**Lecture Review Question**

***Part 1:***

In React, Props include data that is passed to a component, with a component being a function that returns HTML elements. React Props act as function arguments in JavaScript as well as attributes in HTML. Props can only be passed from the parent to a child component and are read only, meaning the parent data cannot be changed by the child component. The example below shows how data can be passed from one component to another. Here, the ‘name’ property is passed from the College component to the Campus component.

class Campus extends React.Component {

    render() {

      return <h2>I study at {this.props.name} </h2>;

    }

  }

  class College extends React.Component {

    render() {

      return (

        <div>

        <h1>What college do you attend?</h1>

        <Campus name="Kevin" />

        </div>

      );

    }

  }

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

State is a built-in JavaScript object that stores a component’s dynamic data. It is this dynamic data that enables a component to keep track of altered data in between renders. The component gets re-rendered when the state changes. Unlike Props, State is private to a component, meaning other components cannot access that state. The state object of a component is defined in its constructor and can be referred to anywhere inside the same component. The ‘this.setState()’ method can be used to update the value in the state object. In the example below, the state value name is initialized in the constructor and is changed from ‘Kevin Street’ to ‘Anguier Street’ using the method ‘setState()’ inside the onCLick event attached to the button.



***Part 2:***

In functional programming, a functor is something that behaves consistently when subjected to a function. Put simply, a functor is a set of values that can be mapped over and is often called a ‘Mappable’ as a result. A functor can be created out of most values. Examples of which are single values, strings, objects and even functions. One of the most common uses of a functor in functional programming however is the mapping of a function over of an array as shown below.

console.log([ 2, 4, 6, 8 ].map(x => x + 10))

The mapping of the defined array then returns an output of [ 12, 14, 16, 18 ]. This output is produced as each value within the array gets 10 added to it inside its map function.

JavaScript objects differ to that of an array in that they don’t have a .map() method in which to iterate over. Therefore, a functor has to be created so that each value inside the wrapper object can be mapped. In the example below, it is called ObjectMapped.

class ObjectMapped {

    constructor (object) {

      this.object = object

    }

    map (param) {

      const mapable = { }

      for (const key of Object.keys(this.object)) {

        mapable[key] = param(this.object[key])

      }

      return new ObjectMapped(mapped)

    }

}

In this example, a function gets applied to each value within the wrapped object. Each value is then put in a new wrapper which in turn wraps an object of the same type.

***Part 3:***

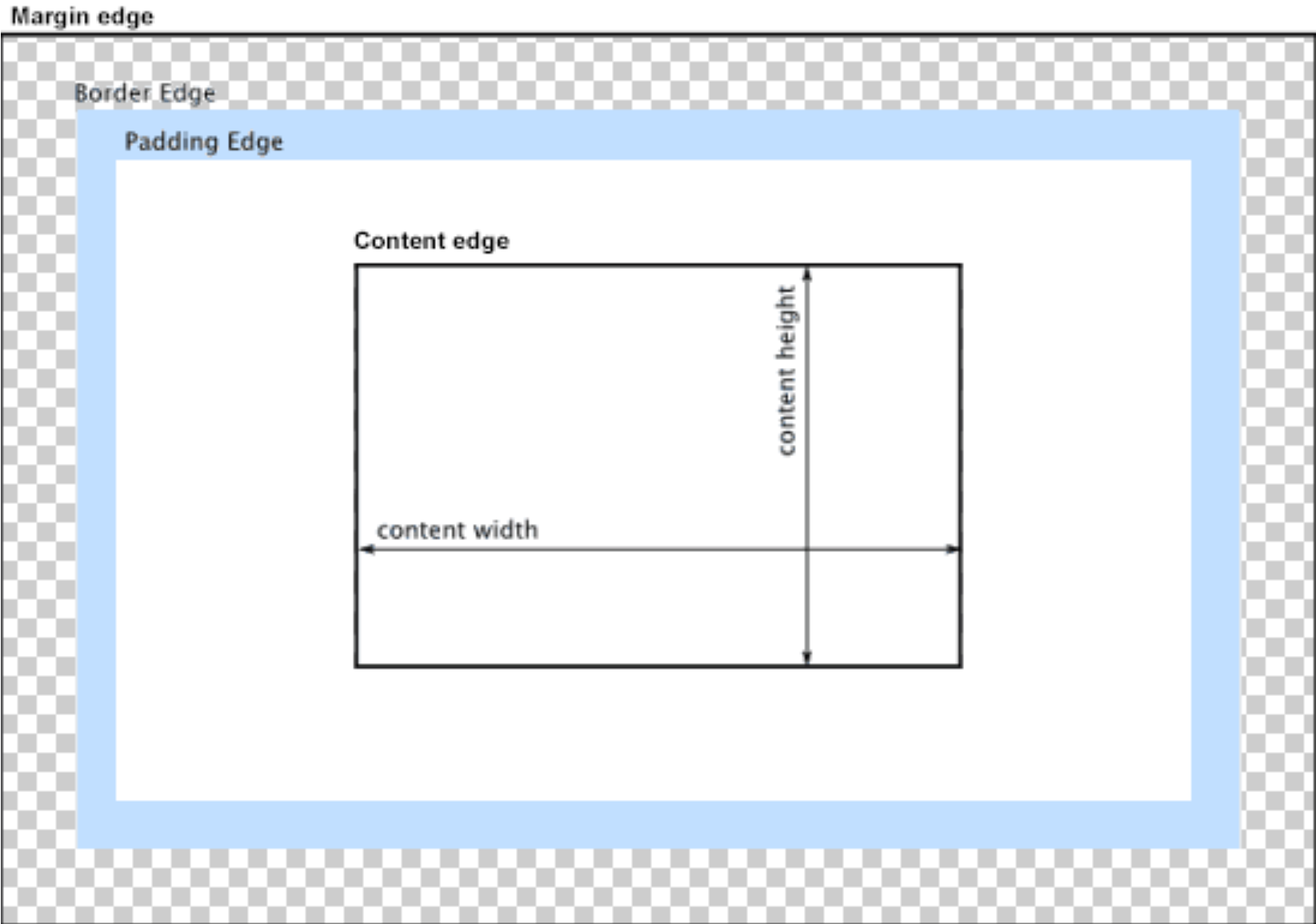
A callback is a function which gets implemented after another function has finished executing. An advantage of using callbacks is that although JavaScript is singled threaded, that thread never gets tied up in a waiting state. This is beneficial when tasks may take some time to be executed. One disadvantage of using callbacks is referred to as ‘callback hell’ as it includes a great amount of nesting which results in code becoming very difficult to follow. This ultimately causes code using callbacks to become harder to follow and more importantly, test.

A promise is an object that may produce a single value some time in the future. One advantage of using promises includes that its possible to chain promises, allowing for precise control over how and where errors are handled. This is accomplished using methods such as .then() and .catch(). Possible disadvantages of using promises include that there is a substantial amount of theory involved in understanding the handling of asynchronous tasks. This ultimately means that promises can prove to be difficult to debug.

A stream is an object which enables the moving of data from a source to its destination by sending it asynchronously in chunks. An advantage of implementing streams includes that it improves both, the time and space efficiency of the application. It accomplishes this by asynchronously sending chunks of data from a source to its destination. One disadvantage of executing streams includes that it is not the most optimal choice in terms of performance. A for-loop is extremely lightweight in comparison to a stream, both in terms of heap and CPU usage. Debugging can also be a more time-consuming process when compared to a more traditional for-loop.

***Part 4:***

All HTML elements can be considered as boxes. The CSS Box Model is a box which wraps around each HTML element and is made up of margins, borders, padding and its content as shown below.



The content of the box comprises of text and images whereas the padding serves to remove an area around the content. The border of the box then extends the padding area so that it includes the elements borders. Finally, the margin area goes around the border, padding and content and is used to separate the element from its neighbours. This can be represented as CSS code as follows:

div {

    background-color: grey;

    width: 230px;

    border: 8px solid blue;

    padding: 35px;

    margin: 10px;

  }

The CSS Box Model can be used to space DOM elements using calling the created spaceDOMElements() method below and styling the appropriate DOM element using the displayed syntax.



***Part 5:***

When a user visits a URL, their browser sends a request to the server to fetch an HTML document with the server returning an HTML page in binary stream format consisting of a text file with a response header ‘content-type’ containing the value ‘text/html’. This tells the browser that it is an HTML document and using the returned information, the browser can convert the binary format into a readable HTML document.

When the browser reads the HTML elements (such as html, head, body) from the HTML code, it creates a JavaScript object called a Node. After the browser creates Nodes from the HTML document, it will create a tree-like structure called a Document Object Model (DOM) of these nodes which will help the browser efficiently render the webpage.

After constructing the DOM, the browser reads in the various CSS sources provided in the HTML document. In our case, it will read in the appropriate bootstrap libraries. After doing this, the browser constructs a CSS Object Model (CSSOM). Each node within this tree contains CSS style information for that DOM element.

The Render-Tree then combines the DOM and CSSOM tress into one tree-like structure, and once it is constructed, the browser starts to display each element on the screen. The process in which the browser positions each Render-Tree node on the screen is called reflow. The layout created details the size and position of each node.

Elements in the Render-Tree can have properties that cause them to frequently change the look or position, the browser creates a layer for it. This process is referred to as ‘paint operation’ and creating layers enable the browser to efficiently perform painting operations throughout the web application’s lifecycle. These layers can be now drawn separately first, then combined to be drawn to the screen.

The final step is called the Composing Operation and it sees these layers being sent to the GPU to be finally displayed on the screen.