

`render()` uses child components passing in prop values to use in it

16

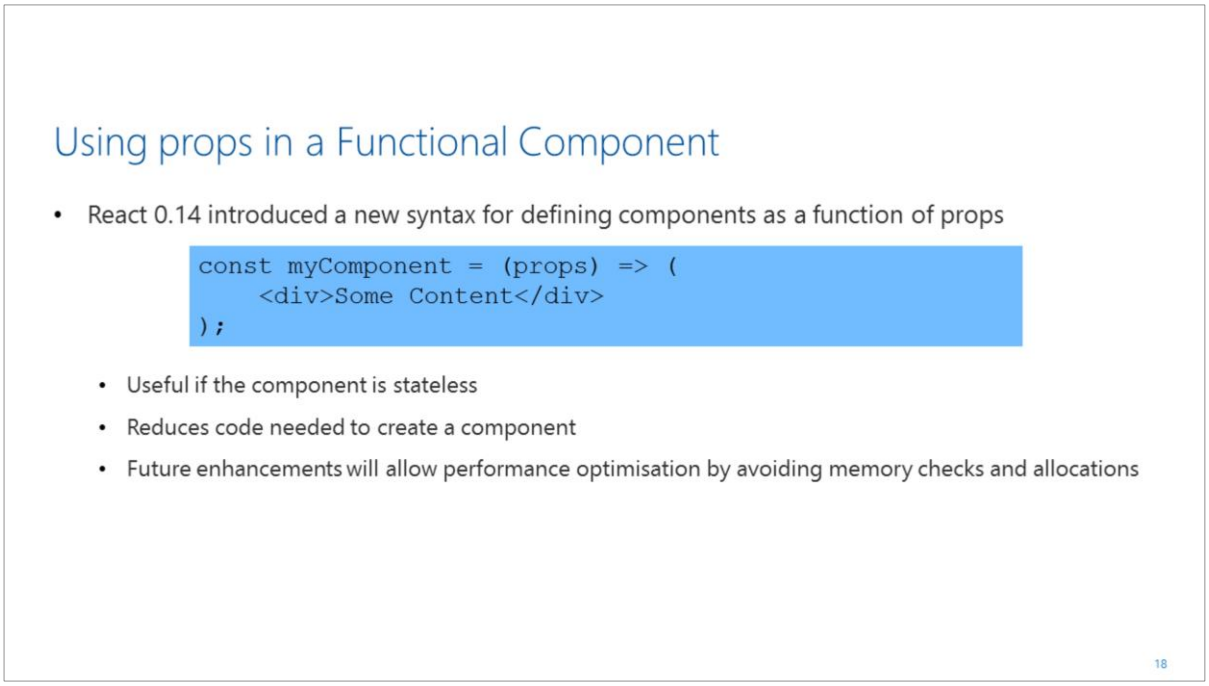


PASSING STATE THROUGH PROPS

```
class Header extends React.Component {
    render() {
        return (
            <div><h1>{this.props.header}</h1></div>
        );
    }
}

class Content extends React.Component {
    render() {
        return (
            <div><p>{this.props.content}</p></div>
        );
    }
}
```

The child components could be declared as shown.

17



Using props in a Functional Component

- React 0.14 introduced a new syntax for defining components as a function of props

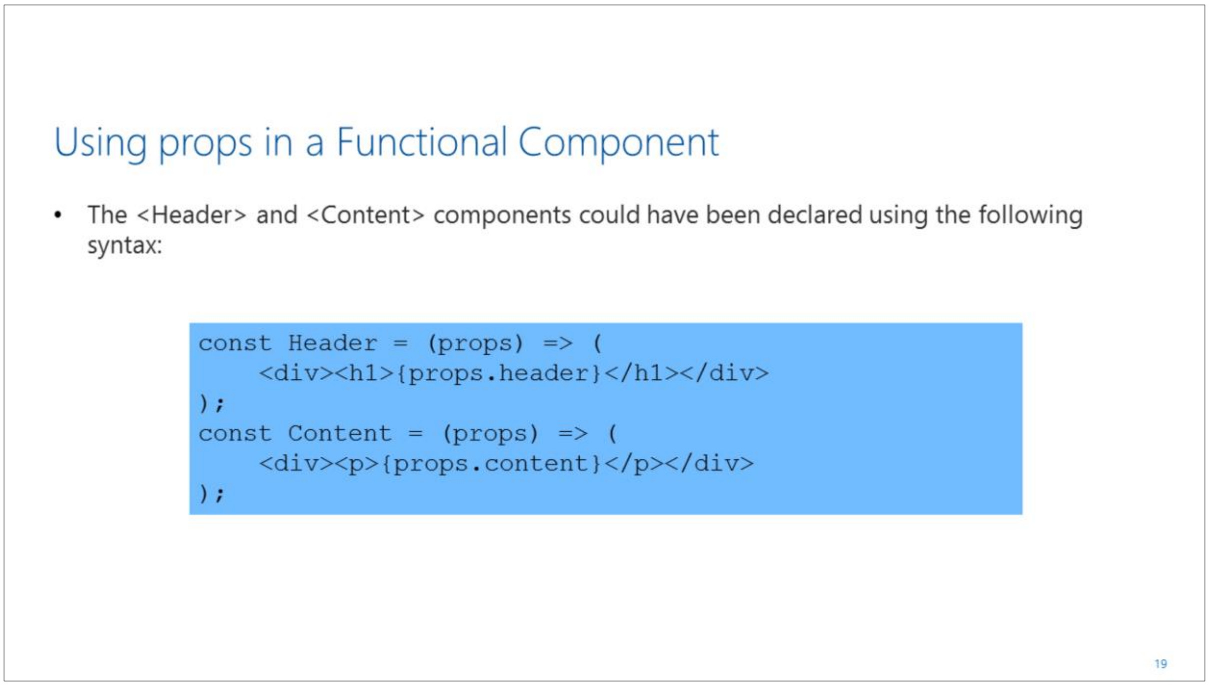
```
const myComponent = (props) => (
    <div>Some Content</div>
);
```

  - Useful if the component is stateless
  - Reduces code needed to create a component
  - Future enhancements will allow performance optimisation by avoiding memory checks and allocations

18

Some advantages of making a functional component if it is a stateless component:

1. No class declaration is needed
2. No need to use the this keyword – which means no need to bind functions (as the component is not an instance of an object!)
3. Focuses on UI behaviour – state managed by higher-level 'container' components
4. Less code for same/more output
5. Bloated components and poor data structures more easily spotted.
6. Easy to understand – even if it contains a lot of markup
7. Easy to test – assertions are straightforward: Given these props, I expect this markup returned.



Using props in a Functional Component

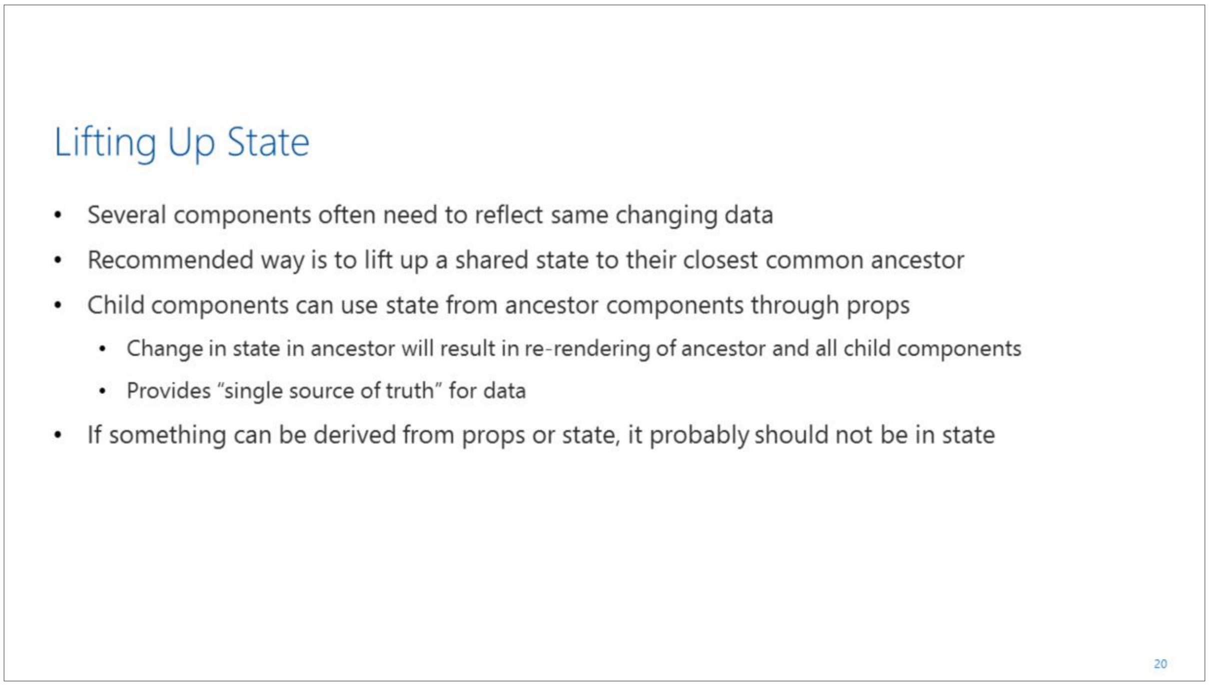
- The <Header> and <Content> components could have been declared using the following syntax:

```
const Header = (props) => (
    <div><h1>{props.header}</h1></div>
);
const Content = (props) => (
    <div><p>{props.content}</p></div>
);
```

19

Some advantages of making a functional component if it is a stateless component:

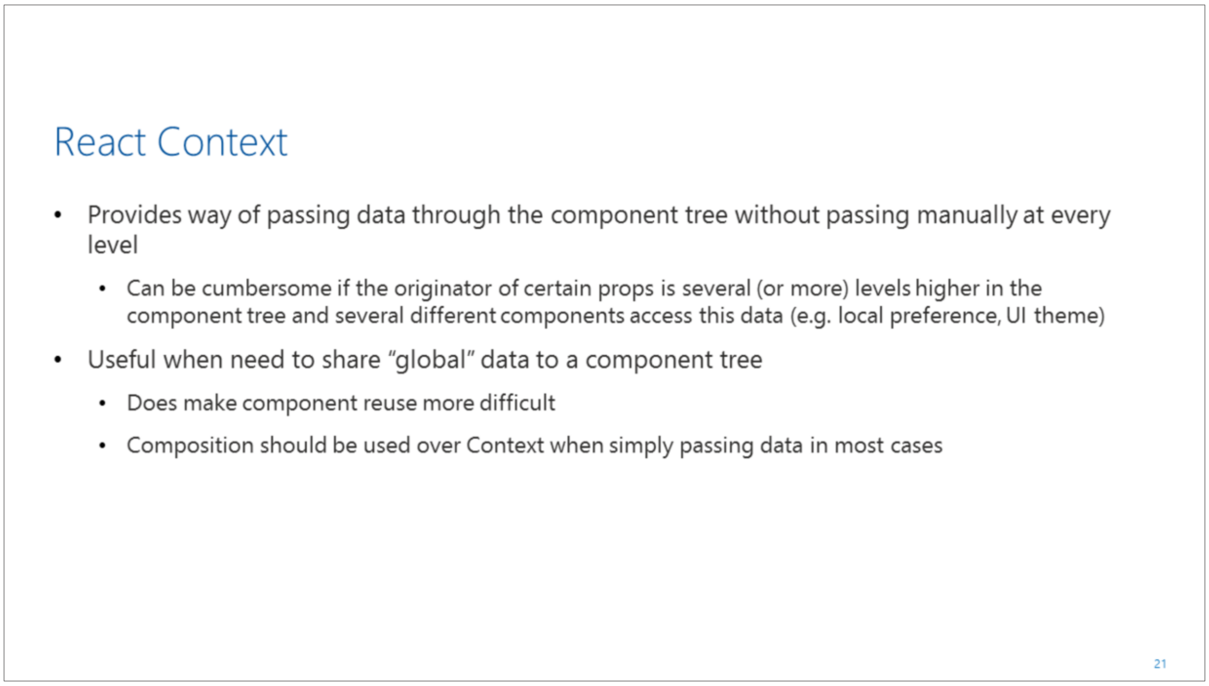
1. No class declaration is needed
2. No need to use the this keyword – which means no need to bind functions (as the component is not an instance of an object!)
3. Focuses on UI behaviour – state managed by higher-level 'container' components
4. Less code for same/more output
5. Bloated components and poor data structures more easily spotted.
6. Easy to understand – even if it contains a lot of markup
7. Easy to test – assertions are straightforward: Given these props, I expect this markup returned.



Lifting Up State

- Several components often need to reflect same changing data
- Recommended way is to lift up a shared state to their closest common ancestor
- Child components can use state from ancestor components through props
  - Change in state in ancestor will result in re-rendering of ancestor and all child components
  - Provides “single source of truth” for data
- If something can be derived from props or state, it probably should not be in state

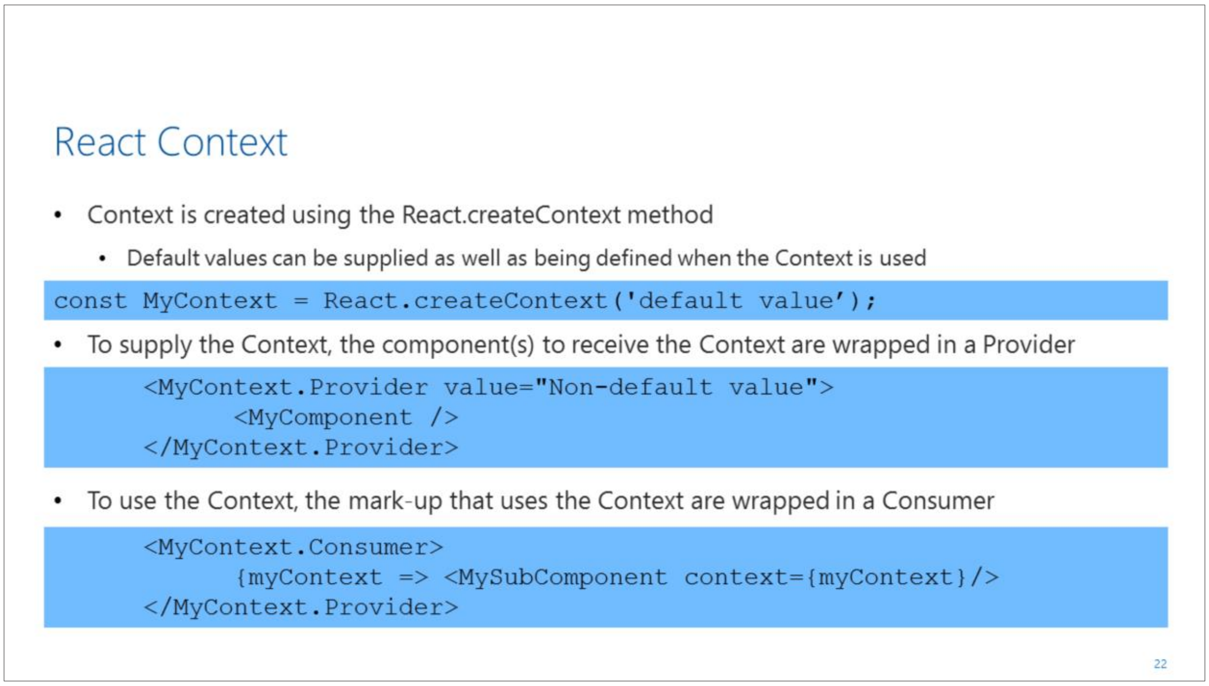
20



React Context

- Provides way of passing data through the component tree without passing manually at every level
  - Can be cumbersome if the originator of certain props is several (or more) levels higher in the component tree and several different components access this data (e.g. local preference, UI theme)
- Useful when need to share “global” data to a component tree
  - Does make component reuse more difficult
  - Composition should be used over Context when simply passing data in most cases

21



React Context

- Context is created using the React.createContext method
  - Default values can be supplied as well as being defined when the Context is used

```
const MyContext = React.createContext('default value');
```

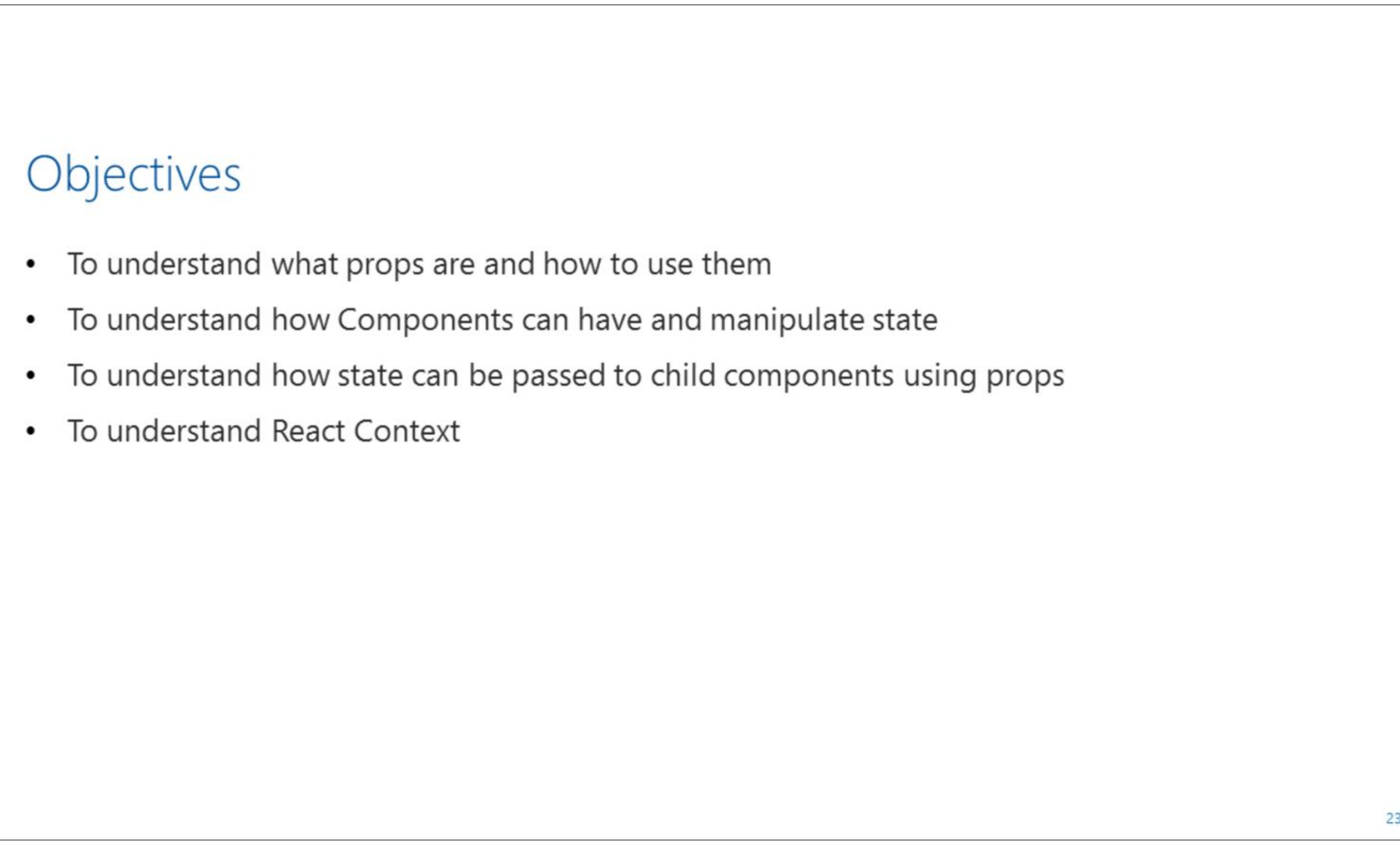
- To supply the Context, the component(s) to receive the Context are wrapped in a Provider

```
      <MyContext.Provider value="Non-default value">
            <MyComponent />
      </MyContext.Provider>
```

- To use the Context, the mark-up that uses the Context are wrapped in a Consumer

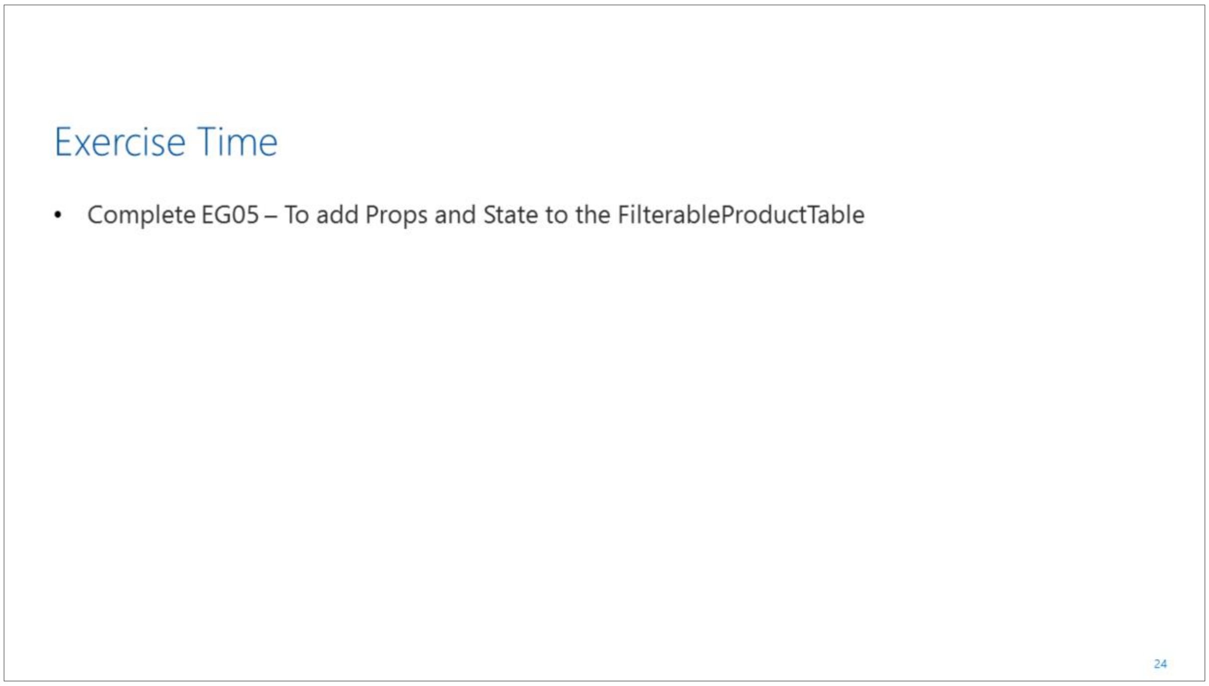
```
      <MyContext.Consumer>
            {myContext => <MySubComponent context={myContext}/>
      </MyContext.Provider>
```

22



Objectives

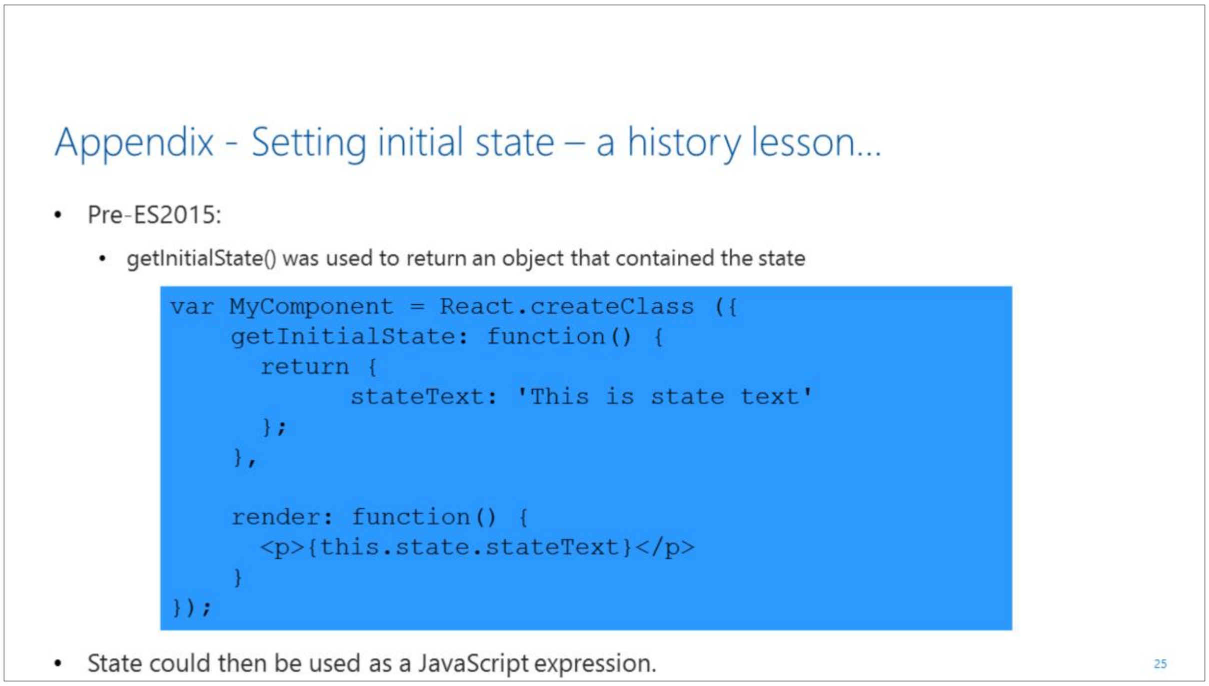
- To understand what props are and how to use them
- To understand how Components can have and manipulate state
- To understand how state can be passed to child components using props
- To understand React Context



Exercise Time

- Complete EG05 – To add Props and State to the FilterableProductTable

24



Appendix - Setting initial state – a history lesson...

- Pre-ES2015:
  - getInitialState() was used to return an object that contained the state

```
var MyComponent = React.createClass ({
    getInitialState: function() {
      return {
            stateText: 'This is state text'
      };
    },

    render: function() {
      <p>{this.state.stateText}</p>
    }
});
```

- State could then be used as a JavaScript expression.

25

The constructor function in ES6 class declarations has done away with the need for the getInitialState() method.