REACT

Render element in index.js with:

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

JSX

One top element

The HTML code must be wrapped in ONE top level element.

So if you like to write two headers, you must put them inside a parent element, like `div` element

Element must be closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

React Component

Create a Class Component

When creating a React component, the component's name must start with an upper case letter.

The component has to include the `extends React.Component` statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a `render()` method, this method returns HTML.

Example

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

Component Constructor

If there is a `constructor()` function in your component, this function will be called when the component gets initiated.

The constructor function is where you initiate the component's properties.

In React, component properties should be kept in an object called `state`.

You will learn more about `state` later in this tutorial.

The constructor function is also where you honor the inheritance of the parent component by including the `super()` statement, which executes the parent component's constructor function, and your component has access to all the functions of the parent component (`React.Component`).

Example

```
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 is by using **props**.

Props are like function arguments, and you send them into the component as attributes.

You will learn more about **props** in the next chapter.

```
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

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

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, and it can be smart to insert some of your components in separate files.

To do that, create a new file with a .js file extension and put the code inside it:

Note that the file must start by importing React (as before), and it has to end with the statement export default Car;.

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

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


}
}

export default Car;
```

Pass data

Props are also how you pass data from one component to another, as parameters.

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

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

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

If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

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

class Garage extends React.Component {
  render() {
    const carname = "Ford";
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carname} />
      </div>
    );
  }
}
```

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