BOOK CHAPTERS (SAMPLE)

The following chapters are included in this sample.
For a full list of the chapter included in the book, refer to the end of this document.

1 *Introduction to React*

React is an open-source UI library developed at Facebook to facilitate the creation of interactive, stateful & reusable UI visual components for websites and applications. It is used at Facebook and Instagram in production, amongst other websites.

React gets its name because components have what's known as 'Reactive State'. This means that when your data in your component changes, your UI reacts and automatically updates to reflect changes.

1.1 What's So Special about React?

- **Speed**

  It's fast. Facebook claim that React can redraw a UI 60 times a second.

- **Reacting Components**

  Software applications use data, which changes frequently. When you build software that uses data, you often need to detect when the data changes and refresh the user interface when it does. React is different. When you write React code, you are writing Components. These Components *React* to data changes and redraw themselves, refreshing the user interface automatically.

- **Reusable Components**

  When you write React code, you are writing Components. They are designed to be Reusable. These Components can be used in one place in the Application, or in as many places as you wish. Facebook wrote React. Think about how many times Facebook reuses the 'like' Component.

- **Virtual DOM**

  Most JavaScript libraries directly update the DOM in the browser. React does not work that way. When React is running, it keeps a 'DOM in memory' that is updated by your components. When React change detection occurs it compares the 'DOM in memory' against the DOM in the browser and only updates the differences in the user interface on the browser. This avoids the performance issues of redrawing the whole user interface (DOM) again and again (rendering cycles), only redrawing component parts that have changed.

- **Server-side Rendering**

  One of its unique selling points is that not only does it perform on the client side, but it can also be rendered server side, and they can work together inter-operably.

- **JSX**

  ANGULAR, EMBER AND KNOCKOUT PUT "JS" IN YOUR HTML.
  REACT PUTS "HTML" IN YOUR JS.

  Most of the other JavaScript libraries (Angular, Ember, Knockout) allow you to embed JavaScript (or bindings) into HTML markup. React does things the other way round. You can embed HTML (or other markup for React Components) into your JavaScript.

  When you look at JavaScript that contains JSX, You will a lot of inline XML-like code, without quotes. You will also see that there are script blocks with different types (for example 'text/babel'). This is because the XML-like code (the JSX) is compiled into regular JavaScript before it is run.

  This initially sounds like a bad idea but think about this: the markup is compiled before runtime. Bad markup will be identified before the code is run.

  We will go into detail on JSX in Chapter 'Introduction to JSX'.

- ***XHP***

  XHP is a similar thing to JSX and it was developed before the React JavaScript library. It enables PHP developers to develop user interface components on the server-side using the same XML format.

- **React Native**

  You can use React to develop native iOS and Android apps using the React Native libraries supplied by Facebook.

- ***AJAX***

  Applications built with React are modular. React is just one of the modules. React is a view library and React has no networking/AJAX features. To perform networking/AJAX operations (such has getting data from the server), developers need to use an additional JavaScript module (such as JQuery) alongside their React code.

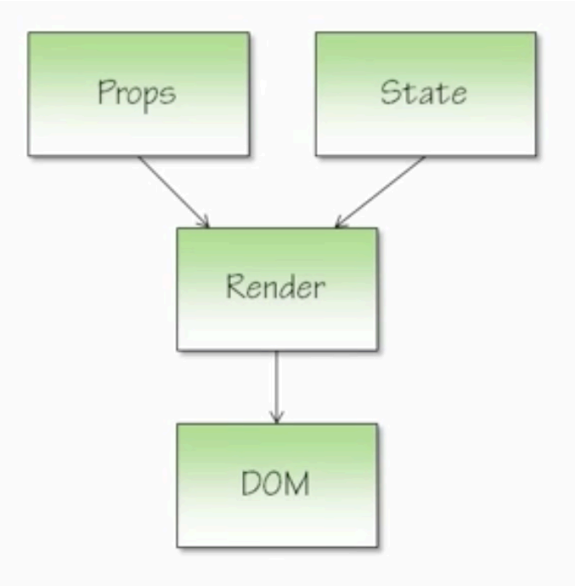
1.2 So What Does React Do?

React is a comparatively small JavaScript framework that allows the developer to write user interfaces that are composed of one or more Components. In these Components React takes care of the following for you:

1. Rendering the Model (see MVC) to the DOM.
2. Responding to Events.

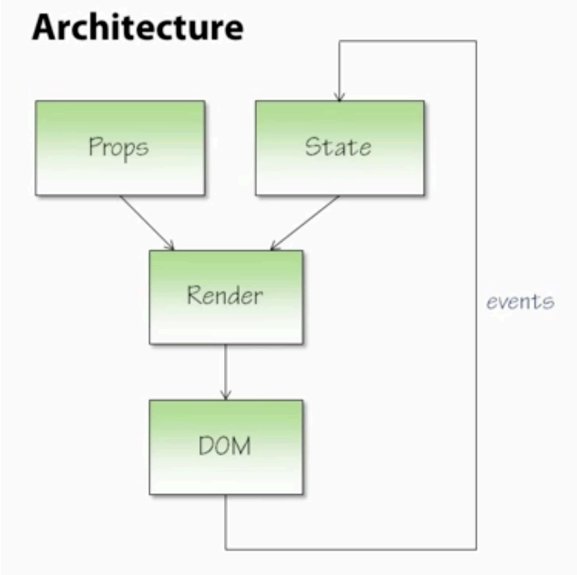
Rendering the Model to the DOM

Component inputs are props and state (the model).
Difference is state can change
Use state components as little as possible
Data flows downwards into render then dom
Way to change the dom is to change the model.



Responding to Events

The events in the DOM fire event handlers in the Components. The Components update the Model to change the DOM.



1.3 Other Libraries

- **JQuery**

  Many people use React and JQuery together. However, React will work fine without JQuery. JQuery developers will need to *think differently* when coding in React. In JQuery developers manipulate the DOM directly (imperatively) in order to change how the UI is rendered. React allows developers to have UI components. These UI components have states that are used to define how the UI is rendered. To change how the UI is rendered, the developer should change the 'state' of a component to tell it to change.

  Many developers use JQuery with React to perform asynchronous operations, as React does not include this capability.

- **Angular**

  Angular also handles DOM manipulation for the developer. It gives the developer to bind data in the model to the markup in the view (i.e. the DOM). When the user changes data in the model, this binding updates the view, similar to how React updates when a state (data) is changed. However Angular does not use a 'Virtual DOM' and it uses change detection algorithms (state tree change algorithms) to figure out what components have changed so it can update the DOM. It is arguably less efficient at updating the UI after a state change has taken place.

  Angular is a larger library and provides you everything you need to write a single-page web application. This library contains more services to the developer, for example routing, http communication with servers.

- **Ember**

  Ember is a larger library than React.

- **Backbone**

  Backbone is a larger library then React.

1.4 React ES5 & React ES6

- **Introduction**

  React has been around for a while and JavaScript ES6 was developed after React had already been released. From release 0.13.0, React started to support the development of React Components in ES6. Please see Appendix 'Versions of JavaScript' for more information about the versions of JavaScript available.

- **Examples**

  The syntax for using React with ES5 is quite different from using React with ES6. This book will attempt to cover both, while recognizing that ES6 (and later) is the future. Don't let the syntax differences put you off – you can see that React is doing similar things whether in ES5 or ES6.

- **Pre-ES6**

  In the context of this book, this means 'React with ES5', or 'any other version of JavaScript before ES6'.

  This book contains many simple 'Pre ES6' exercises which you can tryout using JSBin.com. JSBin.com allows us to quickly write code online without having to setup any kind of project environment. This is the simplest environment in which you can run React code. No compiles, build processes. Just type in the code and it run. As of the time of writing this book, JSBin.com only worked with the earlier React ES5 syntax.

- **Post-ES6**

  In the context of this book, this means 'React with ES6', or 'any other version of JavaScript after and including ES6'.

  This book includes a sample project written in ES6, from which I will provide code samples. Please refer to Chapter 'Introduction to the React Trails Project' Project for details on how to run React in this environment.

2 *Exercises with JSBin.com*

2.1 Introduction

JSBin is a website in which you can develop simple applications and try them out. It is similar to Plunker except that it works with a wider range of libraries. When you open JSBin.com, it shows you a series of vertical panels. Each panel lets you code an aspect of an html page (for example html, CSS, JavaScript). The html page (the product of the code in the other panels) will be executed and output in one of the panels to the right.

2.2 Pre-ES6

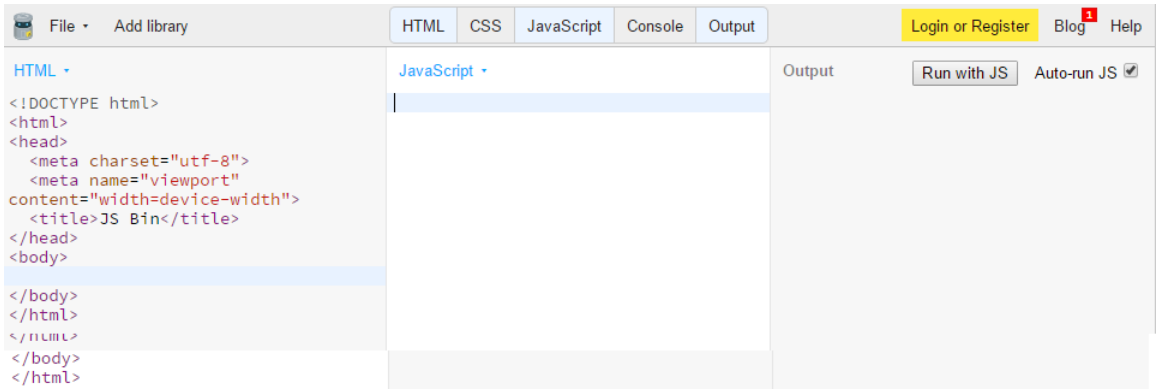
As of the time of writing this book, JSBin.com only worked with the earlier React ES5 syntax. This means that JSBin.com allows you to try out code with the earlier React ES5 syntax but not with the later React ES6 syntax.

2.3 Example - Introduction

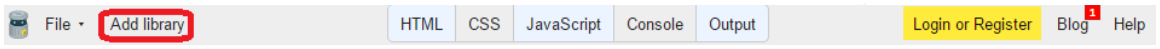
To learn how to get going in jsbin.com with React, we are going to create a tiny 'Hello World' application to display this message to the user.

2.4 Example - Instructions

1. Open your browser and navigate to the following web page: http://jsbin.com/



2. Select 'Add Library' and select 'React with Add-Ons + React DOM 15.1.0' in the list.



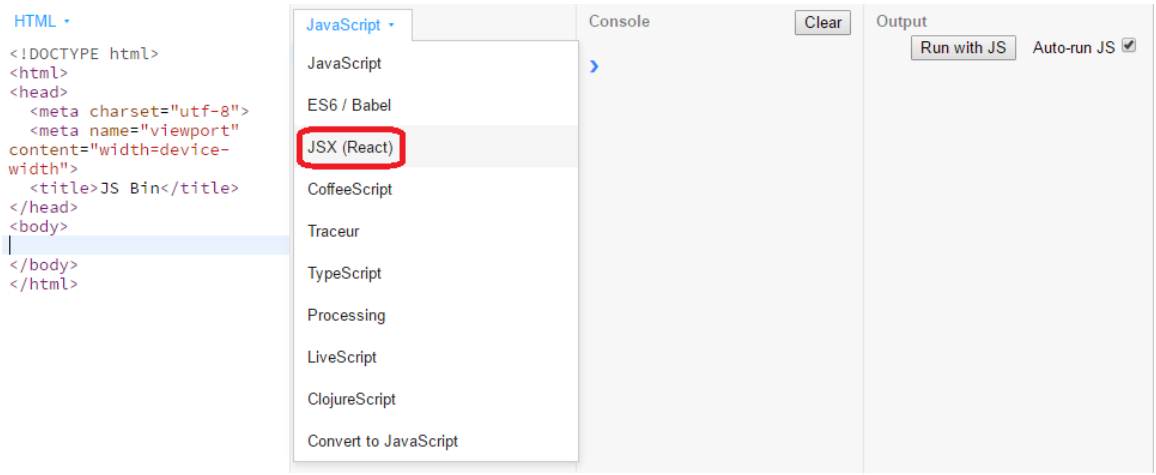
3. We are going to change the html so that it has a container for a 'hello world' simple react component. Edit your html (the panel on the left) and add the following element to the html:

```
<div id=”container”></div>
```

This results in the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <div id="container"></div>
</body>
</html>
```

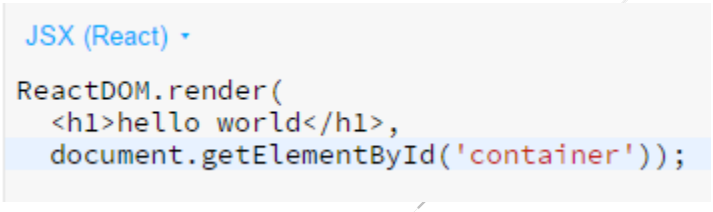
4. We are going to add a JSX react script to create the 'hello world' simple react component. First of all we need to select the script type in the middle panel. It defaulted to 'JavaScript' but we need to change it to 'JSX (React)'.



5. We now add the script code in the panel below where we selected 'JSX (React)':

```
ReactDOM.render(
  <h1>hello world</h1>,
  document.getElementById('container'));
```

The script panel should look like this:



6. That's it, your code should be running. The output panel on the right side shows the output from the running code on the other panels. It should now show the following:

2.5 Tip - Use Line Numbers

Introduction

It easy to make mistakes when writing code in an online editor such as jsbin.com. You will often see 'error at line x' messages when you run the code. This tells you to look at that line of code. However, jsbin.com does not show line numbers by default.

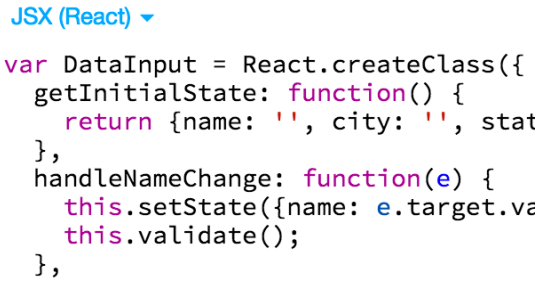
- **Jsbin.com Hidden Trick**

  If you double-click on the panel language selector, it shows the line numbers in your code.

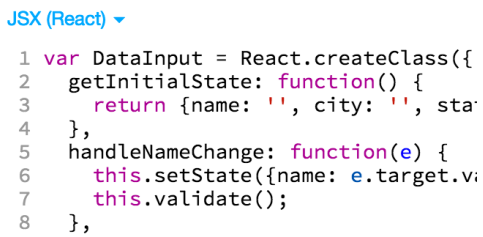
  Double-click this:

  and this:

  becomes this:

2.6 Tip - Use Syntax Highlighting By Copying and Pasting

- **Introduction**

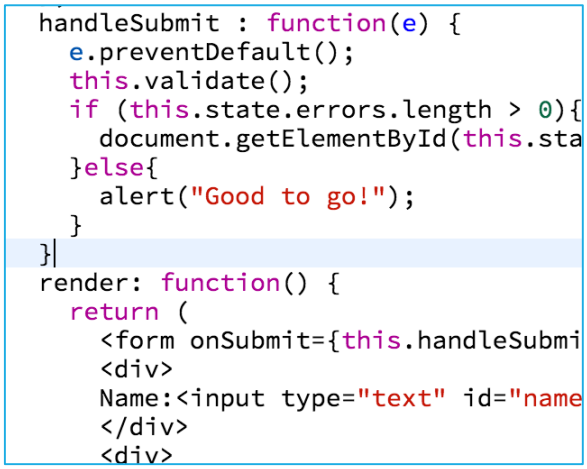
  Unfortunately, most of the online tools like jsbin.com don't have syntax highlighting.

  However there is a solution and I use this all the time. Install another editor on your computer, for example Microsoft Code. When you have 'issues' getting your code running in jsbin (or another online editor), copy and paste the code into your editor on your computer. Save the code you pasted as a '.js' file and then the editor should syntax highlight the file. That makes it much easier to find your problem. Quite often you stare at the screen for minutes then you paste it into Code and immediately see that you made a simple mistake like forgetting to add a quote or a comma.

Example

The code below is missing a comma. The picture on the left is how the code looks in jsbin.com. The picture on the right is how the code looks in Microsoft Code.

As you can see it is much easier to diagnose the issue in Microsoft Code, its highlighted in red!



3 Introduction to Components

3.1 Introduction

React Components are like UI building blocks. They are what you use to assemble a working React application and you need to know how they work.

3.2 Facts about Components

- They need to be named.
- They have only one root node.
- There are different types of Components and they can be created in different ways.
- They must be rendered into a target element in the DOM.
- They can contain other components (Composition).
- They can accept input data through (properties).
- They can store their own data (states) and they REACT when this data changes.
- They respond to events.

Sounds daunting doesn't it? Don't worry: all of the above points are covered in this book.

3.3 Creating React Components

React allows developers to create these components in different ways according to your development environment and state requirements. This is just an introductory chapter so we won't go into much detail on this.

Development Environment

As mentioned earlier, from release 0.13.0, React started to support the development of React Components in ES6.

- Older projects (and websites like jsbin.com) create Components using the pre-ES6 syntax.
- Many more 'modern' React projects create Components using the post-ES6 syntax.

Components with State

Don't worry if you don't know what 'State' means. It just means 'its own data'. If your React Component needs to contain state, then you should create it using 'React.createClass' or by extending React.Component.

- ***Create using React.createClass (Pre-ES6)***

  This is how we usually create the Components in JSBin.com. Components created in this manner are full React Components that can store state. The argument to this function is an object. At a minimum, this object should contain a render function, which returns a single React Component.

- ***Create by Extending React.Compoment (Post-ES6)***

  This is how we usually create Components in a modern React project. ES6 allows JavaScript developers to create classes. You should define your React Component as a class that extends the class React.Component. Components created in this manner are full React Components that can store state. At a minimum, this object should contain a render function, which returns a single React Component.

  If you don't need your React Component to contain state data then you should create it as a Stateless Function. This works in all versions of JavaScript, although you can use a different syntax in ES6 onwards if you want.

Components without State

You write your React Components as a function. No class, nothing. This is a simple way to create React Components that don't hold state. You can still pass in data into these components using Properties but you cannot change this data. This is a great way to create simple user interface components that don't have complex behavior.

4 Introduction to JSX

4.1 Introduction

ANGULAR, EMBER AND KNOCKOUT PUT "JS" IN YOUR HTML. REACT PUTS "HTML" IN YOUR JS.

As mentioned earlier, React is very different from Angular and Ember because it works the other way round. Normally you embed the JavaScript into the HTML, that is the way most JavaScript libraries work. React is different though and JSX is one of the differences, enabling you to put HTML into your JavaScript. Also, this JSX code (that represents HTML in amongst other things) is validated and compiled.

JSX can be used to write HTML in your JavaScript. However it also lets you add Tags that represent React Components as well! So with JSX you are not just limited to just HTML.

4.2 Compilation

JSX is an XML-like syntax that you can add inline into your JavaScript code. Later on the JSX is compiled into regular JavaScript that invokes React code before it is executed on the browser.

However not everyone loves the JSX syntax and the idea of writing the XML-like syntax inline in script. So some people skip steps 1 and 2 above and just write regular JavaScript (3).

Each element in the XML is compiled into a JavaScript function call to React code create the element. Elements are data object that represent markup, usually HTML. Element attributes are converted into arguments in the function calls. Nested elements (elements within elements) become additional arguments in the function calls.

- **Example**

  This JSX code:

```
var profile = <div>
<img src="avatar.png" className="profile" />
<h3>{[user.firstName, user.lastName].join(' ')}</h3>
```

```
</div>;
```

is compiled into:

```
var profile = React.createElement("div", null,
    React.createElement("img", { src: "avatar.png", className: "profile" }),
    React.createElement("h3", null, [user.firstName, user.lastName].join(" "))
);
```

Example - Notes

1. Notice how the 'div' element is compiled into a call to 'React.createElement'.
2. Notice how the nested 'img' and 'h3' elements are compiled into calls to 'React.createElement' within the original call to 'React.createElement' above.
3. Notice how the 'img' element 'src' and 'className' attributes are converted into arguments in the call to 'React.createElement'.

JSX and HTML Markup

JSX is very useful for writing html markup inline in your JavaScript code. For example, the code below uses JSX and JavaScript to render an html H1 element.

```
var headerElement = <h1>Hello There</h1>;
ReactDOM.render(headerElement, document.getElementById('container'));
```

- ***Compiled JSX for HTML***

  When you compile the JSX code for HTML, it creates a call to a React function in the form of React.DOM.[tag]. This is the React function that represents the HTML element.

  For example, the JSX for ' ' would compile into the JavaScript 'React.DOM.div(null)'.

  Remember that the React class that represents the HTML element has a name that begins with a lower-case letter. Therefore, you should use lower-case HTML tag names in your JSX, e.g. .

- ***NOTE: To render html markup, use lower-case tag names in JSX, e.g. h1.***

JSX and React Components

JSX can be used to describe React Components in an XML-like syntax. Later on this XML syntax is converted into JavaScript code. For example the code below uses JSX and JavaScript to render a React Component.

```
var Hello = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
```

```
ReactDOM.render(
  <Hello name="World" />,
  document.getElementById('container')
);
```

- ***Compiled JSX for React Components***

  When you compile the JSX code for React Component, it creates a call to a function of exactly the same name.

  For example, the JSX for '<Hello/>' would compile into the JavaScript 'Hello(null)'. This refers to the 'Hello' React Component, which must be available to be used.

  Remember that your React component class names begin with an upper-case letter. Therefore, you should use upper-case tag names in your JSX to refer to React Components, e.g. <Hello>.

Compiling JSX Code

Your browser cannot is good at executing regular JavaScript code but it is not equipped at running JSX code. Your JSX code must be compiled into regular JavaScript code before it is executed.

If you want to look at JSX compilation in more detail, please take a look at the following page: https://babeljs.io/repl/

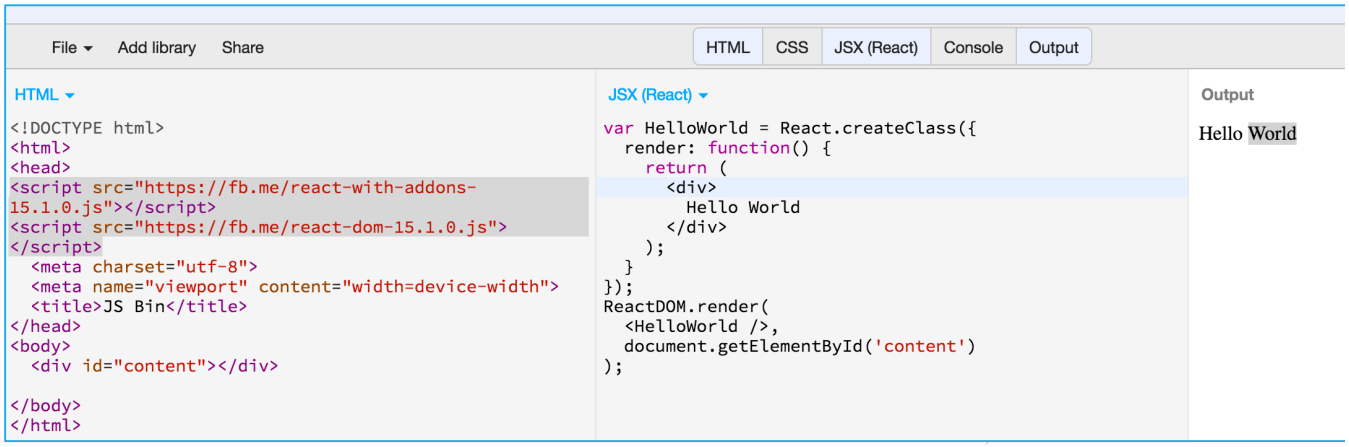
Compiling JSX Code In Your Browser

We have been using jsbin.com to try out our code. This website compiles your JSX code in your browser 'just-in-time' before it runs. This is great for prototyping but not so great for production websites for the following reasons:

- You need to compile the JSX code into regular JavaScript every time the page loads. This makes the page slower. It is much more efficient to do the compilation in the build process and deploy the regular JavaScript.
- The JavaScript code you deploy cannot be minified, linted, debugged or formatted.

JSX Code in JSBin

When you use JSBin and you select 'JSX(React)' in the script dropdown, the website automatically compiles the JSX code therein to regular JavaScript and compiles it in the executing page 'just-in-time'. So you don't really need to worry about anything other than selecting the library 'React with Add-Ons + React DOM 15.1.0' and selecting 'JSX(React)' in the script dropdown. Nice!



- **JSX Code in a Standalone Page – Example**

  You can use the babel library to your page to compile your JSX code on your webpage 'just-in-time'. You need to ensure you load the correct libraries (script tags with 'src') and ensure that the script tag has the correct type 'text/babel'.

  The code below is a standalone html page with react code that you should be able to run on any web server. It compiles the JSX code contained within the last script tag (in bold below) and displays the React Component.

```
<html>
<head>
    <script src="https://fb.me/react-with-addons-15.1.0.js"></script>
    <script src="https://fb.me/react-dom-15.1.0.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
    <script type="text/babel">
    var HelloWorld = React.createClass({
        render: function() {
            return (
            <div>
                Hello World
            </div>
            );
        }
    });
    ReactDOM.render(
    <HelloWorld />,
    document.getElementById('container')
    );
    </script>
</head>
<body>
    <div id="container"></div>
</body>
</html>
```

- ***Example – Notes***

  1. The first two scripts that begin with 'fb' are React Runtime files. They are required for React to work in this page.

2. The third script is the babel browser script. This babel browser script can transform ES6 and JSX code into regular ES5 code.
3. The fourth script tag contains JSX code. Notice how it has the 'type' attribute set to 'text/babel'. This ensures that the babel browser script (loaded above) compiles the JSX code within the script block into regular JavaScript. After this script has been compiled by the babel browser script, it creates a React Component and then mounts it to the div 'content'.

- **Compiling JSX Code in Your Build**

  When you write your React code in a project and you have a build process, this build process will convert the JSX code into regular JavaScript. This avoids having your browser perform JSX compilation and this is more suitable for production websites. We will cover this in Chapters 'Babel' and 'Browserify and Babelify'.

4.3 JSX Attribute Expressions

When you write your JSX code, you can use expressions to provide values or perform calculations. These expressions are delimited by curly braces and they must contain valid JavaScript expressions. When the JSX compiler runs, it converts these expressions into arguments to the 'React' calls produced by the compiler.

- **Example - String Literal Expression**

```
<Hello now={"2016-08-01"} />
```

- **Example - JavaScript Expression**

```
<Hello now={new Date().toString()} />
```

```
<div>
{1==2?<Component1/>:<Component2/>}
</div>
```

4.4 Child Component Elements

A JSX Component may have child Components.

```
<Hello>
<Component1/>
<Component2/>
</Hello>
```

A component can access its children with this.props.children. More on that later!

4.5 JavaScript Expressions – Whitespace

Whitespace between {} expressions is not allowed. This is because JSX is converted into JavaScript, including the JavaScript Expressions.

- **The following outputs "MarkClow":**

```
{"Mark"} {"Clow"}
```

- **You can amend the expression to include spaces in several different ways.**

  If you examine the example code below you will see more than one to get around this.

- **Example Code**

MarkClow - no space
Mark Clow - space
Mark Clow - space

```
ReactDOM.render(
  <div>
  {"Mark"}{"Clow"} - no space
  <br/>
  {"Mark Clow"} - space
  <br/>
  {"Mark"} {" "} {"Clow"} - space
  </div>,
  document.getElementById('conttainer')
);
```

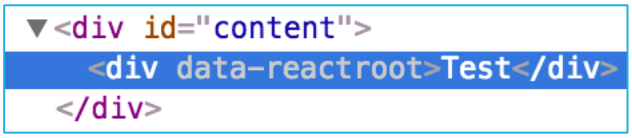
4.6 HTML Attributes

You can write JSX to output HTML. This HTML can (obviously) have attributes. JSX will only output valid HTML attributes, not just anything. However you can use custom attributes if you prefix them with 'data-'.

- **Example – Invalid Custom HTML Attribute**

```
ReactDOM.render(
  <div address="Atlanta">
  Test
  </div>,
  document.getElementById('container')
);
```

results in the following html:



- **Example – Valid Custom HTML Attribute.**

```
ReactDOM.render(
  <div data-address="Atlanta">
  Test
  </div>,
  document.getElementById('container')
);
```

results in the following html:

4.7 HTML Attributes and JavaScript Reserved Words

Attributes cannot use JavaScript reserved words, obviously because JSX generates JavaScript. This causes problems because some JavaScript words overlap with HTML attributes:

- for
- class

The answer to do this is to use a slightly different attribute name instead.

- for – use 'htmlFor' instead
- class – use 'className' instead

- **Example – JavaScript 'Reserved Word Attribute**

The code below does not generate html with the required 'class' attribute.

```
ReactDOM.render(
  <div class="title2">
  Test
  </div>,
  document.getElementById('container')
);
```

It results in the following html:

```
▼<div id="content">
    <div data-reactroot>Test</div>
  </div>
```

- **Example – JavaScript Reserved Word Attribute Name Corrected**

  The code below correctly generates html with the required 'class' attribute.

```
ReactDOM.render(
  <div className="title2">
  Test
  </div>,
  document.getElementById('container')
);
```

  It results in the following html:

```
▼<div id="content">
    <div data-reactroot class=
    "title2">Test</div>
  </div>
```

4.8 HTML Style Attribute

The style HTML attribute works very differently in React.

- **Style Attribute Requires an Object**

  You can add use the 'style' attribute name but it expects to be assigned a JavaScript object within '{' and '}' brackets.

  - ***Double Brackets***

    Note that if you want to assign an inline object in the HTML Style Attribute then you should use double brackets. One set of brackets to indicate a JavaScript expression. One set of brackets to create a JavaScript object.

```
style={{fontWeight:"bold",border:"1px solid #ff0000"}}
```

  - ***Single Brackets***

    You can use single brackets if you use a JavaScript object variable.

```
var style={fontWeight:"bold",border:"1px solid #ff0000"};

…
style={style}
```

- **CSS Style Names**

  Notice that the CSS style names (for example 'font-weight' below) need to be converted to CamelCase for them to work.

- **Example – Html Style Attribute Used Incorrectly**

```
ReactDOM.render(
  <div style="font-weight:bold;border:1px solid #ff0000">
  Test
  </div>,
  document.getElementById('container')
);
```

This results in the following JavaScript error:

```
VM360 bimadozoqi.js:7 Uncaught Invariant Violation: The `style` prop expects a mapping
from style properties to values, not a string. For example, style={{marginRight: spacing
+ 'em'}} when using JSX.
```

- **Example – Html Style Attribute Used Correctly**

```
ReactDOM.render(
  <div style={{fontWeight:"bold",border:"1px solid #ff0000"}}>
  Test
  </div>,
  document.getElementById('container')
);
```

or

```
var style={fontWeight:"bold",border:"1px solid #ff0000"};
ReactDOM.render(
  <div style={style}>
  Test
  </div>,
  document.getElementById('container')
);
```

These result in the following output:

and HTML:

```
▼<div id="content">
    <div data-reactroot style="font-weight: bold; border: 1px solid
    rgb(255, 0, 0);">Test</div>
  </div>
```

4.9 Escaping and Unescaping Content

Introduction

JSX escapes all generated content by default. This prevents cross-site scripting attacks. For more information on cross-site scripting attacks please see the Appendix 'Cross-Site Scripting Attacks'.

Exceptions

Sometimes you need to include unescaped content in your React Components. For example if you need to generate some dynamic html and you cannot find any way around it. For this purpose, React provides an attribute 'dangerouslySetInnerHtml' to enable developers to generate unescaped content.

Example (Pre-ES6)

- ***Introduction***

Let's build a component that contains three elements:

1. Unescaped script block.
2. Unescaped html.
3. Escaped html.

- ***Steps***

1. Create a Skeleton React Application in JsBin.
2. Add the following to the html, inside the body:

```
<div id="container"></div>
```

3. Add the following code to the JSX(React) code:

```
var VeryNaughtyComponent = React.createClass({
  render: function() {
    var naughty = '<script>alert("123")</script>';
    return <div dangerouslySetInnerHTML={{__html: naughty}} />
  }
});

var NaughtyComponent = React.createClass({
  render: function() {
```

```
    var naughty = '<strong>Naughty</strong>';
    return <div dangerouslySetInnerHTML={{__html: naughty}} />
  }
});

var NiceComponent = React.createClass({
  render: function() {
    var nice = '<strong>Nice</strong>';
    return <div>{nice}</div>
  }
});

ReactDOM.render(
  <div>
  <VeryNaughtyComponent></VeryNaughtyComponent>
  <NaughtyComponent></NaughtyComponent>
  <NiceComponent></NiceComponent>
  </div>,
  document.getElementById('container')
);
```

- ***Generated HTML***

- ***Notes***

1. The unescaped '<script>alert("123")</script>' is output unchanged and has purple html highlighting. However, JSBin.com does not execute it for security reasons.
2. The unescaped 'Nice' is output unchanged and has purple html highlighting.
3. The escaped 'Nice' is output escaped. Note that the '' tag is shown in black text to indicate that it is escaped.

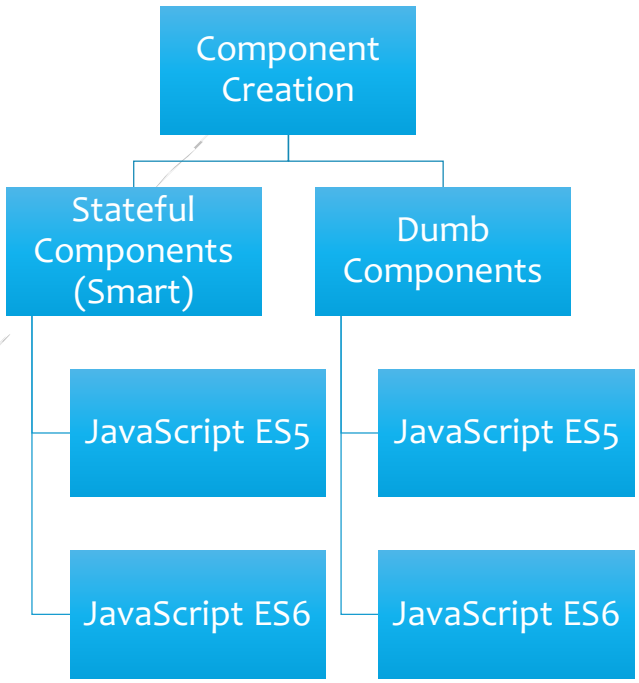
5 *Component Creation*

5.1 Naming

Before you create your React Component, you need to decide on a name for it. As a convention all React Component names are Camel Case and the first letter should be Upper case and all the html element tags should start with lower case character.

5.2 Factors that Affect Component Creation

Not all React Components are the same. Sometimes you will be working with React development in an older project, which uses pre-ES6 JavaScript. Sometimes you will be working with the newer technology: a later version of React with post-ES6 JavaScript. Sometimes you need to store state data in your components, sometimes you don't need to store state in your Components. This is covered in detail later on in Chapter 'Component Design – Thinking in React'. React allows developers to create these components in different ways according to your development environment and requirements.



5.3 Stateful Components

- **Introduction**

Stateful Components tend to be 'smarter'. They are smart enough to store information, telling them what to display and how to display it. When things happen they can also modify

this information, causing their display to change. Stateful Components need Classes, because they store modifiable data and code.

Using React.createClass (Pre-ES6)

This is how we usually create the Components in JSBin.com or in older projects. Components created in this manner are full React Components that can store state. The argument to this function is an object. At a minimum, this object should contain a render function, which returns a single React Component.

Example (Pre-ES6)

- ***Introduction***

  Let's pass car make and model information into a Component and then display it.

We will have custom attributes for car make and car model:

```
<Car make="Citroen" model="Dianne" />
```

The code inside the Car Component will access the values of these custom attributes (ie Properties) using the following variables:

1. this.props.make
2. this.props.model

- ***Steps***

1. Create a Skeleton React Application in JsBin.
2. Add the following to the html, inside the body:

```
<div id="container"></div>
```

3. Add the following code to the JSX(React) code:

```
var Car = React.createClass({
  render: function() {
    return <div>Car: {this.props.make} {this.props.model}</div>;
  }
});
ReactDOM.render(
  <Car make="Citroen" model="Dianne"/>,
  document.getElementById('container')
```

```
);
```

Extending React.Component (Post-ES6)

- ***Introduction***

  This is how we usually create Components in a modern project. ES6 allows JavaScript developers to create classes. You should define your React Component as a class that extends the class React.Component. Components created in this manner are full React Components that can store state. At a minimum, this object should contain a render function, which returns a single React Component.

```
class ExampleComponent extends React.Component {
  render() {
    return <div>Hello, world.</div>;
  }
}
```

- ***Example (Post-ES6)***

  The code below comes from the example project. It is introduced in Chapter 'Introduction to the React Trails Project'.

```
import React from 'react';
import { List, ListItem, ListItemContent, ListItemAction, Icon } from 'react-mdl';
import { Link } from 'react-router';

class SearchResults extends React.Component {

  render() {
    if (this.props.searchResults.length == 0) {
      return null;
    }
    const searchResultList = this.props.searchResults.map(function (value, index) {
      let linkTo = `/details/${value.lat}/${value.lon}`;
      return <ListItem key={value.unique_id}>
        <ListItemContent avatar="person">{value.name} {value.state}
{value.country}</ListItemContent>
        <ListItemAction>
          <Link to={linkTo}>Details</Link>
        </ListItemAction>
      </ListItem>;
    });
    return (
      <div>
        <h2>Results</h2>
        <List>
          {searchResultList}
        </List>
      </div>);
  }
}

export default SearchResults;
```

5.4 Stateless Components

- ### Introduction

  Stateless Components are dumb and are supplied with the data they need to display. They display things and they can let other components know when things happen. Stateful Components can be implemented as Functions as they are simple.

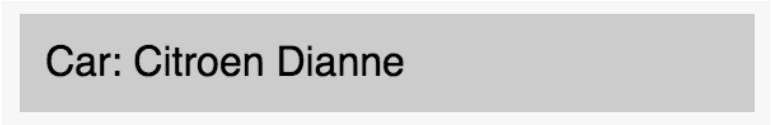
  They are a great way to create simple user interface components that don't have state or complex behavior. React Components created in this manner perform very well because they are simple and require much less overhead than Stateful Components.

  Note that Stateless Functions do not have Lifecycle Functions or References, which we will cover in later chapters.

- ### Example (Pre-ES6)

  - ***Introduction***

    Let's pass car make and model information into a Component and then display it.

    We will have custom attributes for car make and car model:

    ```
    <Car make="Citroen" model="Dianne" />
    ```

    The code inside the Car function will access the values of these custom attributes (ie Properties) using the following variables:

    1. this.props.make
    2. this.props.model

  - ***Steps***

    1. Create a Skeleton React Application in JsBin.
    2. Add the following to the html, inside the body:

    ```
    <div id="container"></div>
    ```

    3. Add the following code to the JSX(React) code:

    ```
    function Car(props) {
      var style={backgroundColor:'#cccccc', padding: '10px'};
      return (
    ```

```
    <div style={style}>
      Car: {props.make} {props.model}
    </div>
  )
}
ReactDOM.render(
  <Car make="Citroen" model="Dianne"/>,
  document.getElementById('container')
);
```

- **Example (Post-ES6)**

  The code below comes from the example project. It is introduced in Chapter 'Introduction to the React Trails Project'.

```
import React from 'react';

const Welcome = () => {
  return (
    <div className="centered">
      <h2>Welcome</h2>
      <p>Welcome to the 'Trail' project.</p>
      <p>This is a small React example project built from React Starterify.</p>
      <p>It enables us to query the <a
href="https://market.mashape.com/trailapi/trailapi">trail api</a>.</p>
      <p>
        This api gives access to information and photos for tens of thousands of outdoor
recreation
        locations including hiking and mountain biking trails, campgrounds, ski resorts,
ATV trails, and more.
      </p>
    </div>
  );
};

export default Welcome;
```

BOOK CHAPTERS (NON-SAMPLE)