Understanding React JS Library

Goal Tracking Application

Session Dependencies

- You will need to ensure that you have installed node.js.
- While you can use any text editor for this session, I recommend that you install VS Code

Why React

I am very excited to be introducing React to you this week.

According to the React (opens new window)documentation, "it makes it painless to create interactive UIs." Having used React in a production business environment I certainly agree with this sentiment.

Speaking broadly, React offers the following benefits:

- A light layer of functionality on top of JavaScript
- An experienced React developer is also an experienced JavaScript developer
- We can program in React-Native to make native mobile application
- However, at the time of writing React-Native is somewhat falling out of fashion

By the end of this week you should be able to address the following questions:

- What is React?
- What are React components?
- What are props?
- What is JSX?
- How can we iterate over a list to output components?
- How do you create a compositional component and what is the point?
- How can I apply conditional styling to components?
- In what direction does data flow through a React Application?

In order to do this we are going to start working on our class project - a goal tracking application

Creating your first React app

`npx`, just like `npm`, is a tool that is installed with a node. `npx` allows us to run executables that are stored in the `npm` repository. We can use `npx` to quickly scaffold a React application:

- the React job market is buoyant and developers are demanding high salaries

Task 1 Create your first react application 🚀

1.1 From within command line run:

```
npx create-react-app goal-app
cd goal-app
npm start
```

1.2 Version control your work and upload it to a GitHub Repo

1.3 See if you can change the home page of your App to "Hello World" and import a different picture.

React Components

React apps are created by composing a series of reusable components - everything in your app is a component. They allow you to split your UI into a series of reusable pieces.

Conceptually a component in react is a JavaScript function.

```
function Welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}
```

You can also use a ES6 class to define a component, but this is quite dated now:

```
class Welcome extends React.Component {

  render() {

    return <h1>Hello, {this.props.name}</h1>;

  }

}
```

Both of the techniques above create equivalent components. Historically, class components differentiated themselves from their functional equivalent in that only a class component could access state and lifecycle methods. However, React 16.8. introduced hooks (opens new window)that democratized functions giving the capabilities of components. As such, there is no need in 2022 to continue using classes - functions are easier to use and more concise.

#JSX

Some of you may have already noticed that something odd is going on with regard to our JavaScript. For instance, `return <h1>Hello, {props.name}</h1>;` is not valid JavaScript, it is known as JSX.

JSX is a syntax extension to JavaScript and allows you to combine the full power of JavaScript to construct views. Let's consider a more complete example in updating our `src/App` function:

```
function App() {

  const name = "Amira";

  const heading = <h1>Hello, {name}</h1>;

  const sum = (x,y) =>  x + y;

  return (

  <div>

    {heading}

    <h2> Yo {name} </h2>
    <h2>  What is the answer to 1 +1. Is it {sum(1,1)} ?
</h2>

  </div>

  );

 }

}
```

There are a few points of note to understand in the above example

- `return(...)` the parentheses `()` is ES6 and allows us to return on multiple lines
- To nest variables and expressions within our JSX we must use curly brackets (e.g. `{name}`)
- There must be a single-parent set of tags within the return statement. In this instance - `<div> ... </div>`
- Every reacts component **must** return something

TIP

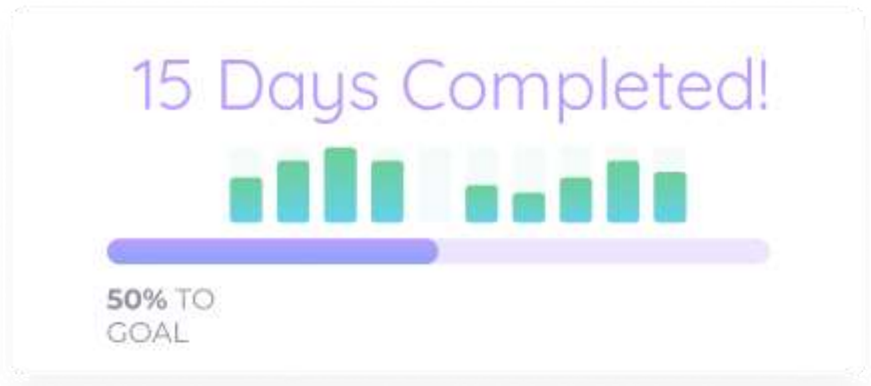
#Task 2 Use JSX 🚀

As I did above, create a new React application and add some JSX to your `App.js` file.

#Components and Props

Throughout the learning, we are going to be making a goal tracking application. You can see a mock-up of the application here (opens new window). The premise is we track our goals and are held accountable to our friends for a fix period of time. For a set period of time (e.g 30 days), we log our habits and receive a score out of 20.

Today, we are going to work on a very small piece of this application - the part that will track the number of days completed (see below).



Let's start thinking about how we would represent the above section of the interface as a component.

#Days Completed Component

Typically a single folder or file represents a component, all of these components will live in a `src/Components`

TIP

#Task 3 Set Up a Component 🚀

- Crate a folder `Components` in the src folder - 'src/Components'
- Create the file `src/Components/DaysCompleted.js`
- Add the following code to `DaysCompleted.js`

```
// src/Components/DaysCompleted.js

import React from "react";
import PropTypes from "prop-types";

function DaysCompleted(props) {
  const { days } = props;

  return (
    <div>
      <h1> {days} Days Completed</h1>
    </div>
  );
}

DaysCompleted.propTypes = {
  days: PropTypes.number.isRequired,
};

export default DaysCompleted;
```

In `App.js`, let's use our component. Within `src/App.js`:

- Import our new component `import DaysCompleted from './Components/DaysCompleted'`
- We can now use it in our `App` function like this:

```
function App() {
  return (
    <div>
      <DaysCompleted days={15} />
    </div>
  );
}
```

#Props

Components can accept inputs know as `props`. Props are one of the main mechanisms to facilitate data flow around our application.

WARNING

Prop data in a React application can only flow from a parent component to a child.

We pass a prop value to a component through the use of the given components attribute. In this case, we are passing 15 to the component DaysCompleted.

```
<DaysCompleted days={15}/>
```

React will then construct a Prop object and pass it in as the first argument in our functional component. Notice how, within `DaysCompleted.js` we use a technique know as destructuring to extract the value `days` out of our props object:

```
const { days } = props;
;
```

The above is the equivalent of:

```
const days = props.days;
```

#Prop Types

Since each component could in theory have any number of props, we can use `PropTypes` to describe props that our component intends to receive:

```
DaysCompleted.propTypes = {
  days: PropTypes.number.isRequired,
};
```

Using the `PropType` above, if days is not passed in as a prop you will see a warning in the console. You can, of course, validate different types (opens new window)(e.g. `PropTypes.array`, `PropTypes.bool`, `PropTypes.number`)

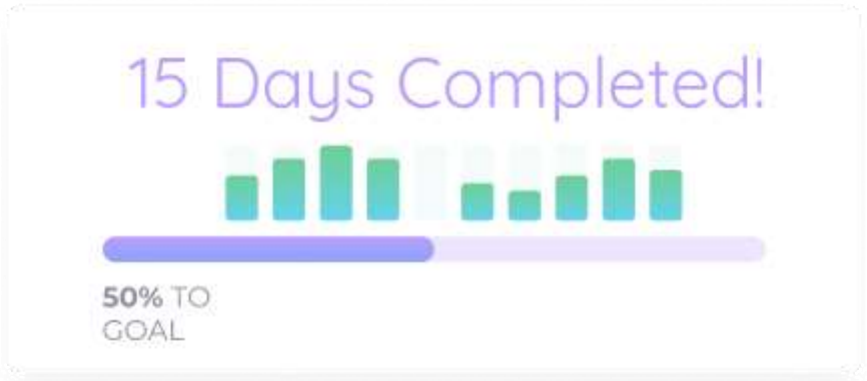
If a `Prop` is not required, you should ensure that you set a default PropType. We can do this as follows:

```
DaysCompleted.propTypes = {
  days: PropTypes.number,
};

DaysCompleted.defaultProps = {
  days: 0,
};
```

#Wrapping one component into another

Hmmm, why would you ever want to do such a thing? Let's take another look at the component that we are creating:



Can you see the border is slightly raised, creating a tile effect? This effect will need to be re-created throughout our application. As such, we should abstract this tile into a component. We can do this by creating a compositional component.

TIP

#Task 4 Compositional components 🚀

Let's create a wrap-around component - also known as a compositional component. Create a new file `Components/Tile.js`. Add the following code:

```
import React from "react";

function Tile(props) {
  const { children } = props;
  const divStyle = {
    boxShadow: "0px 10px 20px rgba(31, 32, 65, 0.05)",
  };

  return <div style={divStyle}>{children}</div>;
}

export default Tile;
```

The goal of tile is to wrap it around another element or component, it would work like this: `<Tile> <SomeElement/> </Tile>`. This can be achieved by taking advantage of the fact that React injects `<SomeElement/>` into the child property of our `props` object.

See if you can work out how to update `<DaysCompleted>` so `<Tile>` is wrapped around it.

TIP

#Task 5 Styling the days completed text 🚀

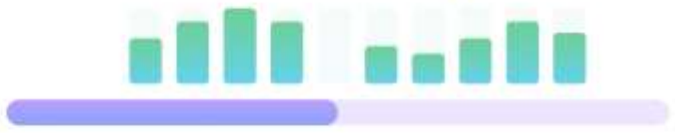
Can you work out how to style the days completed text?

TIP

#Task 6 Optional - Have a go at finishing off the component 🚀

Tip: You should look to compose your component using multiple sub components

15 Days Completed!



50% TO
GOAL