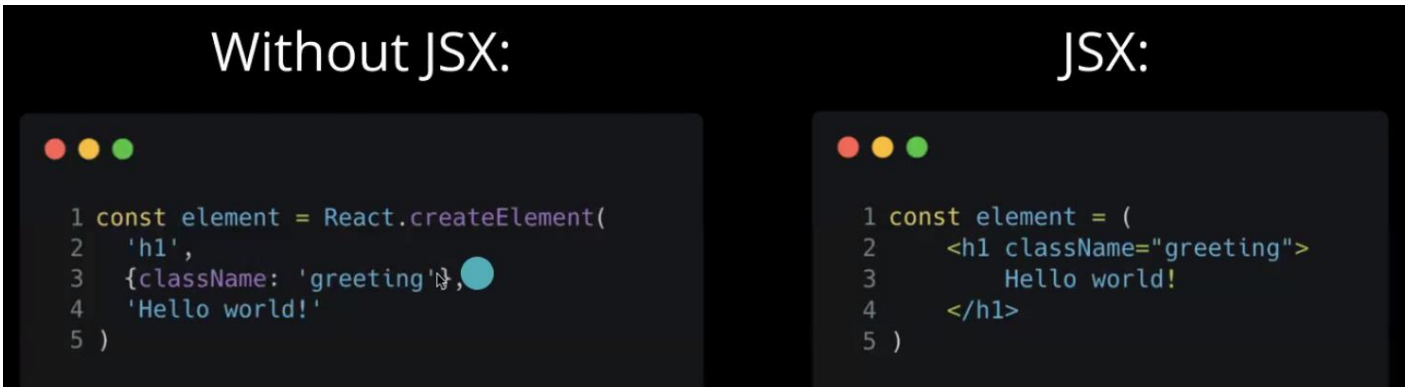
JSX

A combination of JS and HTML.
Stands for JavaScriptXML
Declarative API for writing components

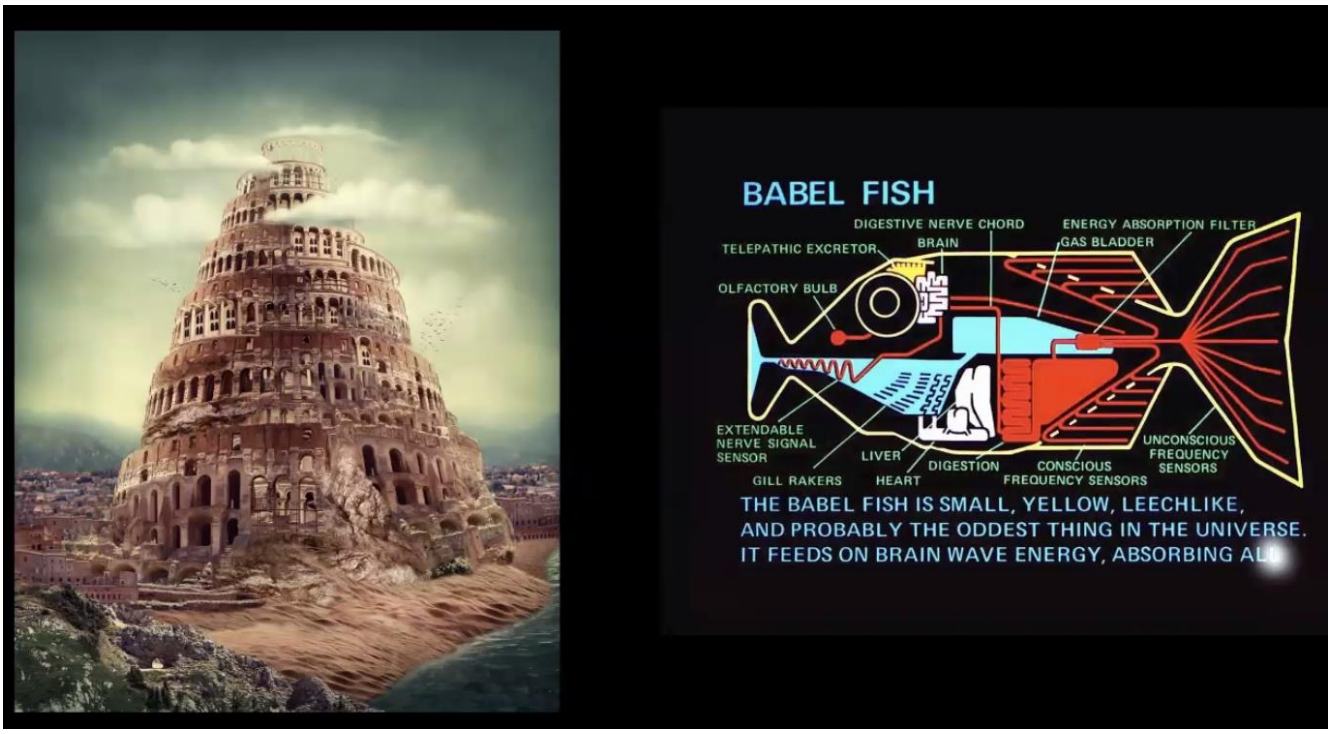
Yesterday's syntax on the left – showing what is happening under the hood.
On the right is how we do it with JSX – much more syntactically similar to HTML.



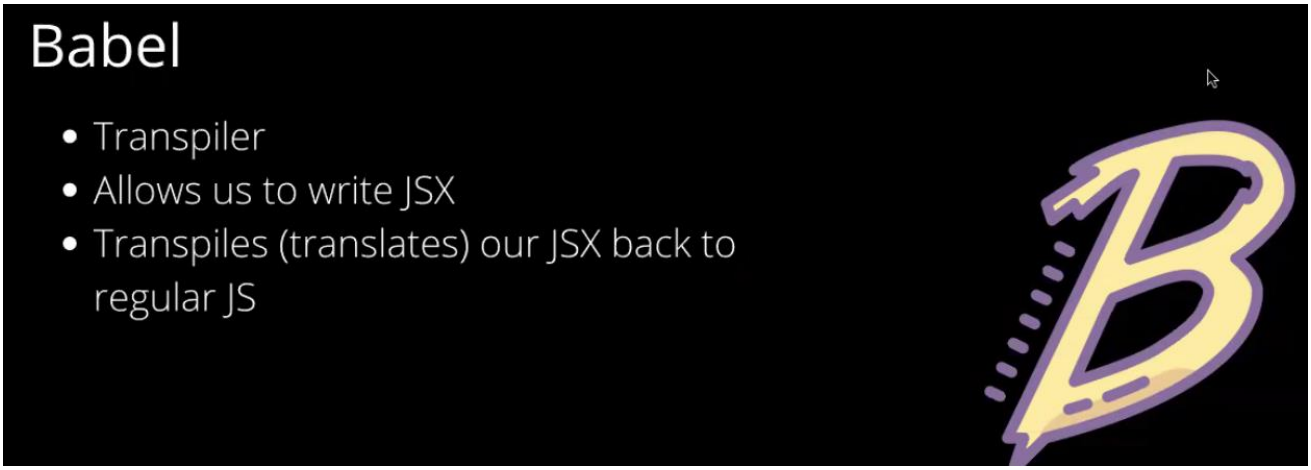
Babel

Liz's synopsis of Tower of Babel in the bible – wanted to make a tower to reach God, God didn't want that so created different languages so people couldn't work together. Everyone spoke same language prior to this.

Babel fish from Hitchhiker's – fish goes in your ear so you can understand any language.



React uses Babel to allow us to use JSX within JS



Liz showed us an example using babeljs.io of how it works:

New-gen JS to old school JS

JSX to JS

<u>Using JSX</u>

Need to add babel-standalone to Task 3 (used in browsers and other non-Node JS environments). We will use a different version for full scale React like Nick showed us in Guest talk this morning and when using with Node later.

We do this by adding line 14 below and changing line 19 to include type:

```
3-helloWorld.html ×
react-intro > 3-helloWorld.html > html > body > script
9        ></script>
10       <script
11         src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
12         crossorigin
13       ></script>
14       <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
15     </head>
16     <body>
17       <div id="root"></div>
18
19       <script type="text/babel">
20         const rootElement = document.querySelector("#root");
21
22         function Heading(text) {
23           return React.createElement("h1", { className: "heading" }, text);
24         }
25
26         ReactDOM.render(Heading("Hello everyone!"), rootElement);
27       </script>
28     </body>
29   </html>
30
```

Liz then showed how to refactor Code for Task 2 from Yesterday to JSX syntax:

```
function Button(props) {
        return <button className={props.className}          Use { } to insert JS in to JSX
        onClick={props.onClick}>{props.text}</button>;      Access props object to get value
}
```

Screenshot containing previous code and yesterday's code commented:

3-helloWorld.html | task2.html M

workshop_React-101-intro-to-react > task2.html > html > body > script > Button

```
23          alert("You clicked me!");
24        }
25
26        function Button(props) {
27          // return React.createElement(
28          //   "button",
29          //   { className: props.className, onClick: props.onClick },
30          //   props.text
31          // );
32
33          //PLAN
34          // Render button on page with JSX ✅
35          // Hook up onClick ✅
36          // Use props to make it customizable
37          return (
38            <button className={props.className} onClick={props.onClick}>
39              {props.text}
40            </button>
41          );
42        }
43
44        const buttonProps = {
45          className: "button",
46          onClick: handleClick,
47          text: "Clickyyyy",
48        };
49
50        ReactDOM.render(Button(buttonProps), rootElement);
51      </script>
```

Can only return 1 thing, so can nest children inside when building the component:

```
// use props to make it customizable
return (
  <p className={props.className} onClick={props.onClick}>
    {props.text}
    <span>blahblah</span>
  </p>
);      You, 4 seconds ago • Uncommitted changes
}
```

Converting the function call in ReactDOM.render to JSX from JS:

```
ReactDOM.render(<Button text={buttonProps.text} onClick={buttonProps.onClick}
className={buttonProps.className}/>, rootElement)
```

The words before the = define the key, the words in {} are accessing the buttonProps object defined above the render, and are passed in as a value. So the "props" being passed in to the Button function call is an object with these key:value pairs.

Proper code with console.log(props) so we can see in the console where these mysterious "props" are!

```
oWorld.html    task2.html M ×    task1.html
p_React-101-intro-to-react > task2.html > html > body > script
      alert("You clicked me!");
    }

    function Button(props) {
      console.log(props);
      return (
        <button className={props.className} onClick={props.onClick}>
          {props.text}
        </button>
      );
    }

    const buttonProps = {
      className: "super-button",
      onClick: handleClick,
      text: "SUPER CLICK",
    };

    ReactDOM.render(
      <Button
        text={buttonProps.text}
        onClick={buttonProps.onClick}
        className={buttonProps.className}
      />,
      rootElement        You, 9 seconds ago • Uncommitted changes
    );
  </script>
</body>
/Users/lizkaufman/Projects/SoC/BC12/03_demos/week-07/workshop    Indents: 4
```

Adding some made up properties called “demo” and “isAProp” (as you can define them inline if preferred, don’t need a prepared object (but this is common practice)):

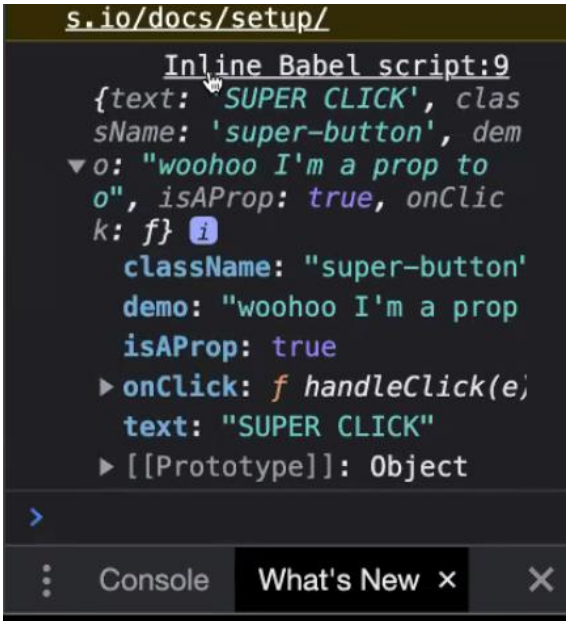
3-helloWorld.html | task2.html M × | task1.html

workshop_React-101-intro-to-react > task2.html > html > body > script

```
23      alert("You clicked me!");
24    }
25
26    function Button(props) {
27      console.log(props);
28      return (
29        <button className={props.className} onClick={props.onClick}>
30          {props.text}
31        </button>
32      );
33    }
34
35    const buttonProps = {
36      className: "super-button",
37      onClick: handleClick,
38      text: "SUPER CLICK",
39    };
40
41    ReactDOM.render(
42      <Button
43        text={buttonProps.text}
44        onClick={buttonProps.onClick}
45        className={buttonProps.className}
46        demo="woohoo I'm a prop too"
47        isAProp={true}        You, 1 second ago • Uncommitted changes
48      />,
49      rootElement
50    );
```

Live Share /Users/lizkaufman/Projects/SoC/BC12/03_demos/week-07/workshop Indents: 5

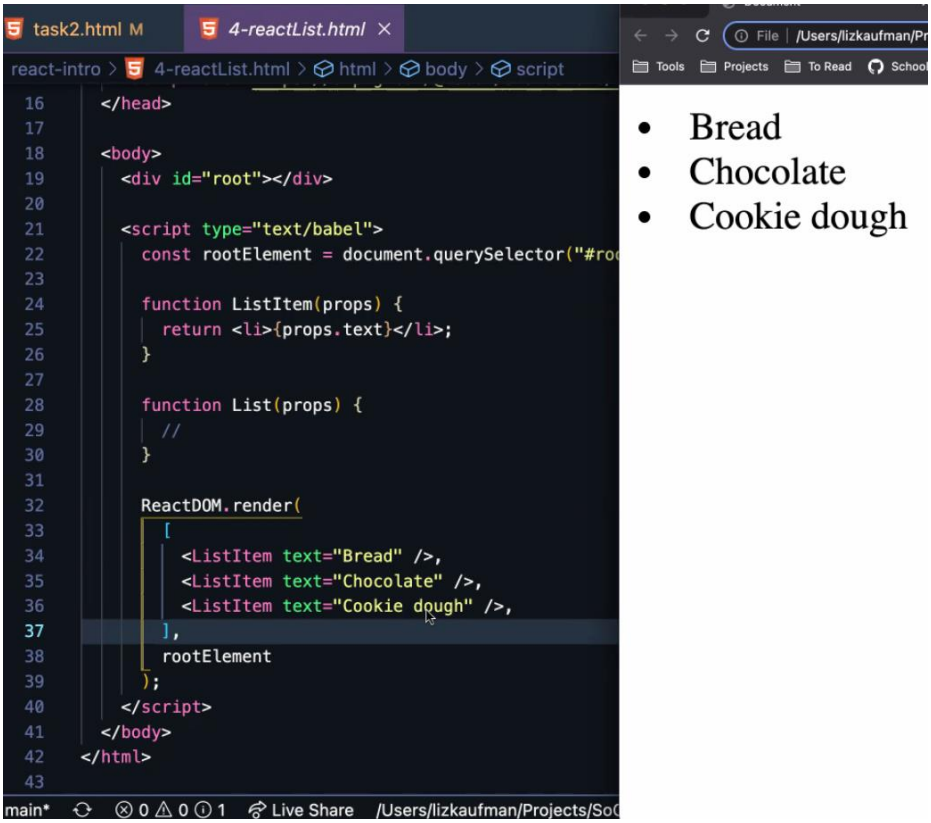
Console log:



Nesting Children in React

Code to create some list items (li). At the moment, these are not nested inside a unordered list (ul) as they are appended directly to the root Element. We are going to complete function on line 28.

N.B. The calls on lines 34-36 are a quick way of calling each item in a self-closing way, instead of <ListItem text="bread"></ListItem>. However, this means they cannot have any children by default.

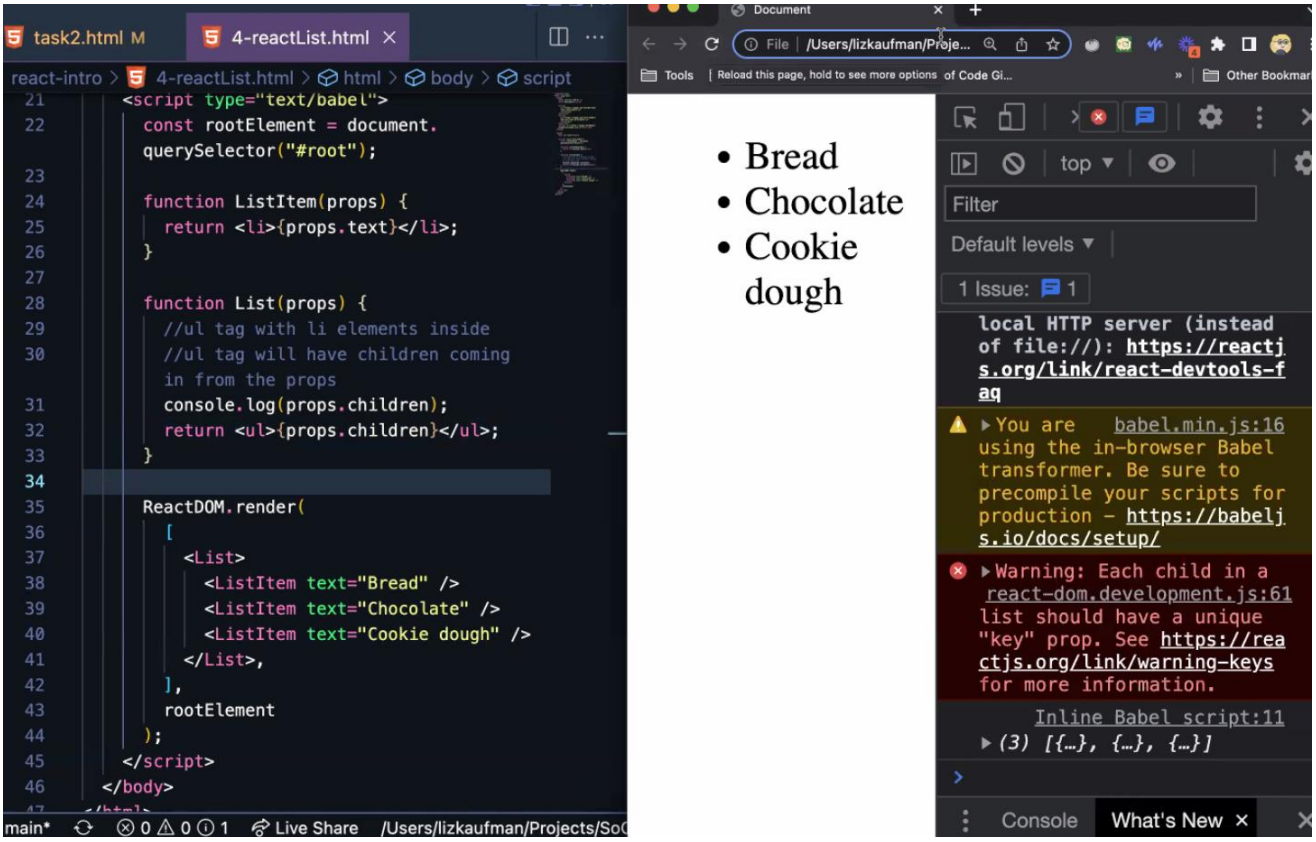


Use keyword of "children" (reserved word) and call inside of the function List, and refactor ReactDOM.render to

Don't need the () after return if on single line, but good practice to have it when the expression spans multiple lines

```
function List(props) {
        return <ul>{props.children}</ul>
};

ReactDOM.render(
        [<List>
                <ListItem text="Bread" />
                <ListItem text="Chocolate" />
                <ListItem text="Cookie dough" />
         </List>
        ],
        rootElement
)
```



Things that are controlled by Boolean

Code for a button that can't be clicked:

5-reactButton.html

react-intro > 5-reactButton.html > html > body > script

```
17
18    <body>
19      <div id="root"></div>
20
21      <script type="text/babel">
22        const rootElement = document.
          querySelector("#root");
23
24        function Button(props) {
25          return (
26            <button onClick={props.onClick}
              disabled={props.disabled}>
27              {props.text}
28            </button>
29          );
30        }
31
32        ReactDOM.render(
33          [
34            <Button
35              text="You can't click me!"
36              onClick={() => console.log("You
                clicked me!")}
37              disabled={true}
38            />,
39          ],
40          rootElement
41        );
```

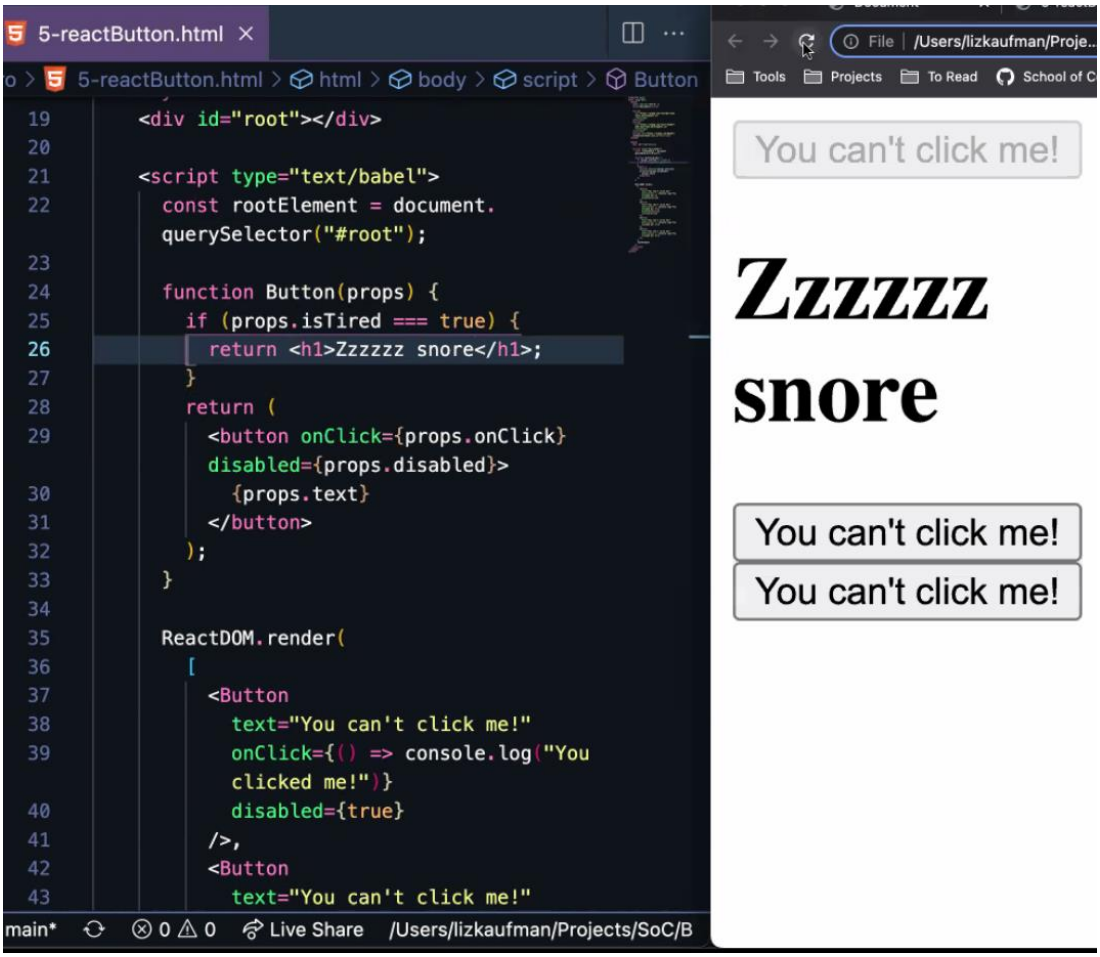
main* ⊗ 0 ⚠ 0 Live Share /Users/lizkaufman/Projects/SoC/B

File | /Users/lizkaufman/Proje...

Tools Projects To Read School of C

You can't click me!

If disabled on line 37 is false, it would be clickable.
Liz added some buttons to show reusability and changed the disabled attribute so some are clickable.
Then added an if check to decide what to render – if isTired === true then render a h1 with Zzzzz, if not, render a button (which isn't clear but there were 4 buttons on the page previously and now there is 3!)



This is rendering conditionally, based on a Boolean. It's passed in as {true} because true as a Boolean value is a piece of JS and we put all JS inside curly braces when inserting it in to JSX.

Didn't do task 7 today as we'll be doing CSS again soon, but can look at it in own time.

Introducing Create React App

This is how to use React "properly", without those dodgy script tags in the head and stuff.

This is like Express Generator, but for React. It spins it up for us without having to install loads of additional libraries like webpack etc.

https://create-react-app.dev/docs/getting-started/

We're going to use it with npx command – npx create-react-app *app-name*

Creates a new folder called app-name and sets it up for react. If you wanted an existing folder, you could put a full stop: npx create-react-app .

npm start runs the development server (like npm run dev when we were doing express)

We had a look through the basic folder structure it gives us – all fairly standard stuff, but the main new thing was the src folder.

In index.js it is creating us an app! But don't worry about it, it's the "trunk of our tree". We are concerned with the branches…which are the other files in this src.

App.js provides an export that is used in index.js, this is like the roots of our tree. This is where the magic happens!

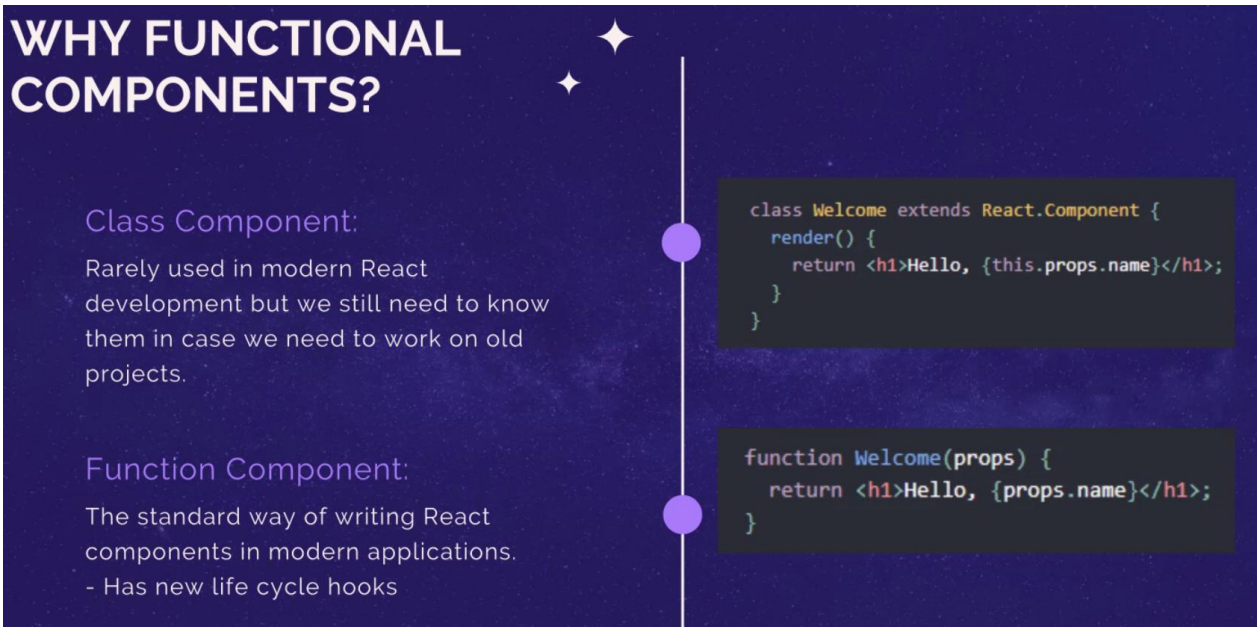
Do npm start and see what happens! It runs a little development server for you and opens a browser tab to localhost:3000 with a placeholder logo and text!

Server automatically reloads like nodemon, so changes are instant when you save the app.js file.

Functional Components

Class component is old way of doing things – React used to rely on classes. You needed to see this as old examples on StackOverflow will still have this syntax etc. Not completely deprecated so still used sometimes too!

Now it is functional and uses hooks.



Remember when we broke down a site into its various elements (like Amazon or Netflix)? This is still part of the process when working with React.

Good page in React docs called Thinking in React: https://reactjs.org/docs/thinking-in-react.html

We are now going to create the above list in this React app!

First, some housekeeping to sort it out to adhere to conventions:

- Create a components folder and inside, a sub-folder called App (Capital A because in JSX we capitalise first letter)
- Then move all 3 App files (.css, .js and .test.js) from root in to this folder (Console will freak out if it is already running (or if you run it now)!)
- Then renamed App.js to index.js as this is convention and also helps when importing, as don't need to point to that specific file, just the folder. It's ok to have loads of index.js all the same name, convention is that we use the folder path to differentiate, not file names.
- Then fixed the import in index.js in root folder by changing import App from './app' to import App from './components/app' (don't need to specify index.js as it will use it automatically!)
- Every single component should have its own folder

Create a List Item component to be used for all items in list:

- Create a ListItem folder (inside components) and an index.js file inside it
- Create function called ListItem and use similar syntax to what we've been doing so far:

```
function ListItem(props){
	return <li>{props.text}</li>
}
export default ListItem
```

index.js .../App U | index.js .../ListItem U | in

bc12-cra-demo > src > components > ListItem > index.js

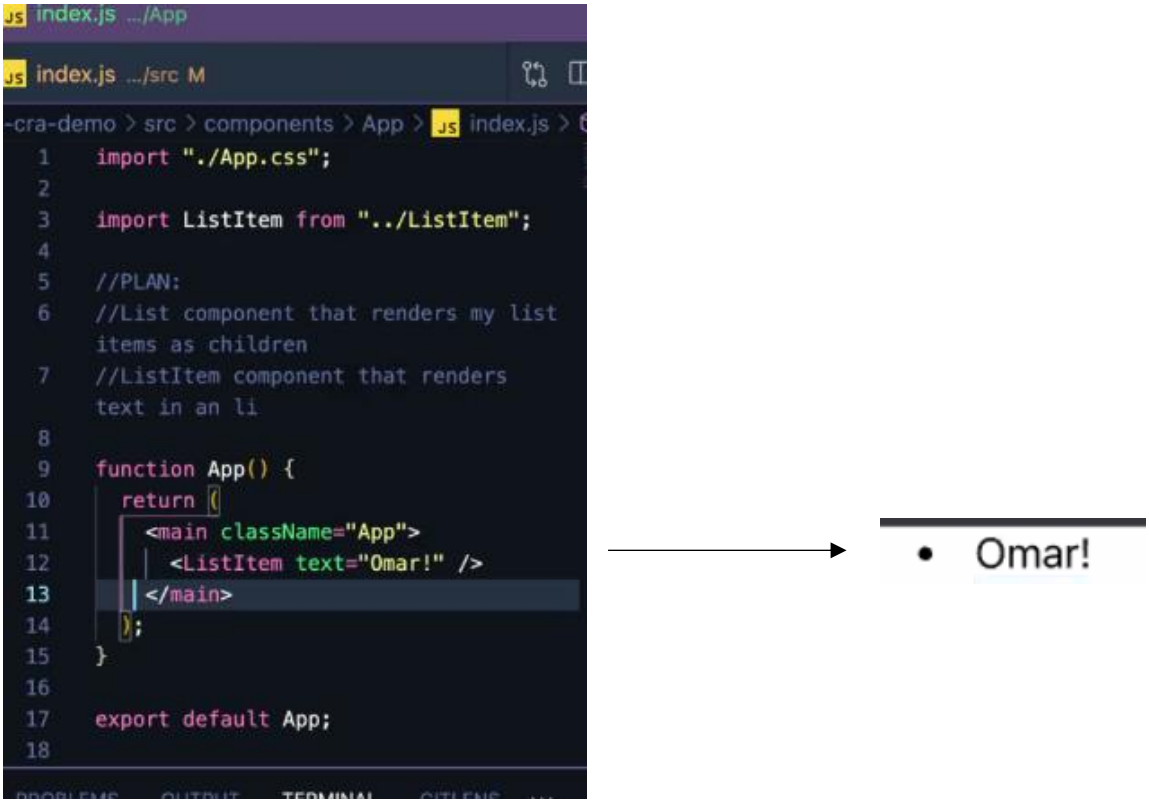
```
1   //PLAN
2   //Function for our ListItem component ✅
3   //Return a <li></li> ✅
4   //Take in props in functional component ✅
5   //Use specific key of props object (text) ins
6
7   function ListItem(props) {
8     return <li>{props.text}</li>;
9   }
10
11  export default ListItem;
12
```

- Import ListItem function in to the index.js inside App folder so we can use it:

import ListItem from '../ListItem'

- Nest it inside the <main> in the function App() (Which represents the <main> of a normal HTML – inside here should look like html):

```
function App() {
        Return (
                <main className="app">
                        <ListItem text="Omar!" />
                </main>
        );
}
```



Then repeat above steps to create the unordered list:

- Create a List folder (inside components) and an index.js file inside it
- Create List function:

```
function List(props){
        return <ul>{props.children}</ul>
}
export default List
```

```
index.js .../App U    index.js .../List U ●    inde
bc12-cra-demo > src > components > List > index.js >
1   // PLAN
2   // Function for List ✅
3   // Take in props✅
4   // return a ul ✅
5   // Render children prop in that ul✅
6   // Export List to use elsewhere✅
7
8   function List(props) {
9     return <ul>{props.children}</ul>;
10  }
11
12  export default List;
13
```

Import it in index.js in App folder and nest List inside <main>, with ListItem nested inside List:

```
function App() {
        Return (
                <main className="app">
                        <List>
                                <ListItem text="React will get easier" />
                                <ListItem text="I promise!" />
                                <ListItem text="Keep practicing" />
                        </List>
                </main>
        );
}
```

index.js .../App U × | index.js .../List U | index.js .../src M

bc12-cra-demo > src > components > App > index.js > App

```
import "./App.css";

import List from "../List";
import ListItem from "../ListItem";

//PLAN:
//List component that renders my list items as children
//ListItem component that renders text in an li

function App() {
  return (
    <main className="App">
      <List>
        <ListItem text="React will get easier" />
        <ListItem text="I promise!" />
        <ListItem text="Keep practicing" />
      </List>
    </main>
  );
}

export default App;
```

- React will get easier
- I promise!
- Keep practicing

Remember

Our "app" is the overall parent and is what is being rendered, so we need to put all other components inside of app.