REACT JS

# Create React App

The create-react-app is an officially supported way to create React applications.

npm install -g create-react-app

Run this command to create a React application named myfirstreact:

npx create-react-app myfirstreact

# what is React?

React is a JavaScript library created by Facebook.

React is a tool for building UI components.

React creates a VIRTUAL DOM in memory.

Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

React only changes what needs to be changed!

# React Directly in HTML

The quickest way start learning React is to write React directly in your HTML files.

Start by including three scripts, the first two let us write React code in our JavaScripts, and the third, Babel, allows us to write JSX syntax and ES6 in older browsers.

<script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>

<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

# React ES6

ES6 stands for ECMAScript 6.

ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

ES6 introduced classes.

A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor() method.

The constructor function is called automatically when the object is initialized.

## Class Inheritance

To create a class inheritance, use the extends keyword.

A class created with a class inheritance inherits all the methods from another class:

## Arrow Functions

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

hello = function() {

return "Hello World!";

}

hello = () => {

return "Hello World!";

}

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

hello = () => "Hello World!";

## Variables

Before ES6 there were only one way of defining your variables: with the var keyword. If you did not define them, they would be assigned to the global object. Unless you were in strict mode, then you would get an error if your variables were undefined.

Now, with ES6, there are three ways of defining your variables: var, let, and const.

### Var

If you use var outside of a function, it belongs to the global scope.

If you use var inside of a function, it belongs to that function.

If you use var inside of a block, i.e. a for loop, the variable is still available outside of that block.

var has a function scope, not a block scope.

### Let

let is the block scoped version of var, and is limited to the block (or expression) where it is defined.

If you use let inside of a block, i.e. a for loop, the variable is only available inside of that loop.

### Const

const is a variable that once it has been created, its value can never change.

const has a block scope.

# React Render HTML

React's goal is in many ways to render HTML in a web page.

React renders HTML to the web page by using a function called ReactDOM.render().

## The Render Function

The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

The purpose of the function is to display the specified HTML code inside the specified HTML element.

ReactDOM.render(<p>Hello</p>, document.getElementById('root'));

The result is displayed in the <div id="root"> element:

## The Root Node

The root node is the HTML element where you want to display the result.

It is like a container for content managed by React.

It does NOT have to be a <div> element and it does NOT have to have the id='root':

Example

The root node can be called whatever you like:

<body>

<header id="sandy"></header>

</body>

Display the result in the <header id="sandy"> element:

ReactDOM.render(<p>Hallo</p>, document.getElementById('sandy'));

# React 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.

Without JSX:

const myelement = React.createElement('h1', {}, 'I do not use JSX!');

ReactDOM.render(myelement, document.getElementById('root'));

JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

## 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:

## Inserting a Large Block of HTML

To write HTML on multiple lines, put the HTML inside parentheses:

### **Example**

Create a list with three list items:

const myelement = (

<ul>

<li>Apples</li>

<li>Bananas</li>

<li>Cherries</li>

</ul>

);

## One Top Level Element

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

So if you like to write two headers, you must put them inside a parent element, like a div element

JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.

# React Components

Components are like functions that return HTML elements.

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.

## 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.

## Create a Function Component

A Function component also returns HTML, and behaves pretty much the same way as a Class component, but Class components have some additions, and will be preferred in this tutorial.

## 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.

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.

class Car extends React.Component {

render() {

return <h2>I am a {this.props.color} Car!</h2>;

}

}

ReactDOM.render(<Car color="red"/>, document.getElementById('root'));

## 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;

## Pass Data

Props are also how you pass data from one component to another, as parameters.