Schedule

Today:

- Recall: React getting started
  - React app - folder structure

- ReactJS:
  - ES6 arrow function
  - JSX
  - React Components

Recall: Introduction to ReactJS

What is react

- React is a **JavaScript** library created by **Facebook**
- React is a **User Interface** (UI) library
- React is a tool for building **UI components**

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

class Test extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}

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

Why React?

Problems solved by react:

- DOM operations are quire expensive in terms of **performance**
- Page has data changes over time at high rates
    - Lots of people commenting on a post
    - Likes being generated…

→ require DOM to:

- updates **very fast**,
- reflect in other parts of UI if they use the same data

How React works?

React creates a VIRTUAL DOM in memory.

- Instead of manipulating the browser's DOM directly,

- React creates a virtual DOM in memory

→ does all the necessary manipulating

→ making the changes in the browser DOM.

React only changes what needs to be changed!

- React finds out what changes have been made, and changes **only** what needs to be changed.

- You will learn the various aspects of how React does this later.

Any others?

So why React?

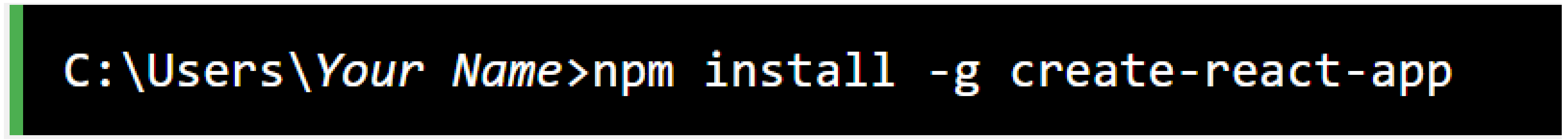
- Opionated

Handlebars for frontend?

- Sure, YES https://handlebarsjs.com/

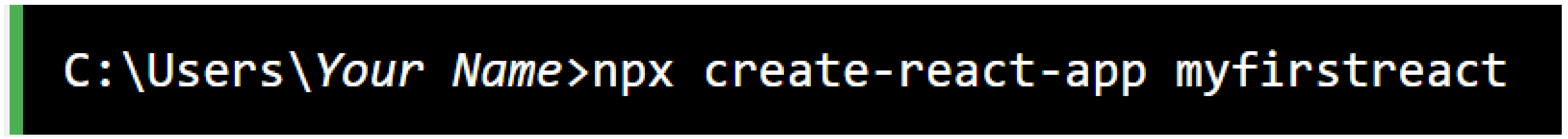
Setting react

- Install create-react-app by running this command in your terminal:



```
C:\Users\Your Name>npm install -g create-react-app
```

- Then you are able to create a React application, let's create one called *myfirstreact*.



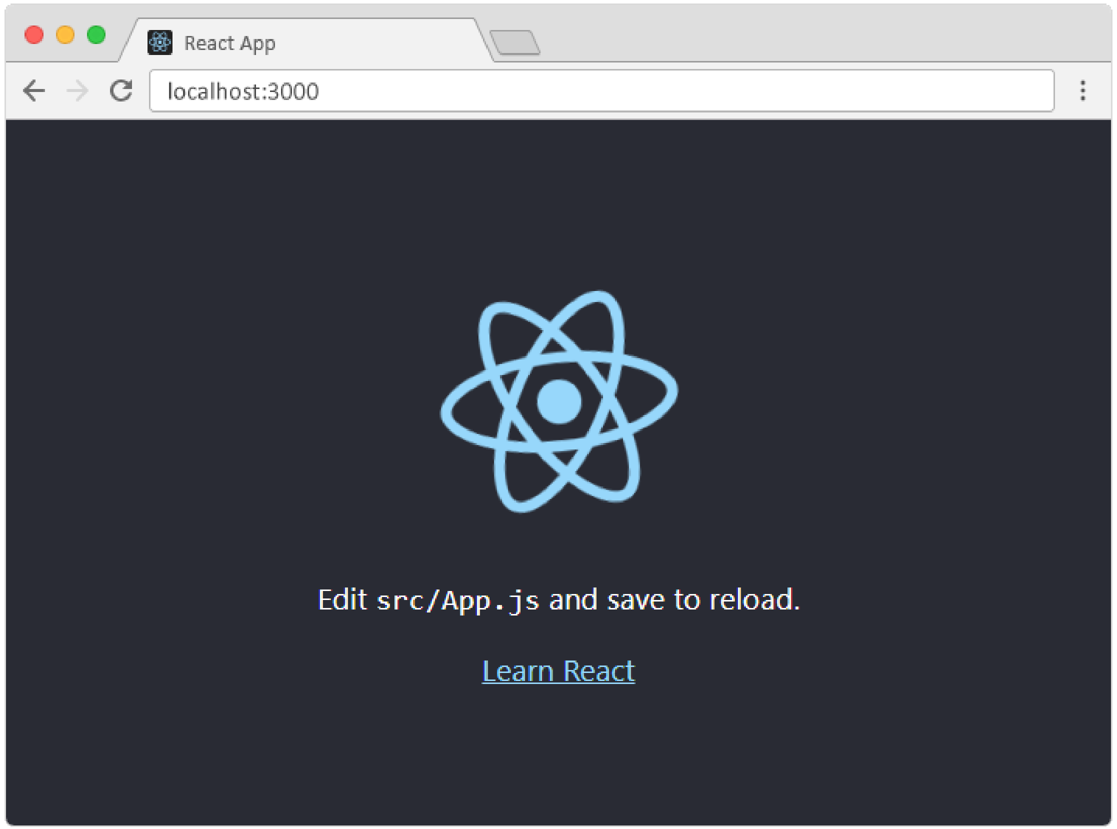
```
C:\Users\Your Name>npx create-react-app myfirstreact
```

Running react

- Move to the *myfirstreact* directory & run application

```
C:\Users\Your Name>cd myfirstreact
```

```
C:\Users\Your Name\myfirstreact>npm start
```



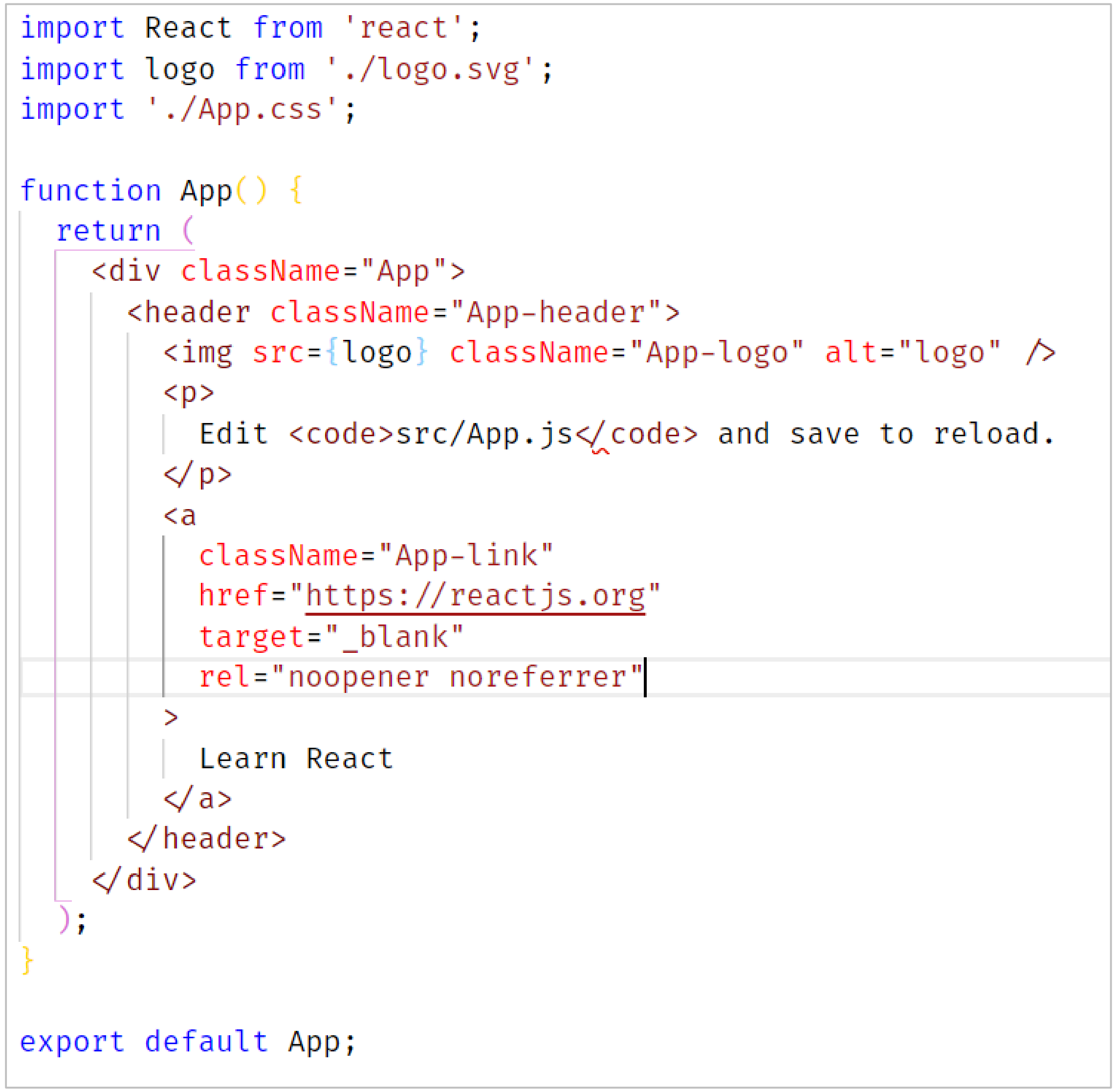
Edit `src/App.js` and save to reload.

Learn React

React app – folder structure

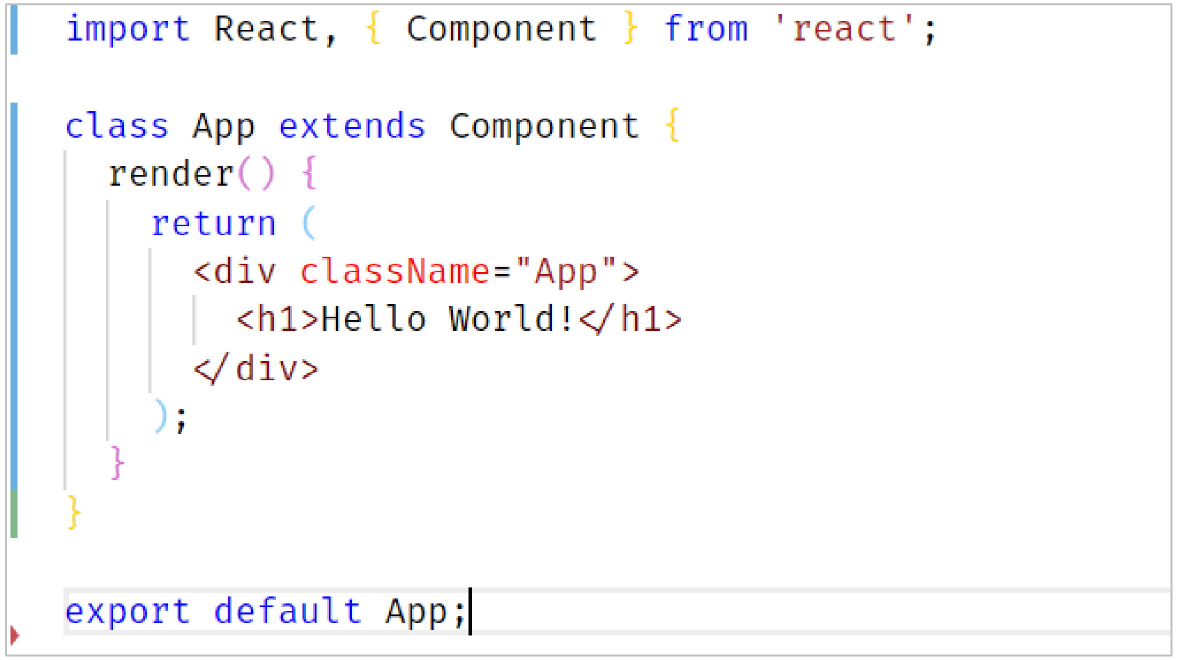
The default `src/App.js`

- Now modify to print "Hello World!"

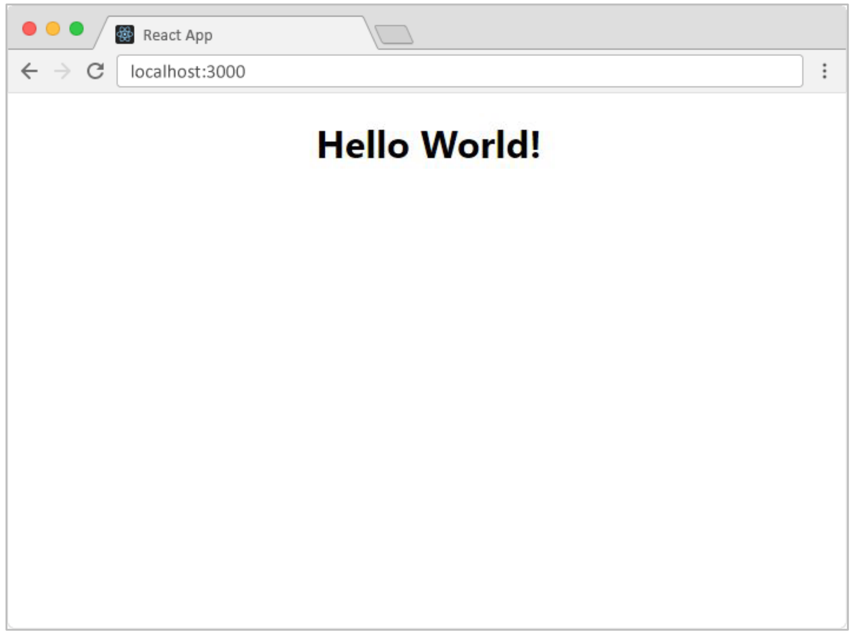


```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

The default `src/App.js`



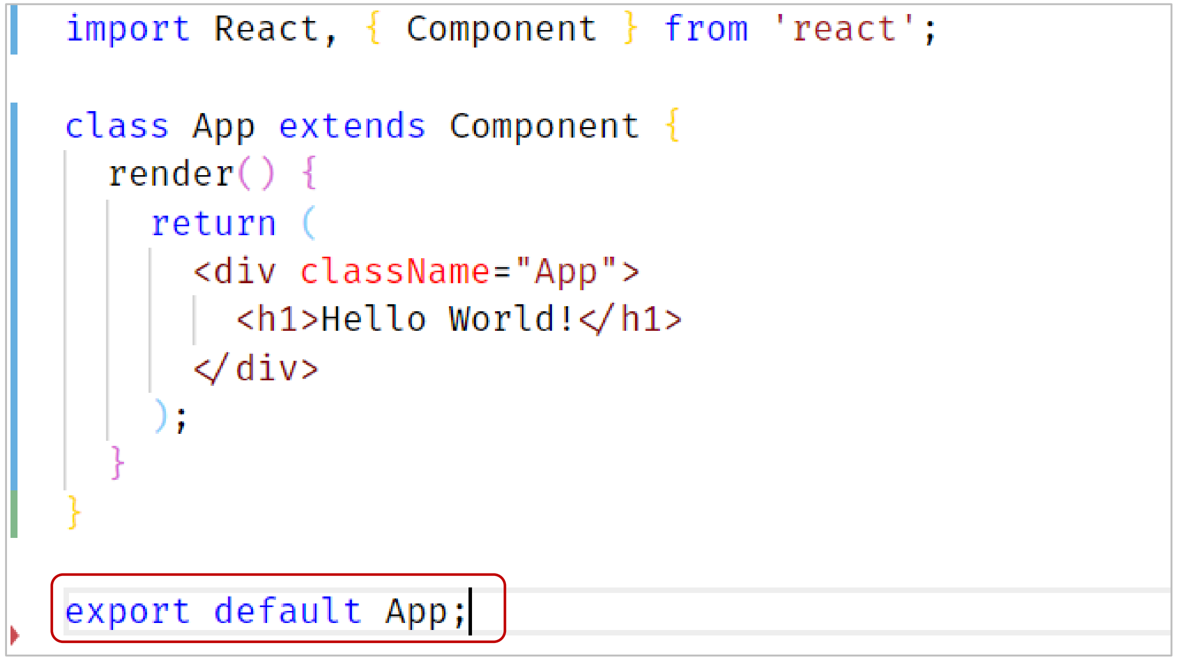
```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello World!</h1>
      </div>
    );
  }
}

export default App;
```



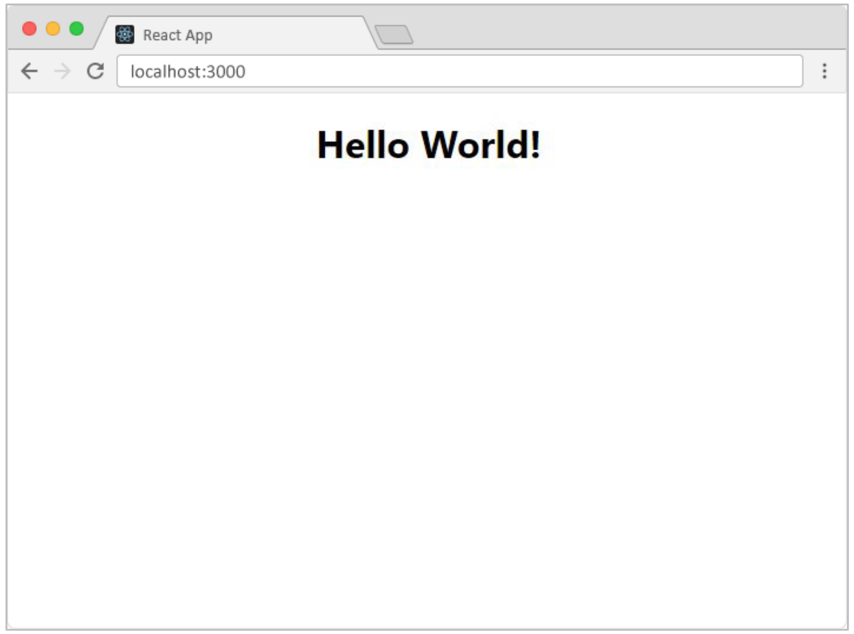
Hello World!

- The HTML like syntax called **JSX** (we will mention it in some later slides)
- Just like *nodemon*, the changes is visible immediately after you save the file, you do not have to reload the browser!

The default `src/App.js`



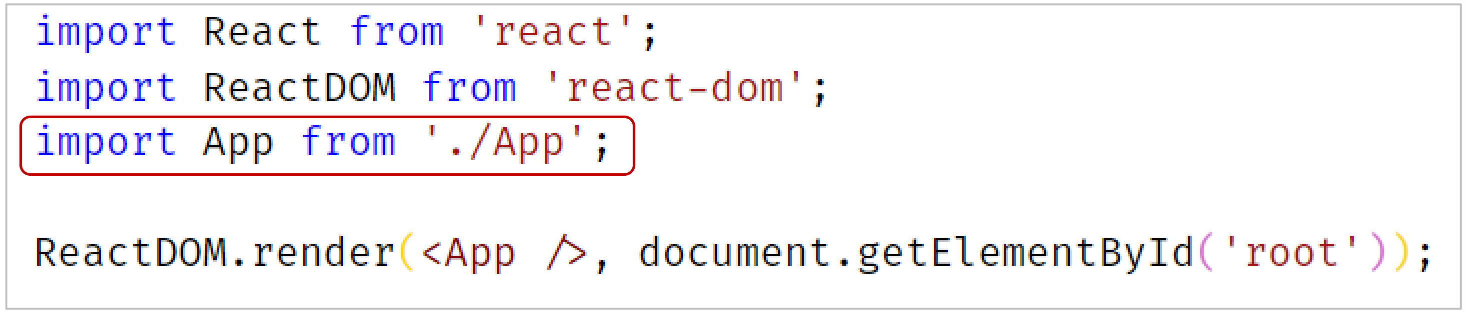
```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello World!</h1>
      </div>
    );
  }
}

export default App;
```



Hello World!

- export default?
  - ES6 module (just like Node module)
  - ~~`require()`~~ → `import … from …`
- `Export not default?` [`difference`]

src/index.js



```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

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

- Another simple version (deleted un-used lines of code)
- import App
- render App into **‘root’**?

public/index.html

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>React App</title>
</head>

<body>

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

</body>

</html>
```

- **The root node**: a *container* for content managed by React.
- Not ~~`<div>`~~ & `id !=`“root”?
  → Any tag & id

React app – folder structure

- **build**: final, production-ready build (wont exist until `npm build`)
- **node_modules**: packages by *npm* or *yarn*
- **public**: static files,
  - NOT imported by app &
  - must maintain its file name (images, *index.html*…)

  → Cached by browser, never download again
- **src**: dynamic files
  - Imported by app
  - Change contents

  → Never worry about the browser using outdated copy

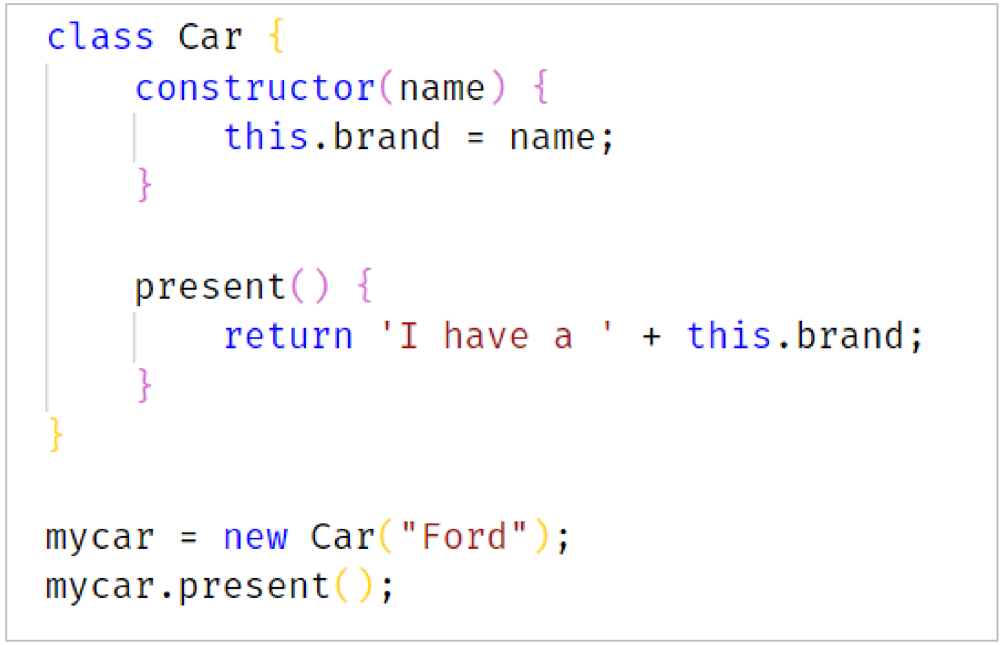
The default `src/App.js`

ES6 arrow function

Recall: ES6

- ES6?
    - ECMAScript 6 (ECMAScript 2015) – standard of JavaScript

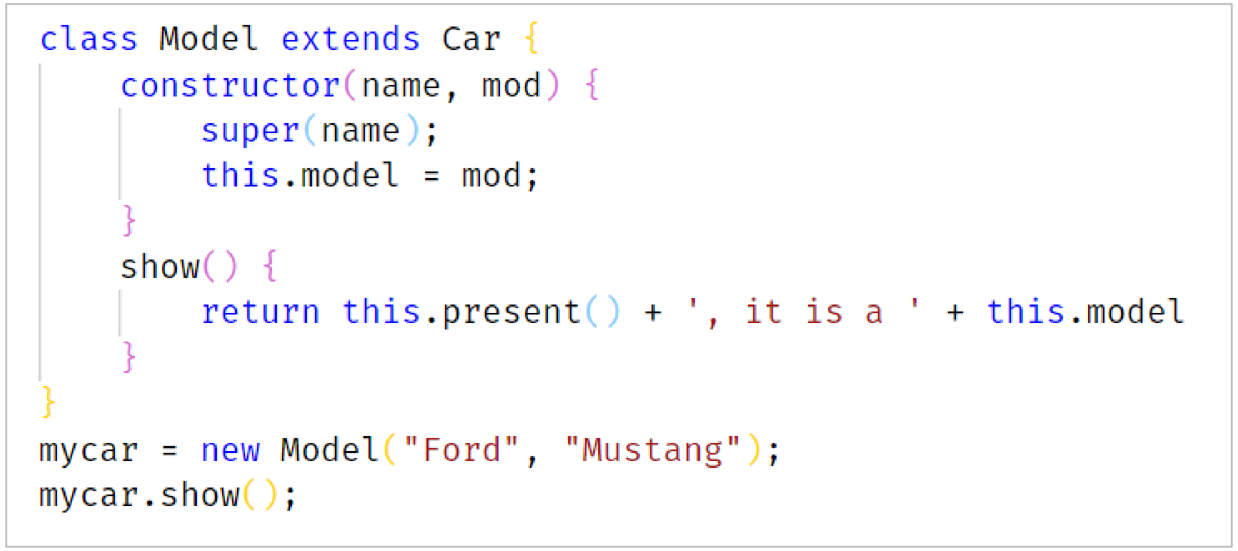
- Class in ES6:



```
class Car {
    constructor(name) {
        this.brand = name;
    }

    present() {
        return 'I have a ' + this.brand;
    }
}

mycar = new Car("Ford");
mycar.present();
```

ES6 Class inheritance

- Class inheritance:
  - Keyword: `extends`
  - Child class inherits all the methods from the parent (super) class



```
class Model extends Car {
    constructor(name, mod) {
        super(name);
        this.model = mod;
    }
    show() {
        return this.present() + ', it is a ' + this.model
    }
}
mycar = new Model("Ford", "Mustang");
mycar.show();
```

- **constructor**: MUST invoke `super()` – constructor of parent class

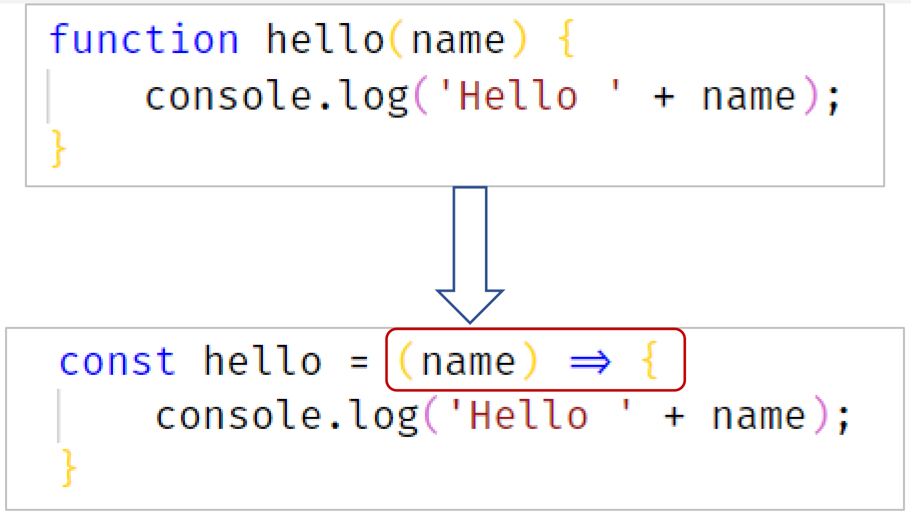
→ Get access to parent properties

- Use `this` to access parent's properties & methods

ES6 arrow function

- Shorter syntax

```
(param1, param2, … paramN) => {
     // statements
}
```



```
function hello(name) {
    console.log('Hello ' + name);
}
```

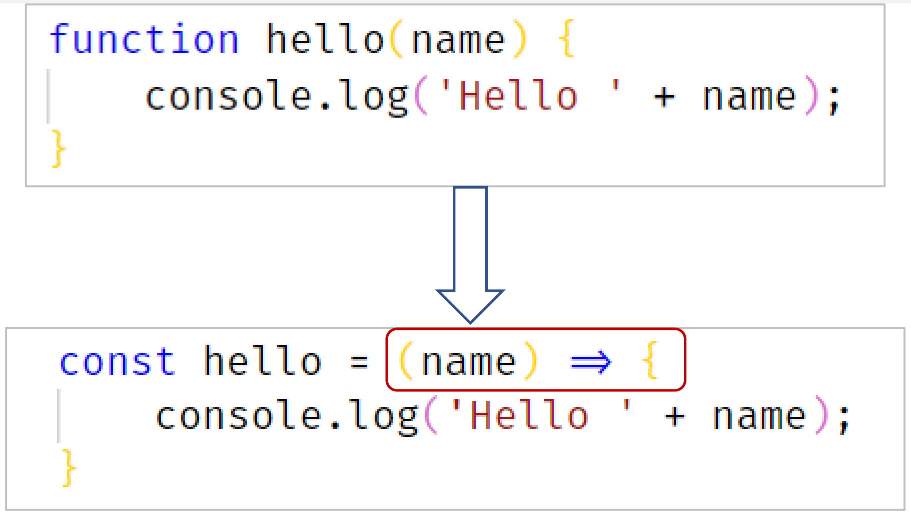
```
const hello = (name) => {
    console.log('Hello ' + name);
}
```

Problem with this?

ES6 arrow function

- Shorter syntax

```
(param1, param2, … paramN) => {
     // statements
}
```



```
function hello(name) {
    console.log('Hello ' + name);
}
```

```
const hello = (name) ⇒ {
    console.log('Hello ' + name);
}
```

Problem with this?

- **regular functions**: `this` = *the object that called the function,*
  - the window, the document, a button or whatever

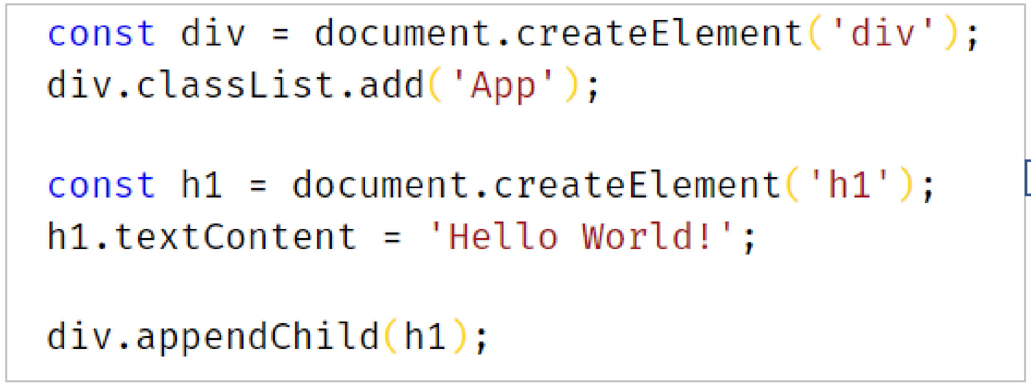
→ `bind()`

- **arrow functions**: `this` always = *the object that defined the arrow function*

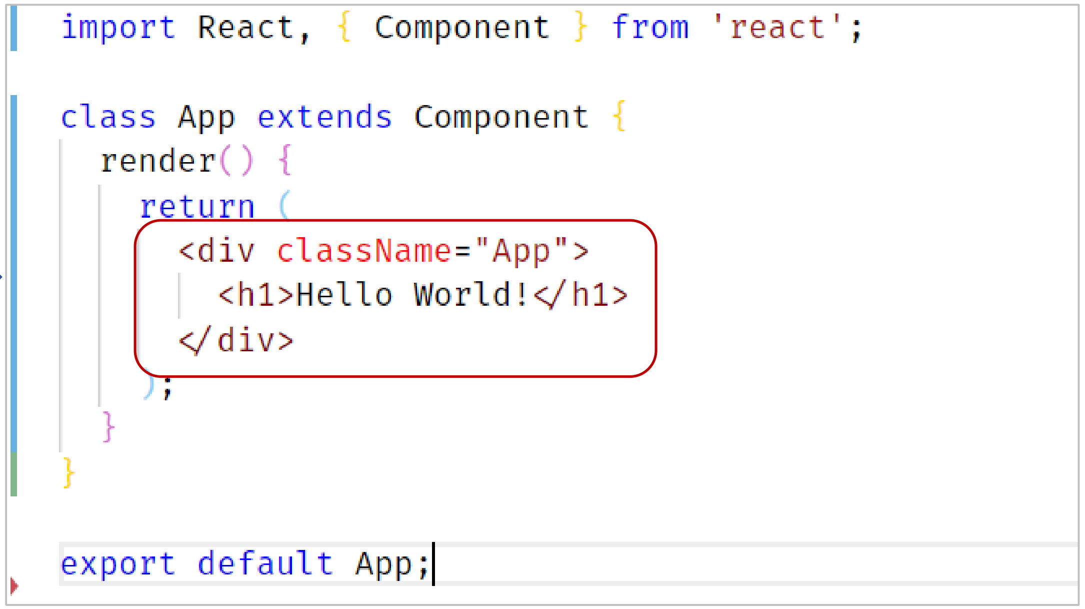
React JSX

JSX?

- JavaScript XML
- HTML in JavaScript
- Easier to write and add HTML in React



```
const div = document.createElement('div');
div.classList.add('App');

const h1 = document.createElement('h1');
h1.textContent = 'Hello World!';

div.appendChild(h1);
```



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

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello World!</h1>
      </div>
    );
  }
}

export default App;
```

Coding JSX

- JSX allow 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.

You are not required to use JSX, but .. why not? :D

JSX syntax

- Expressions:
  - Are written inside {}
  - Expression can be variable, property or any valid JS expression

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

- Multiple lines HTML:
  - Put inside ()

```
const myelement = (
    <ul>
        <li>Apples</li>
        <li>Bananas</li>
        <li>Cherries</li>
    </ul>
);
```

JSX syntax

- **Note**: One top level element
    - The HTML code MUST be wrapped in ONE top level element

e.g. wrap 2 headers inside one DIV element

```
const myelement = (
    <div>
        <h1>I am a Header.</h1>
        <h1>I am a Header too.</h1>
    </div>
);
```

- **Note**: Elements Must be Closed
    - JSX follows XML rules → HTML elements MUST be properly closed
    - Close empty elements with />

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

React components

- independent and reusable bits of code
- are like functions that return HTML via `render()`

- *2 types of component:*
    - Class component
    - Function component

Create a component

- Class Component
  - Component name MUST start with an uppercase letter
  - `extends React.Component`
  - Require `render()` method & this method MUST return HTML

```
class Car extends React.Component {
    render() {
        return <h2>I am a Car!</h2>;
    }
}
```

- Use a ***Class component***:
  - Similar syntax as normal HTML

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

Create a component

- Function Component
  - Component name MUST start with an uppercase letter
  - MUST return HTML
  - Behave similar to Class component but Class has *some additions*

```
function Car() {
    return <h2>Hi, I am also a Car!</h2>;
}
```

- Use a ***Function component:***
  - Similar syntax as normal HTML

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

Component Constructor

- Called when the component gets initiated
  - initiate the **component’s properties**
  - inherit parent component super()
- In React, component’s properties should be kept in an object called **state**

  e.g. add color property & use it in render()

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Props

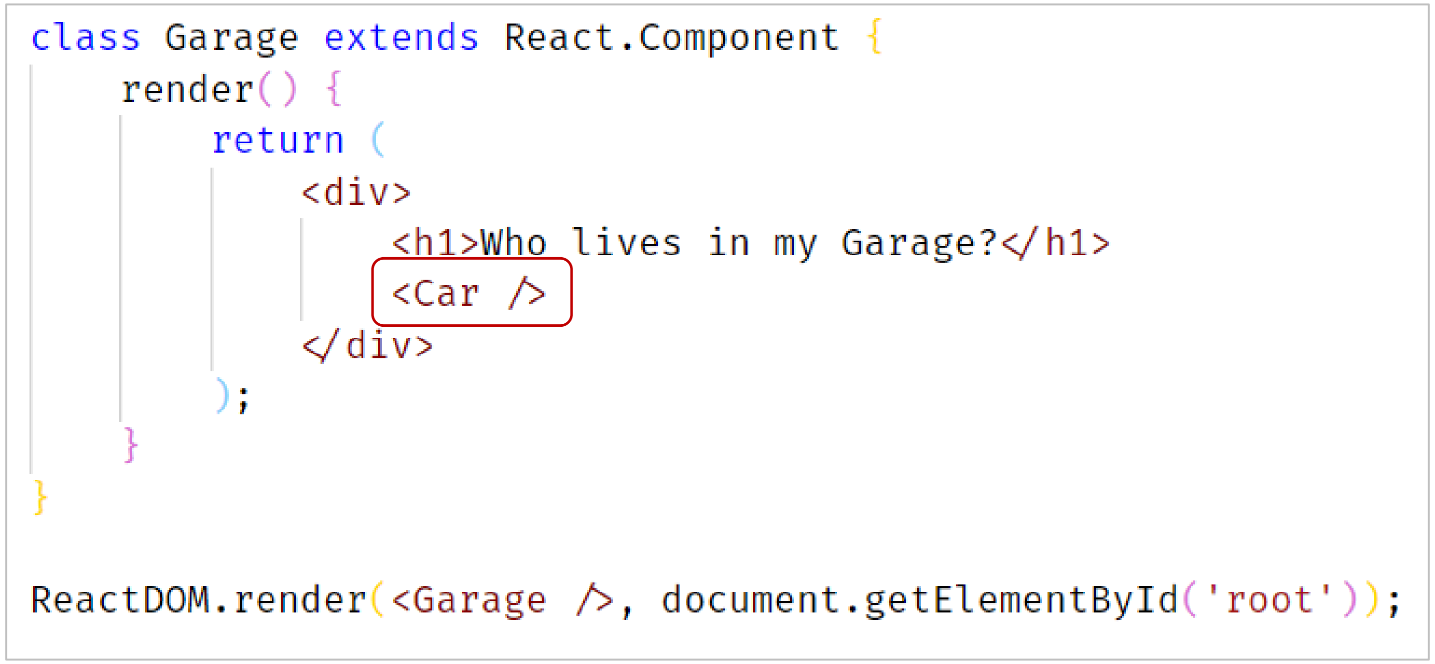
- Another way of handling component properties
- Props = function arguments
  - passed into the component as attributes.

e.g. pass a color to the Car component & use it in render()

```
class Car extends React.Component {
    render() {
        return <h2>I am a {this.props.color} Car!</h2>;
    }
}

ReactDOM.render(<Car color="red" />, document.getElementById('root'));
```

Components in Components

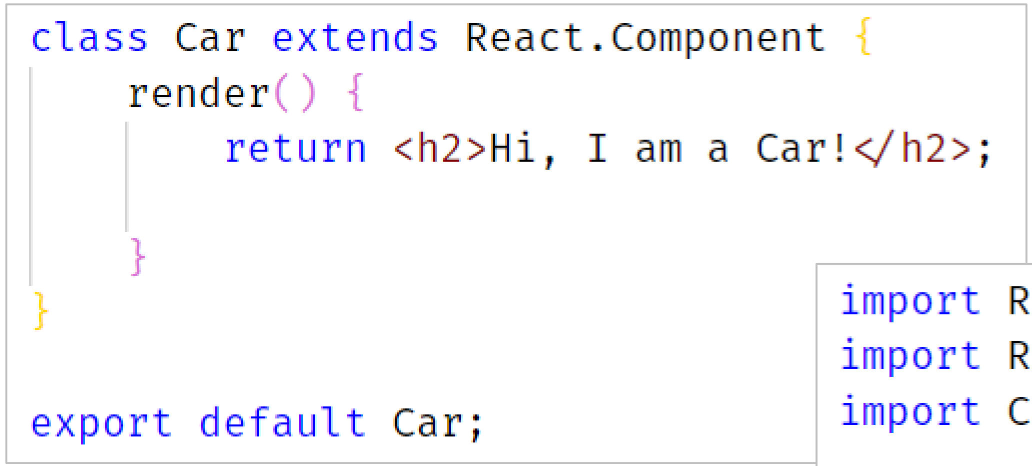
- Refer to components inside other components



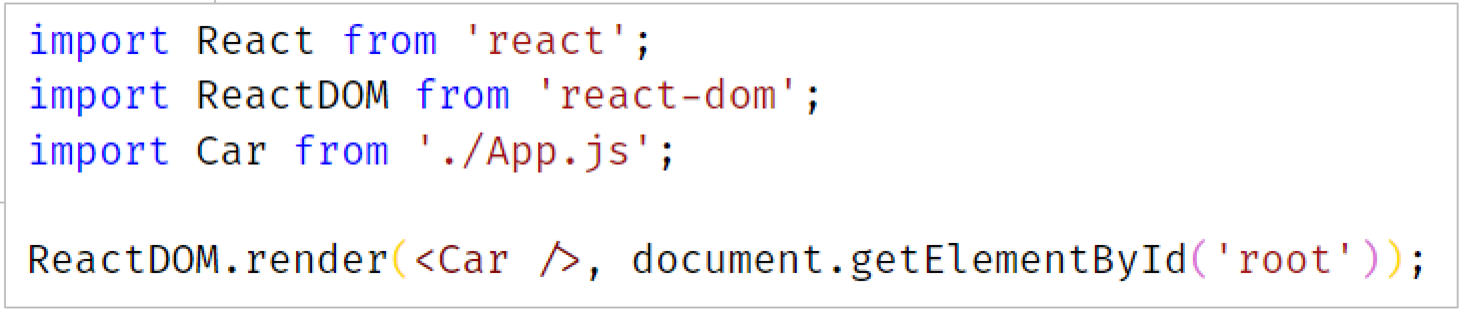
```
class Garage extends React.Component {
    render() {
        return (
            <div>
                <h1>Who lives in my Garage?</h1>
                <Car />
            </div>
        );
    }
}

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

Components in Files

- React is all about re-using code
  - be smart to insert some of your components in separate files.
- Create a new `.js` file and put the code inside it:
- **Note**: the file
  - MUST start by importing React (as before),
  - HAS TO end with the statement `export default Car;`.



```
class Car extends React.Component {
    render() {
        return <h2>Hi, I am a Car!</h2>;

    }
}

export default Car;
```



```
import React from 'react';
import ReactDOM from 'react-dom';
import Car from './App.js';

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

More next week!