TODO:
- React documentation
- FCC
- University of Helsinki’s Full Stack Open course

Introduction

ReactJs is an open source JavaScript library for creating user interfaces. It was created and maintained by Facebook.

It lets you write readable HTML directly within JavaScript.

When using React, all content that needs to be rendered is usually defined as React components.

React - Set Up

React uses its own markup language 'JSX' (a “syntax extension to JavaScript”)

JSX code must be compiled into JavaScript using a 'transpiler' like Babel.

To use React in your page you have two options:

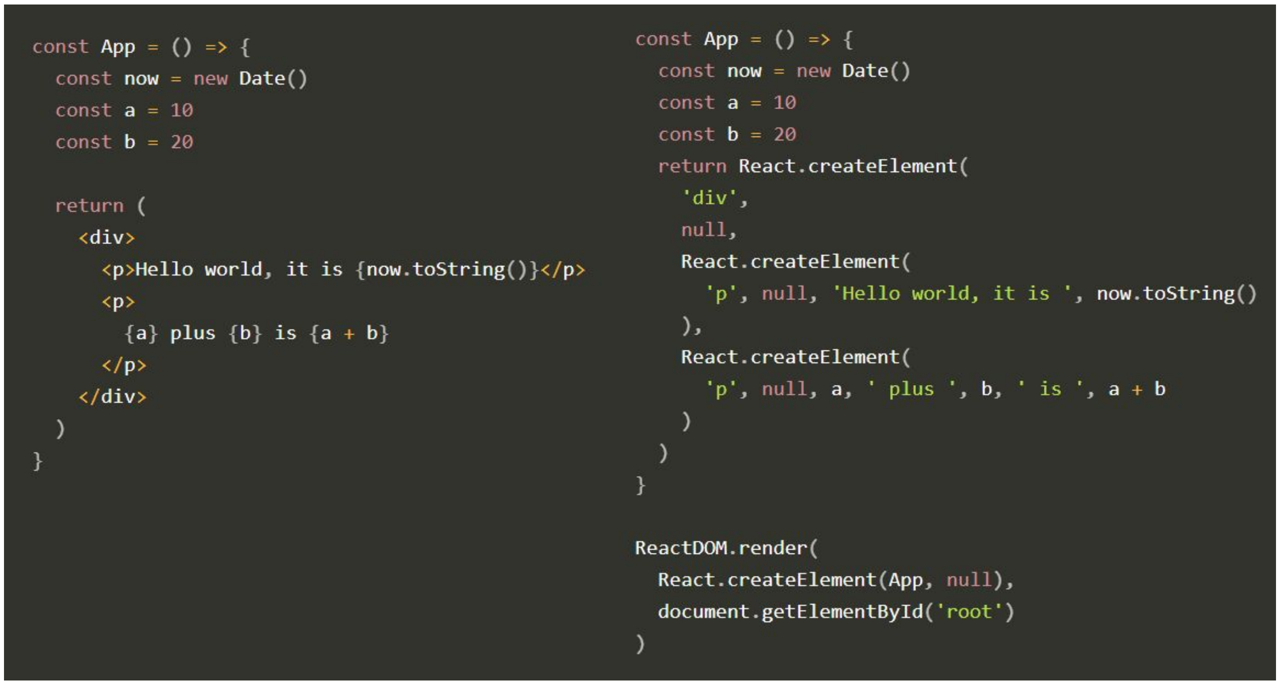
1. you can just use normal JavaScript syntax in a .js file (natively supported by the browser) plus cdns for React - but the syntax is more complex (writing React as "pure JavaScript" without using JSX). No one with a sound mind would actually do this.

2. Or equivalently, you can use "JSX" code (an actual *syntax extension* to javascript) which is neater/cleaner but requires a transpiler. For a quick cdn, use:
`<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>` and add `type = "text/babel"` to any script tag you want to use JSX in. However, this is slow

3. Faster/more complicated method is to download (with npm) a preprocessor that will convert your file to plain js.

Follow this

Remember - You need CDNs for React as well as for the transpiler! There are three in total!



```
const App = () => {
  const now = new Date()
  const a = 10
  const b = 20

  return (
    <div>
      <p>Hello world, it is {now.toString()}</p>
      <p>
        {a} plus {b} is {a + b}
      </p>
    </div>
  )
}
```

```
const App = () => {
  const now = new Date()
  const a = 10
  const b = 20
  return React.createElement(
    'div',
    null,
    React.createElement(
      'p', null, 'Hello world, it is ', now.toString()
    ),
    React.createElement(
      'p', null, a, ' plus ', b, ' is ', a + b
    )
  )
}

ReactDOM.render(
  React.createElement(App, null),
  document.getElementById('root')
)
```

Full Stack Open 2019 Method

https://create-react-app.dev/

- Tool is called create-react-app
- Installed in cmd (with npm and node.js already installed) with:

  npm install create-react-app

- To create an application:

```
$ npx create-react-app part1
$ cd part1
```

- To run the application

```
$ npm start
```

- By default, the application runs in localhost port 3000 with the address http://loalhost:3000
- Configured to compile with Babel automatically

For the code to run, you have to import:

```
import React from 'react'
import ReactDOM from 'react-dom'
```

Commenting

To add comments use `{/* */}`

To write vanilla JavaScript within JSX use curly brackets:

```
{ //blah blah blah this is some JavaScript }
```

Unlike HTML, you can easily embed dynamic content by writing appropriate JavaScript within curly braces.

React Elements

To create a React element:

```
const bro = <div> Hello World! </div>

const bro = (
    <div>
        <h1>This is a block of JSX</h1>
        <p>Here's a subtitle</p>
    </div>
);
//best practice to use parentheses
```

You cannot use "class" to define HTML classes in JSX (class is a reserved for object constructor functions in JavaScript). Use className instead (remember camelCase for everything)

```
const bro = (
    <div className = 'myDiv'>
        <h1>Add a class to this div</h1>
    </div>
);
```

In JSX, every tag can be self closing i.e. instead of <div> </div> you can just use <div />

Every tag needs to be closed, including <br />

These are simple React elements representing HTML tags. *You can also create React elements representing user-defined components (below).*

Rendering to HTML DOM

You need to render your JSX to the HTML DOM using React's rendering API - ReactDOM. ⇒ It is an object containing a .render() method you can use to render React elements to the DOM like this ReactDOM.render(componentToRender, targetNode). componentToRender = React element you want to render, targetNode = DOM node that you want to render the component to.

Must be declared after the JSX element declarations (like how you must declare variables before you use them)!

```
const bro = (
    <div>
        <h1>Hello World</h1>
        <p>Lets render this to the DOM</p>
    </div>

ReactDOM.render(bro, document.getElementById('target'));
```

React Components

React 'components' are (conceptually) like JS functions → they accept inputs (called "props") and return React elements (determining what is shown on the screen)
Components let you “split the UI” into independent, reusable pieces.

There are two ways to create a React component.

1) use a JavaScript function ⇒ creates a stateless 'functional' component.
We call these “function” components, because they are literally JavaScript functions.
'Stateless' because it can receive data and render it, but does not manage or track changes to that data.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}


const DemoComponent = function() {
    return (
        <div className='customClass' />
    );
};
```

2) Using the ES6 class syntax (for object constructor functions) to define a component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

```
class Kitten extends React.Component {
  constructor(props) {
    super(props);

  render() {
    return (
      <h1>Hi</h1>
    );
  }
}
```

Class constructor calls super(), which calls the constructor of the parent class (React.Component). Constructor is a special method called during the initialisation of objects created using class. Best practice is to call a component's constructor with super, and pass props to both. This makes sure the component is initialized properly.

React elements can also represent user-defined components:

```
const element = <div />;
const element = <Welcome name="Sara" />;
```

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}


const element = <Welcome name="Sara" />;

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

You can render dynamic content inside a component:

```
const App = () => {
  const now = new Date()
  const a = 10
  const b = 20

  return (
    <div>
      <p>Hello world, it is {now.toString()}</p>
      <p>
        {a} plus {b} is {a + b}
      </p>
    </div>
  )
}
```

Remember: functions in React must begin with a capital!
Remember: a React component must return only one root element! (“a root element is stipulated”) (e.g. wrapped inside a <div/>),

or an array of components!

```
const App = () => {
  return [
    <h1>Greetings</h1>,
    <Hello name="Maya" age={26 + 10} />,
    <Footer />
  ]
}
```

You can avoid having pointless extra “<div> </div>”s in the DOM tree by wrapping with fragments (empty elements) instead:

```
const App = () => {
  const name = 'Peter'
  const age = 10

  return (
    <>
      <h1>Greetings</h1>
      <Hello name="Maya" age={26 + 10} />
      <Hello name={name} age={age} />
      <Footer />
    </>
  )
}
```

Component composition/Multiple components

When you have created many components - you can create *parent* components that render other components as *children* - by including the child component's name as a tag in the render method of the parent. E.g.

```
const ChildComponent = () => {
  return (
    <div>
      <p>I am the child</p>
    </div>
  );
};

class ParentComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>I am the parent</h1>
        <ChildComponent/>
      </div>
    );
  }
};
```

Defining a component “Hello” and using it inside another component “App”: You can use a component multiple times!

```
const Hello = () => {
 return (
   <div>
     <p>Hello world</p>
   </div>
 )
}

const App = () => {
  return (
    <div>
      <h1>Greetings</h1>
     <Hello />
     <Hello />
     <Hello />
   </div>
  )
}

ReactDOM.render(<App />, document.getElementById('root'))
```

Remember to actually render React components to the DOM (by making a call to the ReactDOM API) - set a React element as = to the Component, like

```
let my-element = < component-name />;
ReactDOM.render(element, document.getElementById("target"));
```

or just go

```
ReactDOM.render(< component-name />, document.getElementById("target"))
```

Writing components with React is easy, and by combining components, complex applications can be kept fairly maintainable. A core philosophy of React is composing applications from many specialized reusable components. Think about your user interface in terms of components - breaking your UI down into building blocks

Another strong convention is the idea of a root component called App at the top of the component tree of the application.

Props

You can pass data to components using “props”. They can have any name you like, be any type of variable (a string, an array, an object).

You simply give the function defining the component a parameter/argument that represents the “props” object. (it’s standard practice to call the object itself “props”).

```
const Hello = (props) => {
 return (
     <div>
     <p>Hello {props.name}</p>
   </div>
  )
}
```

The function (defining the component) receives an object which has fields containing all the props the user of the component defines WHEN THE COMPONENT IS USED!

For example:

```
const Hello = (props) => {
  return (
    <div>
      <p>
       Hello {props.name}, you are {props.age} years old
     </p>
    </div>
  )
}

const App = () => {

 const name = 'Peter'
 const age = 10

  return (
    <div>
      <h1>Greetings</h1>
      <Hello name="Maya" age={26 + 10} />
      <Hello name={name} age={age} />
   </div>
  )
}
```

Another example:

```
const Welcome = (props) => <h1>Hello, {props.user}!</h1>

const App = () => {

  return (
    <div>
      <Welcome user = 'Mark' />
   </div>
  )
}
```

Obviously, like any argument, you can modify the “props” object’s values however you like (e.g. calling array methods like .join()). An example:

```
const List = (props) => {
  return <p>{props.tasks.join(", ")}</p>
};

class ToDo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>To Do Lists</h1>
        <h2>Today</h2>
        <List tasks = {["join the gym", "join the army", "go to
dinner"]}/>
        <h2>Tomorrow</h2>
        <List tasks = {["walk dog", "wonder about life", "do a
pullup"]}/>
      </div>
    );
  }
};
```

Default props

You can use default props (only assigned "if necessary" i.e. you haven't manually set it in the parent component) by accessing the component's defaultProps property (which is itself an object) e.g.

```
const ShoppingCart = (props) => {
   return (
     <div>
       <h1>Shopping Cart Component</h1>
     </div>
   )
 };

ShoppingCart.defaultProps = {items: 0};
```

To make sure components receive props of the correct type, alter the component's propTypes property: (remember, .propTypes property has a lowercase 'p', PropTypes...isRequired has an uppercase 'P')

```
import React, { PropTypes } from 'react';

const ShoppingCart = (props) => {
      return (
        <div>
          <h1>Shopping Cart Component</h1>
        </div>
      )
    };
    ShoppingCart.proptypes = {handleclick: PropTypes.func.isRequired,
free: PropTypes.bool.isRequired};
```

```
const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h>
};
```

```
Items.propTypes = {
  quantity: PropTypes.number.isRequired
}

Items.defaultProps = {
  quantity: 0
};
```

```
class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Items />
  }
};
```

To pass props to an ES6 class component: you can use the this keyword to refer to the class component itself within itself, to access props use this.props.propName. Remember, you still actually only give/assign a child component a property when you render it in the parent component, and THEN within the child component you access that property:

```
class ReturnTempPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
        <div>
            <p>Your temporary password is:
>strong>{this.props.tempPassword}</p>
        </div>
    );
  }
};

class ResetPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
        <div>
          <h2>Reset Password</h2>
          <h3>We've generated a new temporary password for you.</h3>
          >h3>Please reset this password from your account settings
ASAP.</h3>
          <ReturnTempPassword tempPassword = "mypassword" />
        </div>
    );
  }
};
```

State

A *stateless functional component* is a function which accepts props and returns JSX. A *stateless component* is a class that extends `React.Component` but does not use internal state. A *stateful* component is any component that does not maintain its own internal state

`State` is any information your application needs to know about, that can change over time. You want apps to respond to state changes and present an updated UI when necessary. This is called "state management".

To give a React component `state`, declare a state 'property' on the ES6 component class, in the class' `constructor` function (i.e. like any property, when the class is called to create an object - 'initialising the component' - it is created with a `state` property). The `state` object must be set as = a javascript object, like this:

```
class StatefulComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "John"
    };
  }
  render() {
    return (
      <div>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
};
```

State allows you to track important data in your app and render a UI in response to changes in data. React uses a Virtual DOM to track changes behind the scenes. When state data updates, it triggers a re-render of components using that data.

In the render method, before the return statement, you can write any javascript you want, i.e. declare a function, alter the data in state or props, assign data to variables, which you have access to in the return statement.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            name: 'freeCodeCamp'
        }
    }
    render() {
        const name = this.state.name;
        return (
            <div>
            <h1>{name}</h1>
            <div>
    );
    }
};
```