What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any `createElement()` and/or `appendChild()` methods.

JSX converts HTML tags into react elements.

You are not required to use JSX, but JSX makes it easier to write React applications.

Expressions in JSX

With JSX you can write expressions inside curly braces `{ }`.

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

One Top Level Element

The HTML code must be wrapped in *ONE* top level element.

So if you like to write *two* paragraphs, you must put them inside a parent element, like a `div` element.

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Attribute class = className

The `class` attribute is a much used attribute in HTML, but since JSX is rendered as

JavaScript, and the `class` keyword is a reserved word in JavaScript, you are not allowed to

use it in JSX.

Conditions - if statements

React supports `if` statements, but not *inside* JSX.

To be able to use conditional statements in JSX, you should put the `if` statements outside of the JSX, or you could use a ternary expression instead:

Option 1:

Write `if` statements outside of the JSX code:

Option 2:

Use ternary expressions instead:

React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8. There is an optional section on Class components for your reference.

Create Your First Component

When creating a React component, the component's name *MUST* start with an upper case letter.

Class Component

A class component must include the `extends React.Component` statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a `render()` method, this method returns HTML.

Function Component

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

Rendering a Component

Now your React application has a component called Car, which returns an `<h2>` element.

To use this component in your application, use similar syntax as normal HTML: `<Car />`

Props

Components can be passed as `props`, which stands for properties.

Props are like function arguments, and you send them into the component as attributes.

You will learn more about `props` in the next chapter.

Components in Files

React is all about re-using code, and it is recommended to split your components into separate files.

React Class Components

Before React 16.8, Class components were the only way to track state and lifecycle on a React component. Function components were considered "state-less".

With the addition of Hooks, Function components are now almost equivalent to Class components. The differences are so minor that you will probably never need to use a Class component in React.

Even though Function components are preferred, there are no current plans on removing Class components from React.

This section will give you an overview of how to use Class components in React.

React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render() function.

Components come in two types, Class components and Function components, in this chapter you will learn about Class components.

Create a Class Component

When creating a React component, the component's name must start with an upper case letter.

The component has to include the `extends React.Component` statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a `render()` method, this method returns HTML.

Component Constructor

If there is a `constructor()` function in your component, this function will be called when the component gets initiated.

The constructor function is where you initiate the component's properties.

In React, component properties should be kept in an object called `state`.

You will learn more about `state` later in this tutorial.

The constructor function is also where you honor the inheritance of the parent component by including the `super()` statement, which executes the parent component's constructor function, and your component has access to all the functions of the parent component (`React.Component`).

Props

Another way of handling component properties is by using `props`.

Props are like function arguments, and you send them into the component as attributes.

You will learn more about `props` in the next chapter.

Components in Files

React is all about re-using code, and it can be smart to insert some of your components in separate files.

To do that, create a new file with a `.js` file extension and put the code inside it:

Note that the file must start by importing React (as before), and it has to end with the statement `export default Car;`.

React Class Component State

React Class components have a built-in `state` object.

You might have noticed that we used `state` earlier in the component constructor section.

The `state` object is where you store property values that belongs to the component.

When the `state` object changes, the component re-renders

Lifecycle of Components

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: Mounting, Updating, and Unmounting.

Mounting

Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. `constructor()`
2. `getDerivedStateFromProps()`
3. `render()`
4. `componentDidMount()`

The `render()` method is required and will always be called, the others are optional and will be called if you define them.

---

constructor

The `constructor()` method is called before anything else, when the component is initiated, and it is the natural place to set up the initial `state` and other initial values.

The `constructor()` method is called with the `props`, as arguments, and you should always start by calling the `super(props)` before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (`React.Component`).