Framework Training
React
London July 2018
Exercise A-intro

- This exercise introduces **React components** which combine markup, logic, style and state.
- Components are a fundamental building block of the React framework.

Installation

- Install and run the starter version of the project.

```
npm install
npm start
```

React components

- Components allow you to create custom HTML elements which combine markup, logic, style and state.
- This exercise builds a very simple React component.
- Review the starter files.
- **public/index.html** is an HTML file containing two empty sections.

```
  <section class="spain"></section>
  <section class="norway"></section>
```

- We will use React to create components and inject markup inside these sections.
- The entry point for our project is **index.js**. We will create instances of a React component here that are rendered in the HTML sections above.

Define a City component

- We will define a simple component which displays a message

```
Welcome to the city of Seville.
```

- The name of the city will be passed in to the component as an argument (prop).
- Open **src/City.js**
- Import the React library and its Component class.

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

- Import the local stylesheet City.css

```
import "./City.css";
```

- A React component can be defined as an ES6 class.
- The class extends the React Component class.

```
class City extends Component {}
```

- Every component must have a **render** method.
- This method defines the view/UI of the component.
- The render method returns markup, written in **JSX**, the React HTML-like dialect.

```
render() {
    return <section>Hello</section>
    }
```

- We have now created a **definition** of a component.
- Export the component so that it is useable in other modules.

```
export default City;
```

Component instances

- To be visible on a web page, we need to create **instances** of this component.
- File **src/index.js** is the entry point for our React project where we will create component instances.
- React has been separated into two libraries.
- The core library is React.
- React-DOM handles rendering React code in the browser DOM.
- This separation allows React to also be deployed to browser-less server environments.

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

- We need to import the City component, and any local CSS.

```
import "./index.css";
import City from "./City";
```

- We can now use React-DOM to render one or more **instances** of the City component on the page.

```
ReactDOM.render(<City/>, document.querySelector(".spain"));
```

- Hello should now appear on the web page.

Component props

- We will extend the component to pass in the city name as an argument.
- To allow multiple lines of JSX in the return statement, we will wrap the code in parentheses.

```
    render() {
        return (
            <section>Hello</section>
        )
    }
```

- Change the markup to include a SPAN which can be styled.

```
<section>Welcome to the city of<span>name</span></section>
```

- We can pass a city name into the component instance as an attribute in **index.js**.

```
<City name="Seville"/>
```

- This argument is referred to as a **prop** in React.
- Inside the component **this.props** is in scope as an object containing the name/value pairs of the arguments passed in.

```
console.log(this.props);
// {name: "Seville"}
```

- We can use the expression **this.props.name** to display the city name.

```
<span>{ this.props.name }</span>
```

Multiple instances

- We can render **multiple instances** of a component.

```
let spain = document.querySelector(".spain");
let norway = document.querySelector(".norway");

ReactDOM.render(<City name="Seville" />, spain);
ReactDOM.render(<City name="Oslo" />, norway);
```

Different types of component syntax

- There are a number of alternative approaches to writing React components.
- The render method can be written as an **ES6 arrow** function.

```
    render = () => <section>Welcome to the city of<span>
{this.props.name}</span></section>
```

- *Later in the course we will review the scope advantages of using arrow functions.*

Stateless components

- The City component does not have any internal **state**.
- In this case we can rewrite the component more simple as a stateless function.

```
function City(props) {
    return (
        <section>Welcome to the city of<span>{ props.name }
</span></section>
    )
}
```

- Note two changes: we have to explicitly pass in props, and we refer to them without using this.
- We can refactor this component using ES6 arrow syntax.

```
let City = props => <section>Welcome to the city of<span>{
props.name }</span></section>
```

- All four variations of the component do the same thing.
- We will review use cases for when to use each approach later in the course.