# 1.

Props(Properties) – These are the parameters that passed into a component . Use this.props to control the variable. These props are read-only as well so use state to change the props. E.g. isEdit – are we editing the note?

Text

Description automatically generated

State – These are objects that are like variables that remember data/ information about the component specified. The state is often changed. You can initialise the state in the constructor. Use setState to change the state of the component and to refresh the component. E.g. OnEdit we set isEdit to true as we are editing the note.

Text

Description automatically generated

## 2.

With functional programming we call a type of object that can be matched directed to another value a functor. Arrow Functions in javascript are mostly associated with functors. Functors are also abstract because they change and clone one value to another future value.

E.g. Filter

Text

Description automatically generated

I will continue to filter the note id until I find the id that I want to remove.

# 3.

Callback-

They are simple and easy to understand and make it easy to manage async operation in javascripts. Newer developer who be more likely to understand callbacks.

They also have a problem when using multiple async operations to be recursive to the point where it becomes too nested to difficult to follow and understand.

Promise –

Have a more simple and easy to look at syntax which manage async operations. They allow us to merge async operations in a similar simple and easy to look at syntax.

They can be difficult to understand for newer developer who do not fully understand asynchronous programming yet. It is much different than the simple synchronous code which runs gives you results directly and will not continue until then.

Streams. -

With steam you can manipulate data in a stack like manner compared to access the full contents of a dataset. This can make things efficient.

Streams lack error handling which require the developers to implement it.

# 4.

The CSS box model is a model which describes the layout and display of an element from its inside e.g.(the actual) content towards its outside e.g. margin. A series of boxes are used to depict the makeup of the model. All of them have a top, bottom,left and right side.

Margin is the extra space which surrounds the box and the other elements within the same page. The margin is normally invisible.

The border is default an invisible line which surrounds the element which acts as a wall between it and the other elements. The borders’ properties such as color, width and style are normally modified to add aesthetics.

Padding would be the space between the border and (the actual) content. The padding is normally invisible.

The content is which is supposed to be displayed on the page by the element e.g. an image or text. Margin, border, padding respectively surround the content which is on the very inside of the CSS box model.

Below is a diagram which depicts them and further below is code and diagrams to illustrate how they are moved.

Diagram

Description automatically generated

Graphical user interface, text

Description automatically generated

Diagram

Description automatically generated



Graphical user interface, text

Description automatically generated

Diagram

Description automatically generated



.element {

    /\* All sides of the padding of the element will be given 10px \*/

    padding: 10px

}

Diagram

Description automatically generated



# 5.

A browser load and bootstraps a rich web application using an initial URL in these follow steps.

1. Browser needs to communicate with the local/remote web server via a request which the specific URL acts a location instruction.
2. The webserver will communicate back by a response which will contain the HTML page.
3. The browser will translate the html into values it can understand and will use the document object model (structure) in order to replicated the structure of the Html page.
4. The browsers will look at the CSS styles which gives the layout of the DOM elements.
5. The browser will send additional requests to the webserver for all resources which has been referenced on the page e.g. Javascript file, images, CSS stylesheets. This happens while the browser is translating the html.
6. The browser will execute any inline javascript or external javascript after all the html, css and other resources have been loaded.
7. The executed javascript will start the rich web application and will set up event listeners, make api calls and etc when needed.
8. The rich web application can also do the other activity from step 7.
9. These steps are repeated with every new navigation with URL.