React Foundation

Module 1: Baby Steps

Azat Mardan @azat_co

Node University

Hello World

React.js CDN

KISS by linking to CDN:

```
<script src="https://unpkg.com/react@15/dist/react.js"></script>
<script src="https://unpkg.com/react-dom@15/dist/react-dom.js"></script>
```

HTML Structure

The `div` in the body of `index.html`:

```
  <body>
    <div id="content"></div>
    <script type="text/javascript">
    ... // React.js code
    </script>
  </body>
</html>
```

H1 Element

The following snippet creates the h1 React.js object with content 'Hello world!':

```
React.createElement('h1', null, 'Hello world!')
```

H1 Element

The following snippet creates the h1 React.js object with content 'Hello world!':

```
React.DOM.h1(null, 'Hello world!')
```

Rendering

Once the element is created we render it to the DOM element with ID `content` and renders it:

```
ReactDOM.render(
  React.createElement('h1', null, 'Hello world!'),
  document.getElementById('content')
)
```

Running the Page

Open `hello-world/index.html` and check if you see Hello world!

Inspecting HTML

If we inspect the HTML generated by React.js, it will have this attribute:

```
<h1 data-reactroot>Hello world!</h1>
```

HTML Arguments

Virtually all HTML arguments are supported so you can pass them like this:

```
React.createElement('div', {style: {color: red}}, 'Hello ', 'world!')
```

Note: You can add many parameters at the end to combine them.

`for` and `class` Attributes

If you need to use `for` or `class` attributes, their names are `htmlFor` and `className`. For example,

```
React.createElement('div', {className: 'hide', htmlFor: 'input'}, 'Hello world!')
```

Demo

Hello World without JSX

Nesting

```
React.createElement('div', {className: 'post-content'},
  React.createElement('h1', {className: 'post-heading'}, 'Universal Web'),
  React.createElement('p', {className: 'first-post-para'}, 'GitHub Material: ',
    React.createElement('a', {className: 'ext-linnk', href: 'https://github.com/azat-co/universal-web'}, 'github.com/azat-co/universal-web'),
    // ...
  )
)
```

Problem!

Too many `React.createElement()` statements

Solution

```
const cE = React.createElement
```

But there is an event better way!

Meet JSX! 

What is JSX

JSX is a JavaScript alias which looks like a combination of JavaScript and XML (HTML is a form of XML).

https://jsx.github.io/

Hello World in JSX

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('content')
)
```

How JSX Works

JSX is compiled into native/regular JavaScript which is run in the browsers.

Source-to-source compilation is called transpilation.

Why use JSX

JSX allows for easier and faster writing HTML views and elements along with JavaScript

Note: As you've seen from the previous Hello World! example, JSX is optional.

JSX is *the* recommended way of writing React.js apps, because it provides syntax for components, layouts and hierarchy.

Ways to use JSX

1. Pre-process with `babel-cli`: production recommended
2. Build with Gulp, Grunt, Webpack and Babel: production recommended
3. Run-time via `babel-core`: development or demo only

Let's use JSX, Webpack, Babel and npm to make it close to real web development flow

Real Dev Setup

- Babel
- Babel React presets
- Webpack
- npm scripts

Why use Webpack?

What Webpack will do for You

1. Bundle JS, CSS, etc.
2. Minification
3. Dependency management
4. React/JSX transpilation

What is and why Babel?

Babel

Babel is *the* JavaScript compiler.

» Write in ECMAScript 6+/ES2015+, run ES5-friendly code to support old browsers that don't have ES6.

» Transpile JSX into regular JavaScript

» Other plugins

http://babeljs.io/

Coding Time 

New Project with npm

```
$ mkdir react-project
$ cd react-project
$ npm init
```

Install the React Deps!

```
$ npm i react@15 react-dom@15 -D
```

Install Webpack Locally!

```
$ npm i webpack@1 -D
```

Install the Babel Deps!

```
$ npm i babel-core@6 babel-loader@6 babel-preset-react@6 -D
```

Dependencies in `package.json`

```
"devDependencies": {
  "babel-core": "6.13.2",
  "babel-loader": "6.2.4",
  "babel-preset-react": "6.5.0",
  "react": "15.2.1",
  "react-dom": "15.2.1",
  "webpack": "1.13.3"
}
```

`package.json` for Babel

```
"babel": {
  "presets": ["react"]
},
```

Scripts

```
"scripts": {
  "build": "./node_modules/.bin/webpack",
  "build-watch": "./node_modules/.bin/webpack -w"
},
```

Webpack Config

`webpack.config.js`

webpack.config.js:

```
module.exports = {
  entry: './jsx/app.jsx',
  output: {
    path: __dirname + '/js/',
    filename: 'bundle.js'
  },
```

webpack.config.js:

```
  module: {
    loaders: [
      {
        test: /\.jsx?$/,
        exclude: /(node_modules)/,
        loaders: ['babel']
      }
    ]
  }
}
```

Other Files

1. Create `index.html`
2. Create `jsx/app.jsx`

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/javascript" src="js/bundle.js">
    </script>
  </body>
</html>
```

Test Setup

1. `index.html` with `<script src="js/bundle.js"></script>`
2. script with `console.log('start')` in `jsx/app.jsx`
3. Run build with `npm run build` (not `npm build`)

`code/react-project`

Did it build?

Did you see the console log message?

Include Deps

They are not global anymore thanks to Webpack we can use `require` out-of-the-box (not the same as `import`) `app.jsx`:

```
const ReactDOM = require('react-dom')
const React = require('react')
```

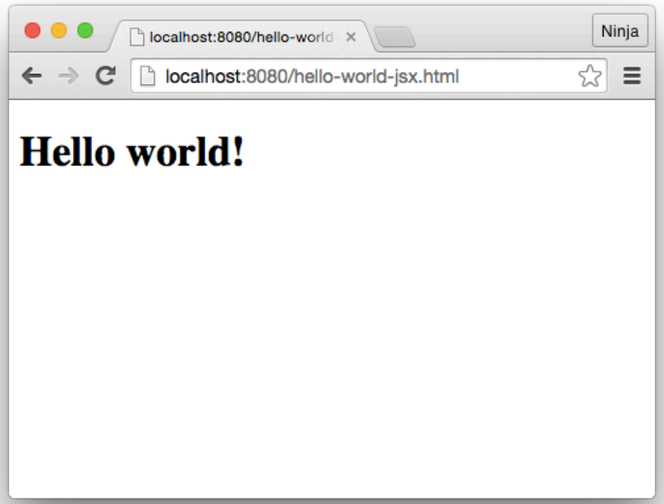
JSX Code

Change `React.createElement` to `<h1>...</h1>` in `app.jsx`:

```
ReactDOM.render(
  <h1>Hello world!</h1>,
  document.getElementById('content')
)
```

Running the Code

Open `code/hello-world-jsx/index.html` (preferably with `node-static` or `http-server`) and check if you see Hello world!



Note

```
npm i -g node-static
```

```
static
```

Open http://localhost:8080

Nested Elements

Nesting React.js components is easy.

Rendering Title and Text

This is how we can nest `<h1>` and `<p>` inside of `<div>`:

```
ReactDOM.render(
  <div>
    <h1>
      Core React.js
    </h1>
    <p>This text is very useful for learning React.js.</p>
  </div>,
  document.getElementById('content')
)
```

Single Top-Level Tag

Remember to always have only one element as the top level tag! For example, this is a **no go**:

```
ReactDOM.render(
  <h1>
    Core React.js
  </h1>
  <p>This text is very useful for learning React.js.</p>
  ,
  document.getElementById('content')
)
```

Order of the Code

Remember that the content element (`<div id="content"></div>`) must precede the React.js code (`<script ...`), for the `getElementById` method to locate the proper DOM element:

```
...
  <div id="content"></div>
  <script type="text/javascript">
    ReactDOM.render(
      // ...
      document.getElementById('content')
    )
  </script>
...
```