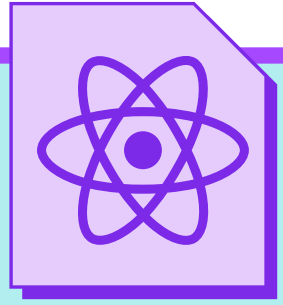
Intro to React



Presented by
Tom Isles

Motivation

Why don't we build websites just using Javascript?

Because it's complicated. It takes a lot of code to do simple things, it's verbose, and as your codebase grows, it becomes harder to keep in your head. Add more developers and this becomes a recipe for disaster!

What is React?

React is a library for building user interfaces.

It allows you to build isolated UI components in a simple, declarative way. When the state of your application changes, affected components will ***react*** accordingly by re-rendering to reflect the new state.

This is great, because it means you don't have to worry about what's happening under the hood. You can change the state, and the web page will "magically" update for you!

How does it work?

A React application is defined as a tree structure. It is made up of components, which are defined in simple javascript. We will discuss this in a later lecture.

React is most commonly used for web development. However, it can be used to create mobile applications, desktop applications and even terminal applications!

Getting Started

We will be creating a basic react application and attempting to understand it.

To do this, we'll use **create-react-app.** create-react-app is a tool developed by Facebook that allows to easily create scaffolds for React applications.

Creating a React App

Let's begin. Use create-react-app to create your first application! create-react-app is a program that is available to use through NPM. Open the terminal and enter the following:

```
npx create-react-app my-first-app
```

Once complete, a **my-first-app** folder will be present. Inside is your brand new React application.

my-first-app

Let's take a look at the most relevant files in our app.

The most important files are highlighted in yellow.

```
public/             // Static assets go here
   index.html       // Served when the page is loaded.

src/                // Your code goes here.
   index.js         // React mounts here.
   index.css        // Global styles go here.
   App.js           // An example component.
   App.css          // Example components CSS file.
   App.test.js      // Test file for example
                    component.
package.json        // Application manifest file.
yarn.lock           // Application manifest file.
README.MD           // Readme
```

Starting the Application

Let's start the application and take a look. Then we'll see how it works.

```
yarn install; yarn start;
// You only need to run yarn install once for now.
```

Once the application is started, you should see a link to the following address. Open it in your browser to see the application.

```
https://localhost:3000
```

App.js

The component file defines a function, App, which is a component function. The function returns a specification of the HTML structure of the component.

However, the output looks a little different to HTML. Why is that?

```
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;
```

JSX

JSX (Javascript XML) is an *extension* of Javascript. It allows you to treat HTML markup like you would any javascript object or variable.

Because we use JSX, we can think about our components like standard Javascript functions. This provides us with a lot of flexibility.

JSX also provides a lot of functions that HTML cannot provide. We can embed variables inside it, and call functions.

```
// You can store JSX within a variable.
const hello = <h1>Hello, JSX!</h1>;

// You can use them with standard javascript logic.
const even = isEven(num)
   ? <h1>Even</h1>
   : <h1>Odd</h1>;

// You can even embed other variables within your
JSX.
const odd = <h1>Even: {isOdd(num)}</h1>;
```

App.js

Let's take another look, given what we've learned.

```
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;
```

App.js

Let's take another look, given what we've learned.

```
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;
```

App.js

Let's take another look, given what we've learned.

```
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;
```

instead of class, we use className

A bracket like this allows us to structure our JSX in a neat way.

Edit the App Component and Reload

Now we know how the application works, try editing what the App component returns. Insert the following:

```
<h1>Hello, World!</h1>
```

Put it just below the <header> tag in App.js. You should notice the app will reload automatically. This is known as **hot module reloading** and allows you to see the changes you make to the app reflected on the page as soon as the file is saved.

Putting it All Together

Let's edit our App.js component to show the current date and time, instead.

```
import * as React from 'react';

function App() {
   const date = new Date(Date.now());
   return (
      <div>
         {date.toLocaleString()}
      </div>
   );
}

export default DateAndTime;
```

Good work!

Appendix: index.js

A standard javascript file. It handles binding your react app to the web page.

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

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

Appendix: index.js

A javascript file that handles binding your react app to the web page.

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

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

We can import our component from our App.js file.

Appendix: index.js

A javascript file that handles binding your react app to the web page.

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

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

The render function binds some JSX to an element that already exists on the page.

The first argument is our JSX, and the second element is the element on the page we attach the JSX to.

Appendix: index.html

This is where the element that we attached our JSX to lives.

```
<!DOCTYPE html>
<html lang="en">
<head>
   ...
</head>
<body>
   <div id="root"></div>
</body>
</html>
```