***Part 1***

Props are data structures that behaves in a similar way to parameters in pure JavaScript.

They are passed to the items that need them to function in a particular way. The state is the collection of the items stored in the component, behaving like a class with a number of fields to store items.

###Props example

const Profile = (props) => { return (

<div style={style}>

<img src={props.logo\_url} alt = "sample-text-img-for-props" height="250px"/>

<h1>{props.title}</h1>

</div> );

}

###State example

class Profile extends React.Component {

constructor(props){

super(props)

this.state = {"show\_technologies": false}

this.see\_our\_technologies = this.see\_our\_technologies.bind(this);

}

***Part 2***

The term "functor" refers to the collection of different types of objects, from int to strings. These items can be mapped from, and copies of the same values can be passed into different methods themselves.

Functors are very important all across java script

###JS Example

['123', '456', '789'].map(s => parseInt(s)).map(n => n / 10)

***Part 3***

*Callbacks*

Advantage:

> Ensures the application the continue interacting with the user within the application while retrieving the requested file

Disadvantage:

> If there is a necessity of further callbacks dependent on the initial one, the layout can quickly become inefficient and highly coupled, which is unreliable in a timing application

*Promises*

Advantage:

> Allows the application to be decoupled while ensuring a result is returned indicating the success of the contract of the agreement between the application and the associated API. This is by storing the promise within a variable itself.

Disadvantage:

> When the promise is returned, the application i/o and performance is interrupted to handle the promise request

*Streams*

Advantage:

> Streams allows data to be processed on once retrieved from any source which the stream decided to get the source from, without the client needing to working about it

Disadvantage:

> The data returned could be processed in an unexpected way which the application would not know what to do, making debugging and maintenance difficult to do with streams

***Part 4***

The CSS box model describes the layout which we can use to place spacing between certain dom elements. It captures the details in a box format. The elements, from the most exterior side, can be described as margin, border, padding and content. The margin separates the element from other elements in the exterior way. The border describes the external layout which separates from other dom elements. The padding separates the content of the dom element from the border itself and the content is the elements which is to be displayed to the user themselves.

div {

  width: 320px;

  padding: 10px;

  border: 5px solid grey;

  margin: 0;

}

***Part 5***

When the URL is reached through an http request, the application will start loading the assets that is needed from the server onto the browser engine. The assets are a combination of HTML5, CSS3, JavaScript libraries and other multimedia needed.

As each file is loaded, the engine parses the file and completes the immediate tasks required of the applications, such as fetching other resources needed in the immediate future. After running the necessary commands, the application is built using the provided tools, such as how the html decides the view would look like combined with the styling instructions from the CSS files. The process of fetching, building, and styling repeats until all the files needed have been fetched.