ReactJS Basics

Overview

- JSX
- Styling
- Components
- Props

ReactJS, JSX and DOM

JSX

- JSX stands for JavaScript XML
- JSX is an extension to JavaScript syntax that allows you to write HTML within JavaScript.

Simple JSX

```
const elem = <h1> Hello! This is  JSX </h1>
```

In ReactJs JSX is used to create *“React”* elements which are later rendered to the ReactDOM.

From the example above, *elem* holds a React element.

JSX Rules

1. Use HTML tags.
2. Only one parent tag must be used in a JSX element.
3. Multiple tags can be nested within a parent tag.

Other JSX forms - multiple line JSX

```
const multiLineElement = (
        <div>
                <h1>Welcome to the world of JSX</h1>
                <p>
                        We can write multiple line JSX by surrounding the JSX
                expression in a parenthesis ()
                </p>
        </div>
)
```

Other JSX forms - JS expressions with HTML

```
const  name = “Codetrain”;

const element = <h1> welcome to {name} </h1>
```

JS expressions can be embedded into JSX by surrounding them with curly braces {}.

ReactDOM-rendering elements

ReactJS uses a virtual DOM to render React-JSX- elements.

ReactJS comes with a simple method on the ReactDOM object that takes two parameters- the React-JSX-element to render and the target element to render to.

Lets understand how the render process occurs.

ReactDom-Rendering elements

Supposing there is a in your HTML file as below:

```
<div id=”root”></div>
```

This html element is usually called the “root” DOM node.
Thus because everything inside it will be managed by ReactDOM.

NB: Typically there should be at least one root DOM node in your application.

ReactDOM- Rendering elements

We can render to the root DOM node;

```
const element = <h1>This is our first ReactJSX element</h1>;

ReactDOM.render(
                        element ,
                         document.getElementById('root')
                        );
```

Adding Attributes to JSX

You may add attributes to JSX just like adding attributes like id, classes , src to HTML tags.

Remember JSX is JavaScript and HTML after all.

There are only but a few tricks and tips.

Adding Attributes to JSX

You can use string literal as attribute values:

```
const element = <div id=”banner”>This is the banner</div>;
```

Multi-worded attributes should use camelCase:

```
const element = <div className=”banner”>This is the banner</div>;
```

Adding Attributes to JSX

You can specify float, and number literals for attributes by using the curly braces:

```
const element = <div tabIndex={1}>Tab 1</div>;
```

Adding Attributes to JSX

You can use curly braces to embed JavaScript expressions into attributes

```
const url =”https://codetraingh.com”;

const element = <a href={url} >codetrain</a>;
```

Styling in JSX

React elements can be styled by passing a style object to the style attribute of a react elements.

HOW?

Pretty simple.

React elements can take a style attribute just like HTML Tags.

There are just implemented differently. Instead of using a String literal, you use a JS Object.

Styling - HTML Approach

In a HTML, the style attributes uses a string literal

```
<h1 style=”color: red; background-color: black; font-size: 700”>
        This is a title
</h1>
```

Styling- JSX Approach

In JSX styling rules are

1. Converted to JS Object of property-value pairs
2. Snake property names are converted to camelCase property names

```
const element = (
        <h1 style={ {color:”red”,backgroundColor:”black”,fontSize:700} }>
                This is a title
        </h1>
);
```

ReactJS Components

What is a component?

A component is an atomic part of the application that is self-contained and optionally references other components to compose its output.

Components let you split the UI into independent, reusable pieces.

ReactJS Components

ReactJS has two types of components :

- Function based components
- ES6 class based components

Functional components

Functional component is a pure function that accepts a single props object as argument and returns a React element

NB: we will learn more about props later but for now props are JS Objects that are passed to components.

Function Component - Example

```
const HeaderComponent = props => {

    return <h1>This is the page header</h1>

}
```

NB: ES6 arrow function syntax
Typically components names should be capitalized

Class Component

Class components uses ES6 class and extends React.Component Object

```
Class HeaderComponent extends React.Component {
    render ( ) {
        return <h1>This is the page header </h1>
    }
}
```

Rendering Components

We can equally render component just like JSX

```
const HeaderComponent = props => <h1>This the page header</h1>;

const element = <div><HeaderComponent /> </div>

ReactDOM.render(element, document.getElementById('root'));
```

Rendering Components

NB: React treats components starting with lowercase letters as DOM tags. Components must be capitalized to be in scope

Composing components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail.

Component nesting enables flexible UI development.

Props - Properties

What are props?

Props mean properties. There are two types of props that can be passed to a component.

- ReactJS props - these are props that ReactJS passes to the component

- Custom props - these are props that we can passed to component as though they were attributes

Passing Props to component

Props-properties- can be passed to components just like JSX attributes (style, id, className, etc).

```
const app = <Welcome name=”Codetrain” />
```

In the code snippet above we passed the prop *“name”* to Welcome component.

NB:Lets just assume the component is already created

Accessing Props in component

Once props have been passed to a component the can be accessed within the component on the props object.

```
const Welcome = props => <h1> Welcome to {props.name} </h1>
```

NB: props passed to a component is *READ ONLY*. THEY SHOULD NOT BE MUTATED.

Now lets create some components

Exercise 1: Instructions

1. Open app.js file In the previous app you created. You will notice that by default a functional components has be created for you and exported.

2. Replace the App functional component with an ES6 arrow functional Component with the same name (App)

3. In your new component return an h1 JSX with the text “Codetrain is awesome, react works!!!!” and run your app.

4. Once you complete step 3 convert the component to a class component

Exercise 2: Instructions

1. Create a new react project

2. In app.js, create a class component beneath the App functional component with the name WelcomeComponent.

3. In render() of WelcomeComponent return an h1 JSX with the text “Codetrain is awesome, react works!!!!”

4. Now, In App(), return <WelcomeComponent /> and run your app

5. Once step 4 works, replace Codetrain with a prop (name) and pass the props value to the WelcomeComponent in App()