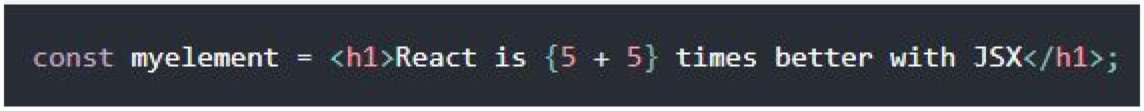
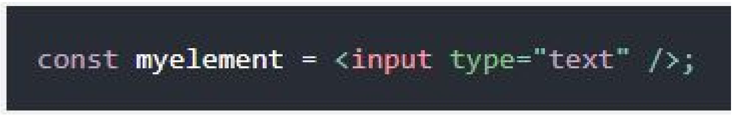
Introduction:
- React is a JS library created by Facebook and is used for building user interfaces and front-end applications.
- React is a framework.
- It makes front-end JS much easier.
- Uses a virtual DOM.

JSX:
- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.
- JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
- JSX converts HTML tags into react elements.
- To use vanilla JS in React, do **{JS code}**.
  I.e. You can put any valid JavaScript expression inside the curly braces in JSX.
- E.g.



```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

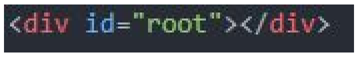
- Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names. For example, class becomes className in JSX.
- **Note:** All elements in JSX must be enclosed. For the tags that don't have a closing tag, like input, you can do **<input … />**. JSX follows XML rules, and therefore HTML elements must be properly closed. Close empty elements with />
  E.g.



```
const myelement = <input type="text" />;
```

- JSX will throw an error if the HTML is not properly closed.

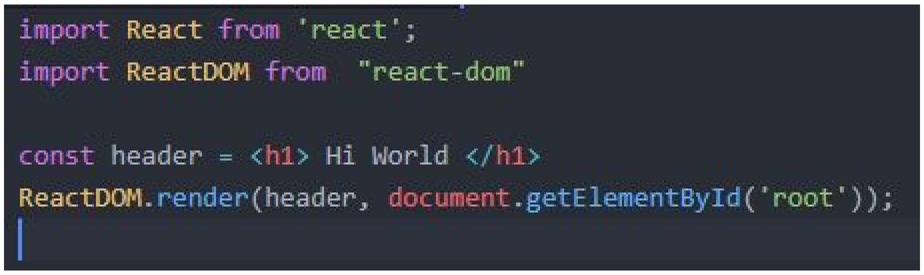
ReactDOM.render:
- Elements in React are the smallest building blocks of React apps. An element describes what you want to see on the screen.
- React elements are immutable. Once you create an element, you can't change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.
- Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements. React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.
- When you create a React app, you'll see a folder called "public" and in it, you'll see a file called index.html. Inside index.html, you'll see this line:



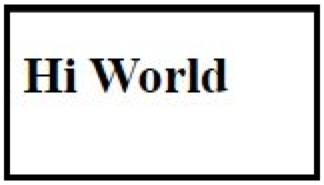
```
<div id="root"></div>
```

  We call this a "root" DOM node because everything inside it will be managed by React DOM.Applications built with just React usually have a single root DOM node.

- To render a React element into a root DOM node, React uses a function called **ReactDOM.render()**. It takes two arguments, HTML code and an HTML element.
  I.e. To render a React element into a root DOM node, pass both to ReactDOM.render().
- The **ReactDOM.render(element, container)** function is a top-level API that takes a React element root of an element tree and a container DOM element as arguments. It then turns that passed React element into a corresponding DOM element (tree) and then mounts that element as a child in the container DOM element. Before mounting any DOM elements to the container though, React performs a diff between the passed element tree and the currently mounted DOM to determine which of the DOM nodes in the currently mounted DOM have to be updated in order to match the newly passed element tree.
- **Note:** In practice, most React apps only call **ReactDOM.render()** once.
- E.g.



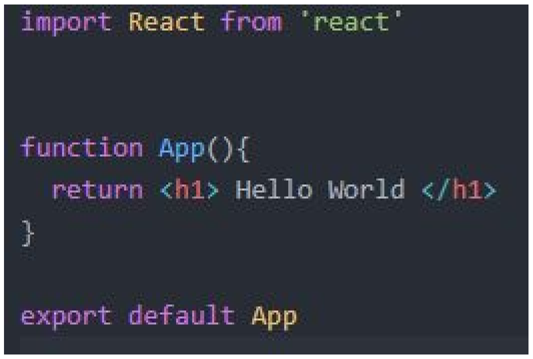
```
import React from 'react';
import ReactDOM from  "react-dom"

const header = <h1> Hi World </h1>
ReactDOM.render(header, document.getElementById('root'));
```



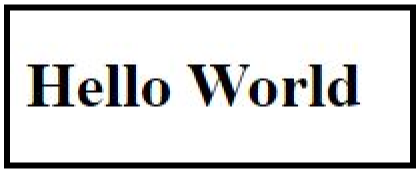
Hi World

Components, Render, Props:

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. Conceptually, components are like JavaScript functions. They accept arbitrary inputs called props and return React elements describing what should appear on the screen.
- **Note:** Always start component names with a capital letter. React treats components starting with lowercase letters as DOM tags. For example, <div /> represents an HTML div tag, but <Welcome /> represents a component and requires Welcome to be in scope.
- The simplest way to define a component is to write a JavaScript function.
- E.g.



```
import React from 'react'


function App(){
  return <h1> Hello World </h1>
}

export default App
```

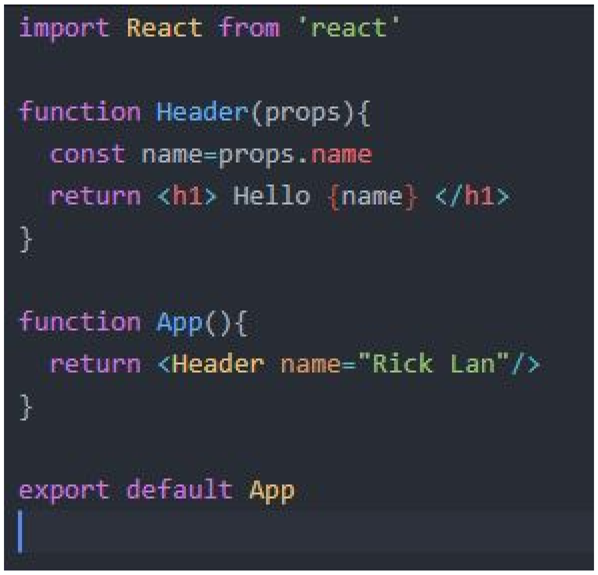


Hello World

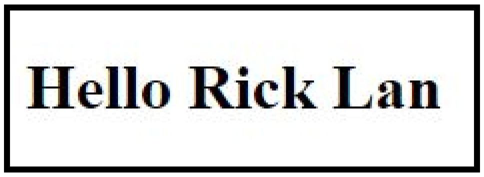
- While the above function does not take in any **props**, functions can take props. **Props**, which stand for properties, are arguments passed into React components. React Props

are like function arguments in JavaScript and attributes in HTML. To send props into a component, use the same syntax as HTML attributes. When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a prop.
Note: React Props are read-only. You will get an error if you try to change their value.

- E.g.



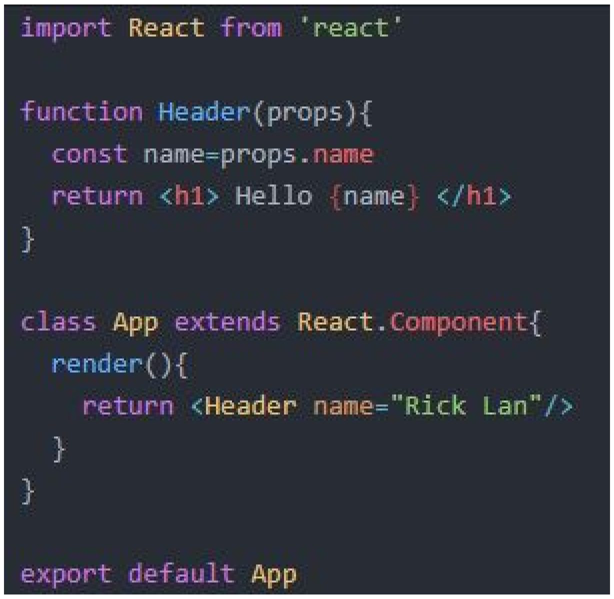
```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

function App(){
  return <Header name="Rick Lan"/>
}

export default App
```



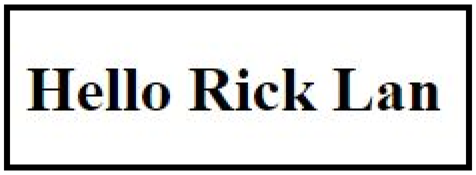
Hello Rick Lan

In this example, the function App is passing the props “name” to the function Header. This function is a valid React component because it accepts a single props object argument with data and returns a React element. We call such components **function components** because they are literally JavaScript functions.

- You can also use an ES6 class to define a component.
- E.g.



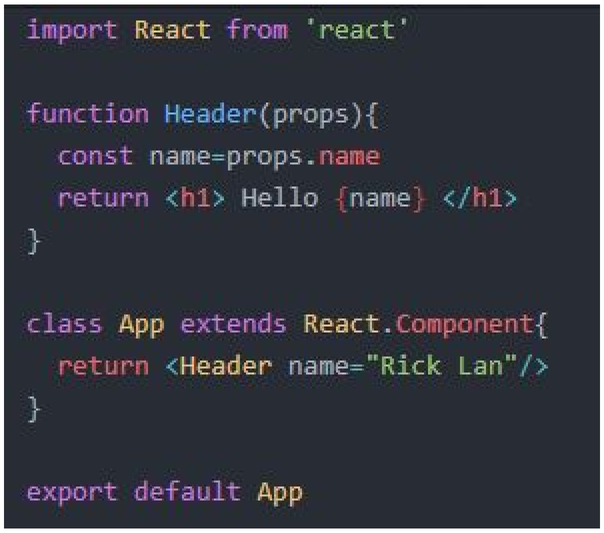
```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

class App extends React.Component{
  render(){
    return <Header name="Rick Lan"/>
  }
}

export default App
```



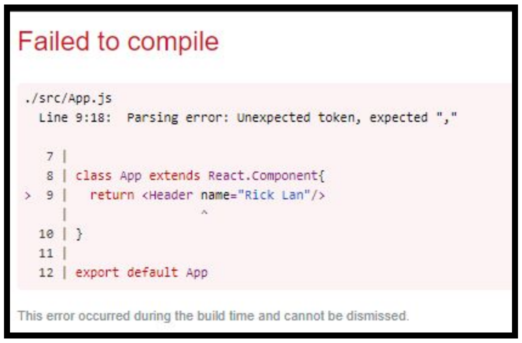
Hello Rick Lan

Notice that I changed App from **function App()** to **class App extends React.Component**, but visually, nothing changed.

- **Note:** If you're using the **class ___ extends React.Component** way, you need to use the **render()** method in it. The render() method is the method that actually outputs HTML to the DOM. If you don't have the render() method, you'll get an error. Furthermore, if you put the render() method inside a function, you'll get an error.
- E.g. Here, I removed the render() method from **class App extends React.Component** and I got an error.



```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

class App extends React.Component{
  return <Header name="Rick Lan"/>
}

export default App
```

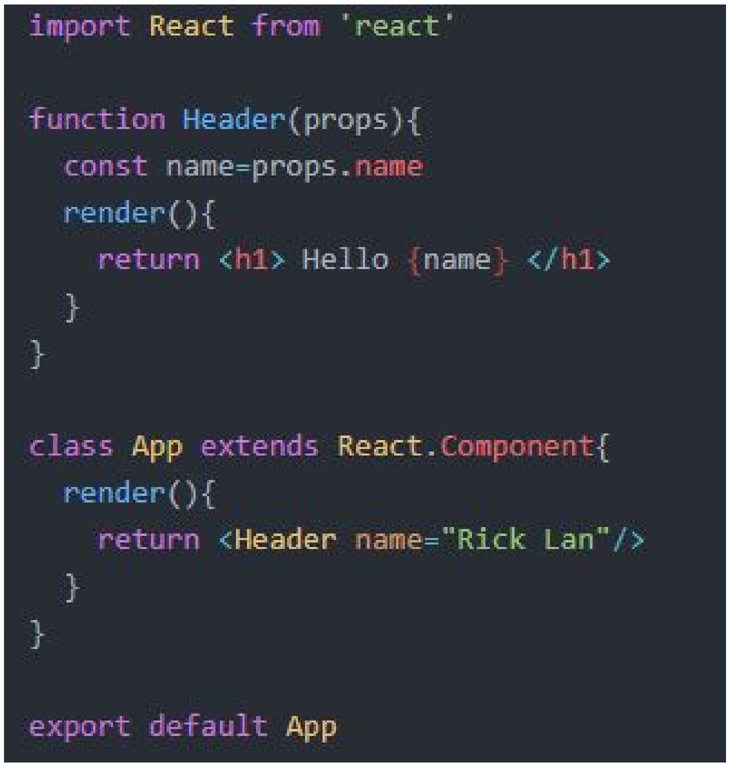


Failed to compile

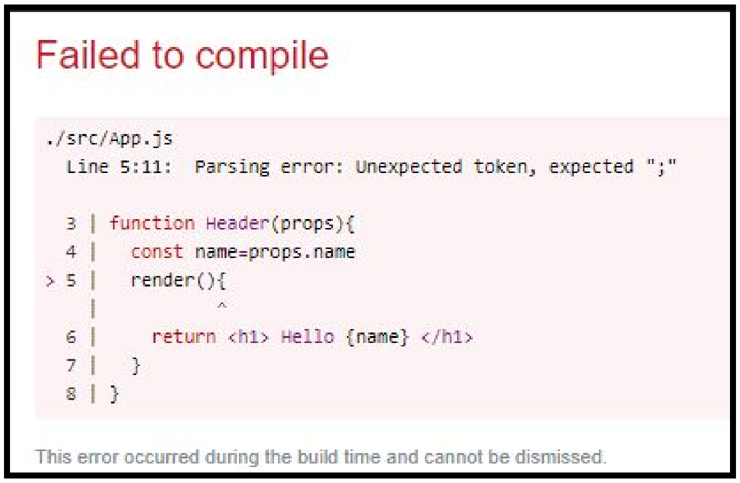
```
./src/App.js
  Line 9:18:  Parsing error: Unexpected token, expected ","

   7 |
   8 | class App extends React.Component{
>  9 |   return <Header name="Rick Lan"/>
     |                 ^
  10 | }
  11 |
  12 | export default App
```

This error occurred during the build time and cannot be dismissed.

- E.g. Here, I added the render() method to **function Header(props)** and I got an error.



```
import React from 'react'

function Header(props){
  const name=props.name
  render(){
    return <h1> Hello {name} </h1>
  }
}

class App extends React.Component{
  render(){
    return <Header name="Rick Lan"/>
  }
}

export default App
```

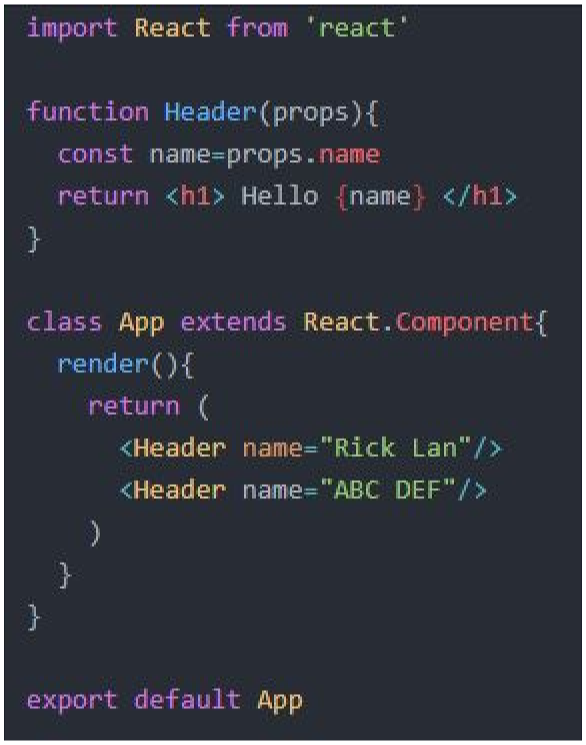


Failed to compile

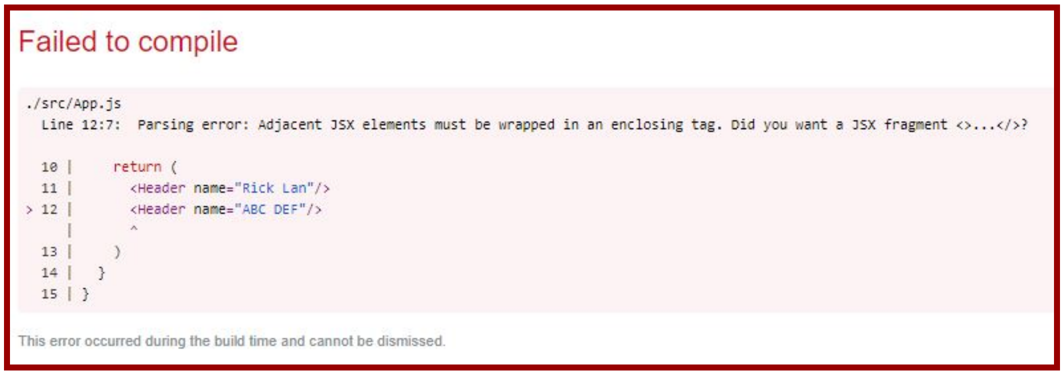
```
./src/App.js
  Line 5:11:  Parsing error: Unexpected token, expected ";"

  3 | function Header(props){
  4 |   const name=props.name
> 5 |   render(){
    |         ^
  6 |     return <h1> Hello {name} </h1>
  7 |   }
  8 | }
```

This error occurred during the build time and cannot be dismissed.

- **Note:** All components must return something. Furthermore, only 1 parent element can be returned.
- E.g. The below code creates an error because I am returning 2 parent elements.



```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

class App extends React.Component{
  render(){
    return (
      <Header name="Rick Lan"/>
      <Header name="ABC DEF"/>
    )
  }
}

export default App
```

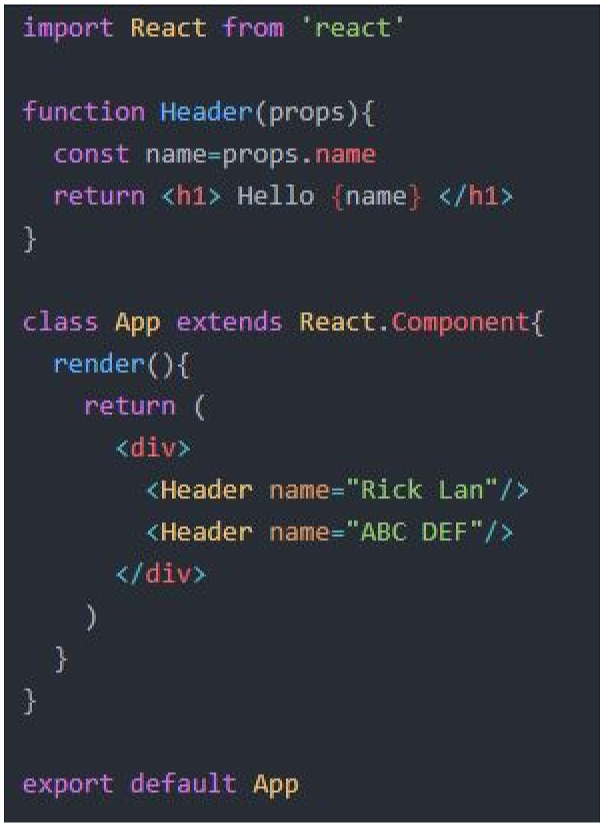


Failed to compile

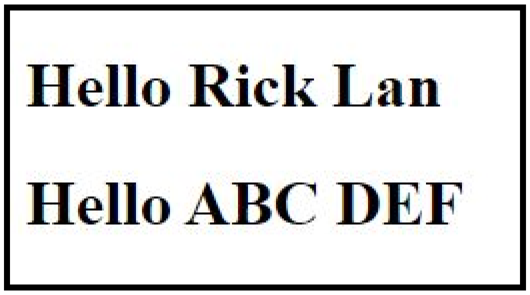
```
./src/App.js
  Line 12:7:  Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>?

  10 |     return (
  11 |       <Header name="Rick Lan"/>
> 12 |       <Header name="ABC DEF"/>
     |       ^
  13 |     )
  14 |   }
  15 | }
```

This error occurred during the build time and cannot be dismissed.

However, if I wrap the 2 lines in a <div> tag, then the error is resolved.



```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

class App extends React.Component{
  render(){
    return (
      <div>
        <Header name="Rick Lan"/>
        <Header name="ABC DEF"/>
      </div>
    )
  }
}

export default App
```



```
Hello Rick Lan

Hello ABC DEF
```

- The difference between component.render and DOM.render is the following:
- component.render only creates the virtual DOM. It does not add it to the actual browser DOM.
- ReactDOM.render does both. It creates or updates the virtual DOM, and then additionally adds it to the actual browser DOM.
- As shown, components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. As shown above, when App wants to use Header, it just does <Header /> and passes the props in.
- **Note:** Whether you declare a component as a function or a class, it must never modify its own props.
- Consider this sum function:
  ```
  function sum(a, b) {
    return a + b;
  }
  ```
  Such functions are called **pure** because they do not attempt to change their inputs, and always return the same result for the same inputs.
- In contrast, this function is impure because it changes its own input:
  ```
  function withdraw(account, amount) {
    account.total -= amount;
  }
  ```
- React is pretty flexible but it has a single strict rule: All React components must act like pure functions with respect to their props.

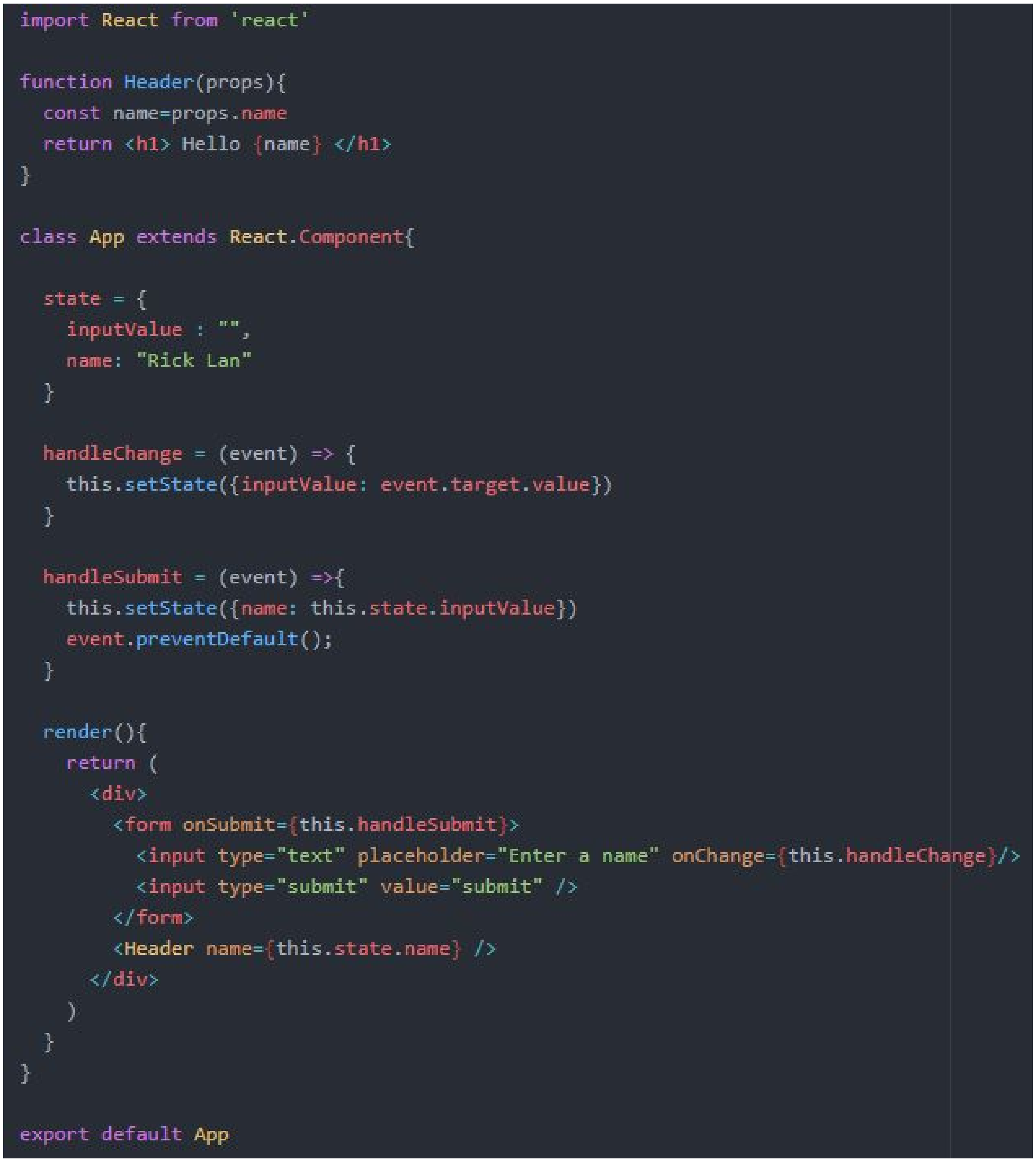
States:

- React components have a built-in state object.
- The state object is where you store property values that belong to the component. It determines how that component renders and behaves.
- When the state object changes, the component re-renders.
- Refer to the state object anywhere in the component by using the **this.state.propertyname** syntax.
- To change a value in the state object, use the **this.setState()** method.
- When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).
- Always use the setState() method to change the state object, it will ensure that the component knows its been updated and calls the render() method.
- **Note:** State Updates May Be Asynchronous. React may batch multiple setState() calls into a single update for performance.

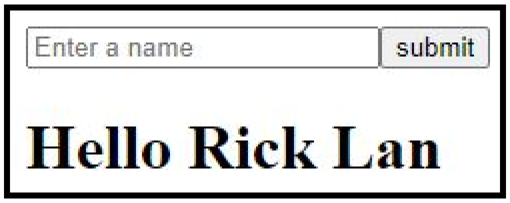
Events:

- Just like HTML, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.
- React events are named using camelCase, rather than lowercase.
  E.g. onClick instead of onclick.
- With JSX you pass a function as the event handler, rather than a string.
  I.e. React event handlers are written inside curly braces.
  E.g. **onClick={shoot}** instead of **onClick="shoot()"**.
- In React, to stop an event from happening, you can call the preventDefault function.

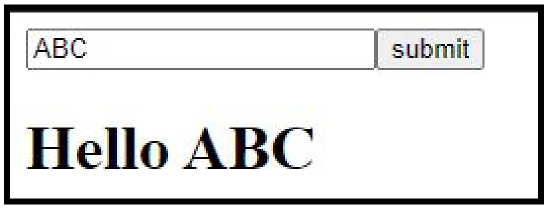
- This example demonstrates both state and events.



```
import React from 'react'

function Header(props){
  const name=props.name
  return <h1> Hello {name} </h1>
}

class App extends React.Component{

  state = {
    inputValue : "",
    name: "Rick Lan"
  }

  handleChange = (event) => {
    this.setState({inputValue: event.target.value})
  }

  handleSubmit = (event) =>{
    this.setState({name: this.state.inputValue})
    event.preventDefault();
  }

  render(){
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <input type="text" placeholder="Enter a name" onChange={this.handleChange}/>
          <input type="submit" value="submit" />
        </form>
        <Header name={this.state.name} />
      </div>
    )
  }
}

export default App
```



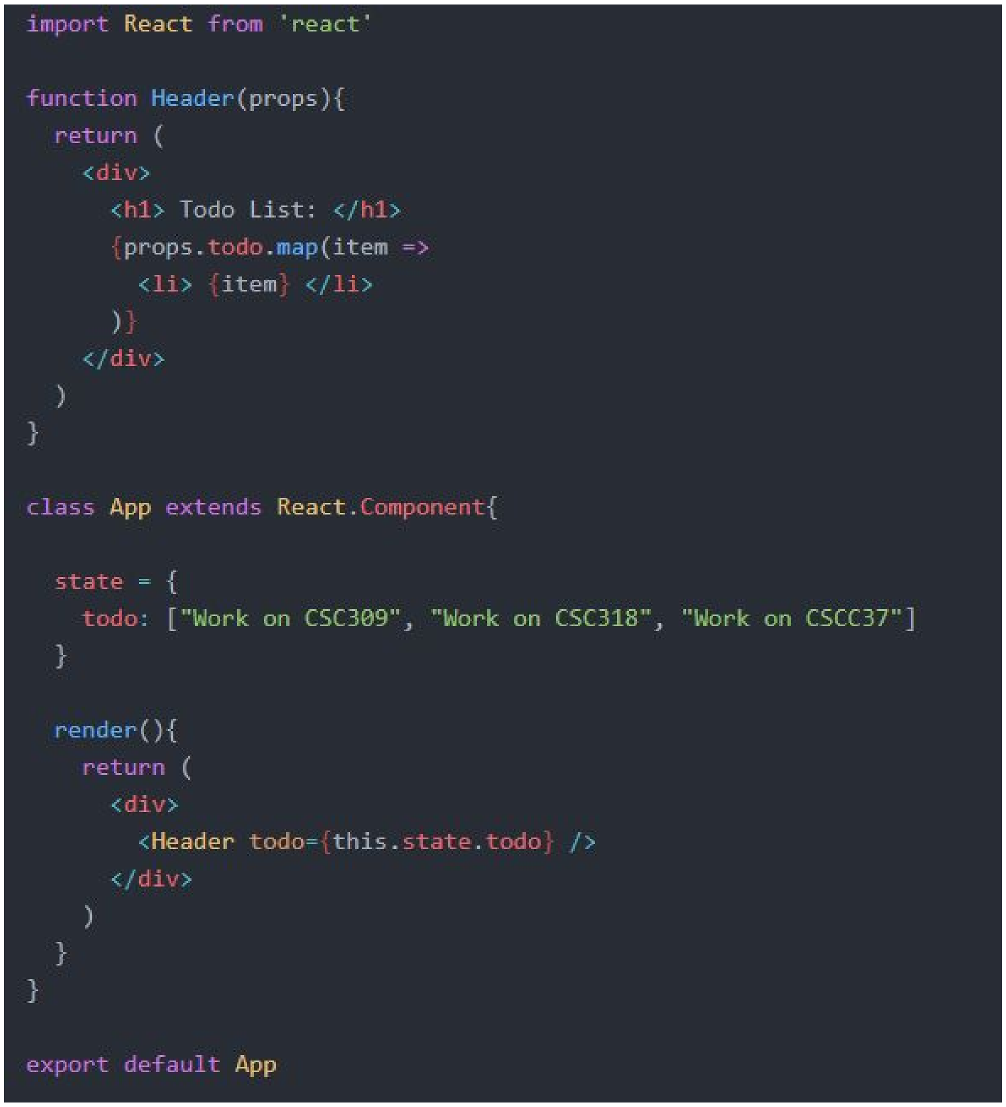
Hello Rick Lan



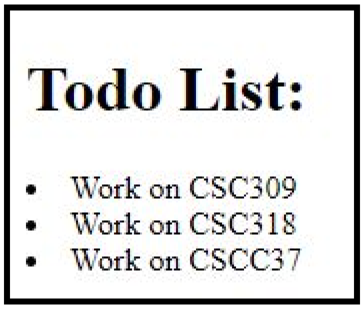
Hello ABC

Map:

- When you call **map()** on an array, you can have it run through all the items in that array and do something interesting with them.
- E.g.



```
import React from 'react'

function Header(props){
  return (
    <div>
      <h1> Todo List: </h1>
      {props.todo.map(item =>
        <li> {item} </li>
      )}
    </div>
  )
}

class App extends React.Component{

  state = {
    todo: ["Work on CSC309", "Work on CSC318", "Work on CSCC37"]
  }

  render(){
    return (
      <div>
        <Header todo={this.state.todo} />
      </div>
    )
  }
}

export default App
```



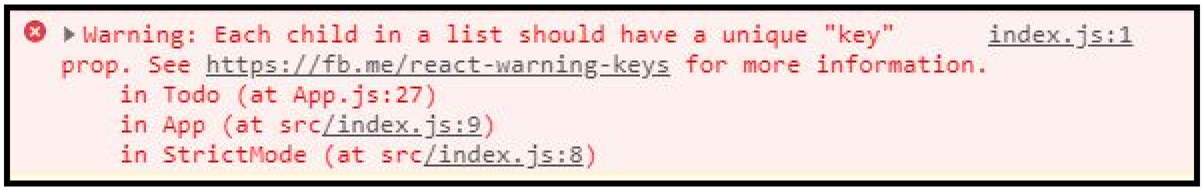
```
Todo List:

•  Work on CSC309
•  Work on CSC318
•  Work on CSCC37
```

Keys:

- Keys help React identify which items have changed, are added, or are removed.
- React uses a virtual DOM and only paints whatever is needed. It doesn't repaint the entire tree, like how a browser does.
- Consider this code:

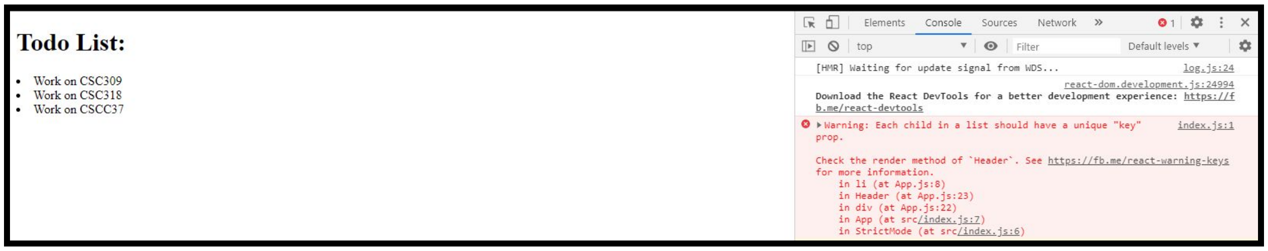
  ```
  <ul>
     <li> … </li>
     <li> … </li>
     ...
  </ul>
  ```

  It can be hard to differentiate between the <li>'s. To fix this issue, we can give each <li> a unique key.
- You can either create the key values yourself or have it auto generated.
- The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys. You can also use react-uid to generate your keys.
- If you don't put key values, it won't break the code, but you will see this error message in your console:



```
▶Warning: Each child in a list should have a unique "key"      index.js:1
prop. See https://fb.me/react-warning-keys for more information.
    in Todo (at App.js:27)
    in App (at src/index.js:9)
    in StrictMode (at src/index.js:8)
```

- In the example for map above, you see the error because I didn't give a key for the todo array.

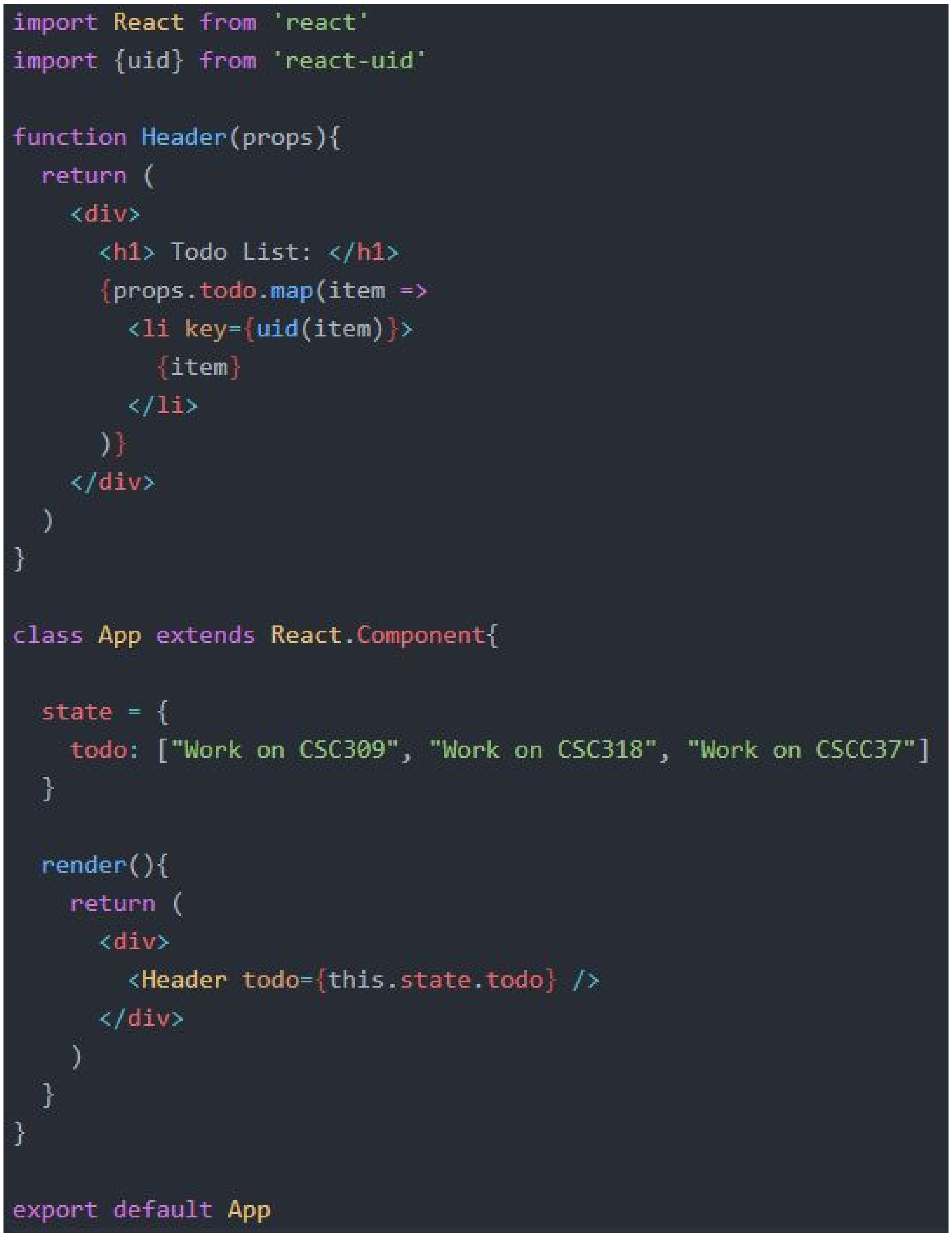


Todo List:

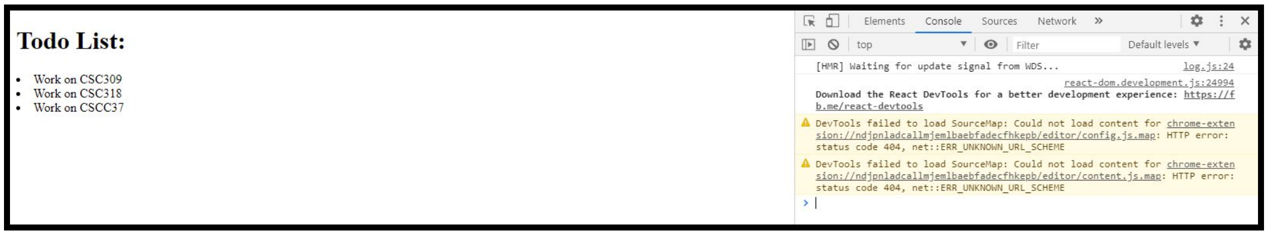
- Work on CSC309
- Work on CSC318
- Work on CSCC37

```
[HMR] Waiting for update signal from WDS...                     log.js:24
                                        react-dom.development.js:24994
Download the React DevTools for a better development experience: https://f
b.me/react-devtools
▶Warning: Each child in a list should have a unique "key"        index.js:1
prop.

Check the render method of `Header`. See https://fb.me/react-warning-keys
for more information.
    in li (at App.js:8)
    in Header (at App.js:23)
    in div (at App.js:22)
    in App (at src/index.js:7)
    in StrictMode (at src/index.js:6)
```

- However, if I used react-uid, as shown below, the error goes away.



```
import React from 'react'
import {uid} from 'react-uid'

function Header(props){
  return (
    <div>
      <h1> Todo List: </h1>
      {props.todo.map(item =>
        <li key={uid(item)}>
          {item}
        </li>
      )}
    </div>
  )
}

class App extends React.Component{

  state = {
    todo: ["Work on CSC309", "Work on CSC318", "Work on CSCC37"]
  }

  render(){
    return (
      <div>
        <Header todo={this.state.todo} />
      </div>
    )
  }
}

export default App
```



Todo List:

- Work on CSC309
- Work on CSC318
- Work on CSCC37

```
[HMR] Waiting for update signal from WDS...                          log.js:24
                                           react-dom.development.js:24994
Download the React DevTools for a better development experience: https://f
b.me/react-devtools
DevTools failed to load SourceMap: Could not load content for chrome-exten
sion://ndjpnladcallmjemlbaebfadecfhkepb/editor/config.js.map: HTTP error:
status code 404, net::ERR_UNKNOWN_URL_SCHEME
DevTools failed to load SourceMap: Could not load content for chrome-exten
sion://ndjpnladcallmjemlbaebfadecfhkepb/editor/content.js.map: HTTP error:
status code 404, net::ERR_UNKNOWN_URL_SCHEME
>
```