React Components and props

Prior Knowledge

- Understand how JSX is used in React.
- Understand the basic structure of a `vite` project.

Learning Objectives

- Be able to write functional components.
- Be able to pass information to components via props.
- Be able to access the props of a functional component.
- Understand how to extract functionality into its own component.

React Components

Components allow us to split our apps into independent and reusable units. A major benefit of separating our code into separate components is that we can then begin to reason about each component in isolation.

React components can be one of two types. Either a `function` or a `class`.

Throughout this course we will be dealing with and writing `functional` components. Before the introduction of `React hooks` class based components were the only way to manage state within React. This is no longer the case and we can deal with the simpler and more succinct functional components in all cases. They have their own section in these notes if you're interested in how they work.

Naming conventions

All of our components will start with a capital letter. This is to differentiate them from html tags as they use the same syntax when they're rendered.

By convention our first component will be called `App`. This is the component that will be rendered by `main.jsx` and all of our other components will be children of this component.

Rendering components

Components are rendered using `<>` brackets in the same way as html elements. For example

```
import ReactDOM from 'react-dom/client';

const App = () => {
  return (
    <div>
      <h1>My First React App</h1>
    </div>
  );
};


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Note the self closing tags here. We also have the option of using a second tag but the first is more concise and is the way you will see the most. It is possible to use an opening and closing tag but as we have no need to pass children to the App this is a topic for another day. (See the composition notes if you're interested)

```
<App /> // renders App
<App></App> // also renders App
```

Our components form a parent child relationship. Each of our components can render child components, just as they would render child html elements.

An example App structure could look something like below:

```
App
├── Header
├── Nav
├── Content
    ├── WelcomePage
    ├── Sidebar
```

```
const App = () => {
  return (
    <div>
      <Header />
      <Nav />
      <Content />
    </div>
  );
};

const Header = () => {
  return <h1>App Title</h1>;
};

const Nav = () => {
  return <nav></nav>;
};

const Content = () => {
  return (
    <section>
      <WelcomePage />
      <Sidebar />
    </section>
  );
};

const WelcomePage = () => {
  return (
    <main>
      <h2>Welcome to the App</h2>
    </main>
  );
};

const Sidebar = () => {
  return (
    <section>
      <p>Some Side Content</p>
    </section>
  );
};
```

As we increase the complexity of our apps we can work on isolated components and deal with one piece of functionality at a time.

Props

We will frequently need to pass information between components. React provides a mechanism for this called `props`, short for `properties`.

The syntax for passing props is similar to passing html attributes. Props are key-value pairs. We name the key of the prop on the left and pass the value on the right of the `=` as shown below.

```
const App = () => {
  return (
    <div>
      <Sum num1={2} num2={3} />
    </div>
  );
};

const Sum = (props) => {
  console.log(props, '<< Sum props'); // { num1: 2, num2: 3 }
  return (
    <p>
      {props.num1} + {props.num2} = {props.num1 + props.num2}
    </p>
  );
};
```

All of our components are passed a single argument, a `props` object. The object contains all of the key-value pairs that have been passed from the parent component.

We can pass any values that we like from the parent to a child component. However we cannot pass props directly from a child to it's parent nor can we pass props from a component to a sibling.

See Passing Props to a Component for more info.