React, Getting Started

Content

- create-react-app
- Basic JSX
- Dividing up into components
- Basic state
- Render from basic state
- Conditional render
- Passing props
- Functions, and event handling

Setup

We need to check that we are starting our React projects outside of any Git repo, as we will be using `create-react-app` which will bring down everything we need for the project, including starting up a Git repo.

https://reactjs.org/docs/create-a-new-react-app.html

Commands:

```
> npx create-react-app <project-name>

> cd <project-name>

> yarn start
```

These steps pull down and execute all we need from create-react-app, and we then step into the folder. The last command kicks off the server at `localhost` port `3000`.

neil-orange-peel

colours.neilorangepeel.comcolours.neilorangepeel.com

I like this guy for getting some easy colour into the project to get things kicking quickly. As we will be dealing in components, it's nice to use a few colours to see what we are dealing with.

Thinking in Components

First off we just need a simple way to start thinking in components, which is how React encourages us to think. To do this we are setting up a simple portfolio app and going to build it out in React.

You can use your portfolio, or my example as you like.

It's just going to be one page for now. We first wrote it in plain HTML (JSX), and then we are going to transform that into components so that we can start pulling it apart.

This is a version half-way through the process. This is a class component, and it is also the main component `App.js`. Also note the `<About />` component:

```
import './App.css';
import { Component } from 'react'
import About from './components/AboutComponent'

// class component
class App extends Component {

    // the render function returns JSX
    render() {
        console.log("The page is rendering.")
        return (
            <div className="App">

                <header className="App-header">
                    <h2> Portfolio </h2>
                </header>

                <main className="main-section">

                    <div className="main-heading">
                        <h3> Main Section </h3>
                    </div>

                    <div className="main-wrapper">

                        <About />

                        <div className="projects-section">
                            <h3> Projects </h3>
                            <div> Record Store </div>
                            <div> Language Tree </div>
                        </div>

                        <div className="contact-section">
                            <h3> Contact </h3>
                            <p> Address: somewhere </p>
                            <p> Phone: 000 888 777 </p>
                        </div>

                    </div>

                </main>

            </div>
        );
    }
}

export default App;
```

This is the `About` component, and it has `Bio` as a dependency.

```
// functional component
import Bio from './Bio.js'

const About = () => {

    return (
        <div className="about-section">
            <h3> About </h3>
            <Bio />
            <p>
                I'm enjoying learning to code and looking for
opportunities.
            </p>
        </div>
    )
}

export default About;
```

And here is Bio

```
const Bio = () => {
    return (
        <p>
            This is the bit of stuff about me. I like tennis and coding
and music etc.
        </p>
    )
}

export default Bio;
```

I'll also give you my `App.css` as an example. This is far from the finished product, but a quick smash-out to get the components on to the page and in some places that will help flesh out the ideas.

```
header {
  height: 20vh;
}

.main-section {
  background-color: green;
  height: 80vh;
  display: flex;
  flex-direction: column;
  justify-content: space-around;
  align-items: center;
}
```

```
.main-heading {
  height: 10%;
}

.main-wrapper {
  background-color: darkslategrey;
  width: 100vw;
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-items: center;
}

.about-section {
  width: 30%;
  background-color: maroon;
}

.projects-section {
  width: 30%;
  background-color: mediumseagreen;
}

.contact-section {
  width: 30%;
  background-color: lightcoral;
}
```

I'll leave it as a task for you to finish making all the components for the next section.

State

We are now going to add in `state`, which is like a data store that will be available in our `App` component. It's best to think of data flows in React as coming from the top-most node in the hierarchy (for us, `App.js`), and then flowing down into the other components. For now, `App` is going to hold the only state, the only data source, and we will release it down to the other components only as needed.

State also provides some data to be rendered to screen (very simple here), or can hold the state of a particular component (for example, whether it is being shown or not). You can see examples of this below.

The last element is how we can get state into the children of `App`. We do this by passing `props`. We attach data to keys on the element, and pass it on through.

```
import './App.css';
import { Component } from 'react'
import MainWrapper from './components/MainWrapper'
import Header from './components/Header'

class App extends Component {

  state = {
```

```
      mainSectionHeading: "This is the Main Section Heading",
      mainSubHeading: "This is a SubHeading",
      showHeader: true,
      missionStatement: "I'm coding and looking for work."
    }

    render() {
      console.log("The page is rendering.")
      return (
        <div className="App">

          <div>
            {
              (this.state.showHeader)
              ?
              <Header />
              :
              <h3> No header shown </h3>
            }
          </div>

          <main className="main-section">

            <div className="main-heading">
              <h3> { this.state.mainSectionHeading } </h3>
              <h4> { this.state.mainSubHeading } </h4>
            </div>

            <MainWrapper mission={ this.state.missionStatement }
hardCodedString="Hard Coded String" />

          </main>

        </div>
      );
    }
}

export default App;
```

Next we have the wrapper with props coming through. We are just passing the props on down to their final destination. You can see here that we can get hold of the data through this `props` object provided to us, and accessing the keys we decided on in `App`.

In this example we are rendering `About` twice (because we can), and sending it different props each time (again, just because).

```
import About from './AboutComponent.js'
import Projects from './ProjectsComponent'
import Contact from './ContactComponent'
```

```
const MainWrapper = (props) => {

    console.log("PROPS IN MAINWRAPPER")
    console.log(props)

    return (
        <div className="main-wrapper">

            <About mission={props.mission} />
            <About mission={props.hardCodedString} />

            <Projects />

            <Contact />

        </div>
    )
}

export default MainWrapper;
```

Here we see the data's final destination in `About`. We are getting our bits of the state object passed on through. This will get inefficient in a large app, but for now things are simple, and the main indea is to understand that we are passing data down through the app from the top of the tree out to the branches.

Note too that we only have the one `About` component, but we are able to render it with different content depending on the props being passed to it. This will become more pertinent in the future, but important to get used to early on in this process.

```
import Bio from './Bio.js'

const About = (props) => {

    return (
        <div className="about-section">
            <h3> About </h3>
            <Bio />
            <p>
                { props.mission }
            </p>
        </div>
    )
}

export default About;
```

Now we are going to extend the `App` component that is going to handle a click event for a component. There are several key aspects to what is going on here.

Firstly, our click event could do anything we like, but in this instance it is going to change an aspect of `state` that will have consequences for what is being shown on the page. You should never change `state` directly, and rather you should use `this.setState((prevState) => {...})`.

One of the arguments given to `setState`'s callback is the previous state, which should be used to access any values from `state` that are required for you to make the change.

In this example we are toggling the boolean that governs whether a heading is shown. In another example it could be incrementing a value, or any number of possibilities. Again, do **not** modify `state` directly - use `setState`. And when modifying `state` based on a previous value, use the `prevState` version of `state` given to the callback.

```
import './App.css';
import { Component } from 'react'
import MainWrapper from './components/MainWrapper'
import Header from './components/Header'

class App extends Component {

  state = {
    mainSectionHeading: "This is the Main Section Heading",
    mainSubHeading: "This is a SubHeading",
    showHeader: true,
    missionStatement: "I'm coding and looking for work.",
    address: "15 Props St, Propsland"
  }

  // handleHeaderClick() { // normal looking function
  handleHeaderClick = () => {
    this.setState((prevState) => {
      const toggle = !prevState.showHeader
      return {
        showHeader: toggle
      }
    })
  }

  render() {
    console.log("The page is rendering.")
    return (
      <div className="App">

        <div onClick={ this.handleHeaderClick } >
          SHOW OR HIDE HEADER
        </div>

        <div>
          {
            // if (this.state.showHeader) {
            (this.state.showHeader)
            ?
            // Do this
```

```
            <Header />
            :
            // else do this
            <h3> No header SHOWN </h3>
          }
        </div>

        <main className="main-section">

          <div className="main-heading">
            <h3> { this.state.mainSectionHeading } </h3>
            <h4> { this.state.mainSubHeading } </h4>
          </div>

          <MainWrapper mission={ this.state.missionStatement }
hardCodedString="Hard Coded String" myAddress={ this.state.address } />

        </main>

      </div>
    );
  }
}

export default App;
```

Other elements on focus on here include the ternary operator, which functions as an if else statement within the JSX. It can be seen here:

```
<div>
    {
        (this.state.showHeader)
        ?
        <Header />
        :
        <h3> No header SHOWN </h3>
    }
</div>
```

In more regular JS, this would look more like this:

```
if (this.state.showHeader) {
    render(<Header/>) // just an example function
}
else {
    render(someOtherThing) // and another example for illustration
}
```

I've fudged the JSX, but hopefully the point comes across. We are going to render one or other thing, and which thing is rendered will be based on the boolean at `this.state.showHeader`. We can also render nothing when the header is not shown, but we are rendering something so that we can see both conditions and more easily understand the idea.

The final element of note here is this segment:

```
<div onClick={ this.handleHeaderClick } } >
    SHOW OR HIDE HEADER
</div>
```

The first thing to get very clear is that the way React works is to run the `render()` function any time the state is changed. **Commit this to memory, as this is a vital aspect of React and key to how it all functions.**

In this example, we are changing `state` when this `<div>` is clicked using React's `onClick` event, and attaching a function to it. The function (as discussed), changes the state, which in turn runs `render()` again.

Be careful of not doing the following:

```
<div onClick={ this.handleHeaderClick() } >
```

If we do the above, the code within the `{ }` is evaluated, meaning that `this.handleHeaderClick` is run. Running it changes `state` which in turn triggers `render()`, and in turn triggers a state change, and triggers `render()` and changes `state` and triggers `render()` and..

We need to pass the whole function, and React/the browser will call the function at the appropriate time (on a click in this instance).