**React State @ 21 Jun 2019**

create-react-app clock

yarn start

in index.js, add the following

function tick() {

  const element = (

    <div>

      <h1>Hello World</h1>

      <h2>It is {new Date().toLocaleTimeString()}.</h2>

    </div>

  )

  // re-render this function every second

  ReactDOM.render(element, document.getElementById("root"))

}

setInterval(tick, 1000)

**Encapsulate Component**

Encapsulate the visual aspect of the clock in a separate component

Take out the visual part of the clock into a separate file called clock.jsx

import React from "react"

export default function Clock(props) {

  return (

    <div>

      <h1>Hello World</h1>

      <h2>It is {new Date().toLocaleTimeString()}.</h2>

    </div>

  )

}

Modify index.js

import Clock from "./Clock"

function tick() {

  ReactDOM.render(<Clock />, document.getElementById("root"))

}

**To make clock more reuseable**

Modify index.js

  ReactDOM.render(<Clock date={new Date()} />, document.getElementById("root"))

Setup the interval in the component

Modify clock.js

      <h2>It is {props.date.toLocaleTimeString()}.</h2>

**State**

All the instance variables, properties collectively constitute the current state of that object

If we change the internal instance variables, the state of the object is changed

Modify clock.jsx to a class component to use lifecycle methods and states

export default class Clock extends React.Component {

  render() {

    return (

      <div>

        <h1>Hello World</h1>

        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>

      </div>

    )

  }

}

* Props become .this
* It is only useful for when there is state and not just props (display straight data)

To add state to the class

* Class component will have an internal state
* Constructor initialises the class
* React will call the constructor once when it creates an instance of the class component (i.e. when it is called here
* <Clock date={new Date()} />
* Render will be called when you update the component dynamically

How to avoid re-rendering the whole DOM – Point of React (only re-render the parts that are being changed)

**LifeCycle methods**

The super is React.component

Establish an initial state

Then calls render, renders the initial display of the component

And then calls componentDidMount

Each seconds this.tick, which changes the date to a new date

We use setState that we are modify the state, and tells react to re-render the DOM

The setsState then takes the difference in value in state in render and the existing DOM, and just renders the difference and leaves the rest untouched