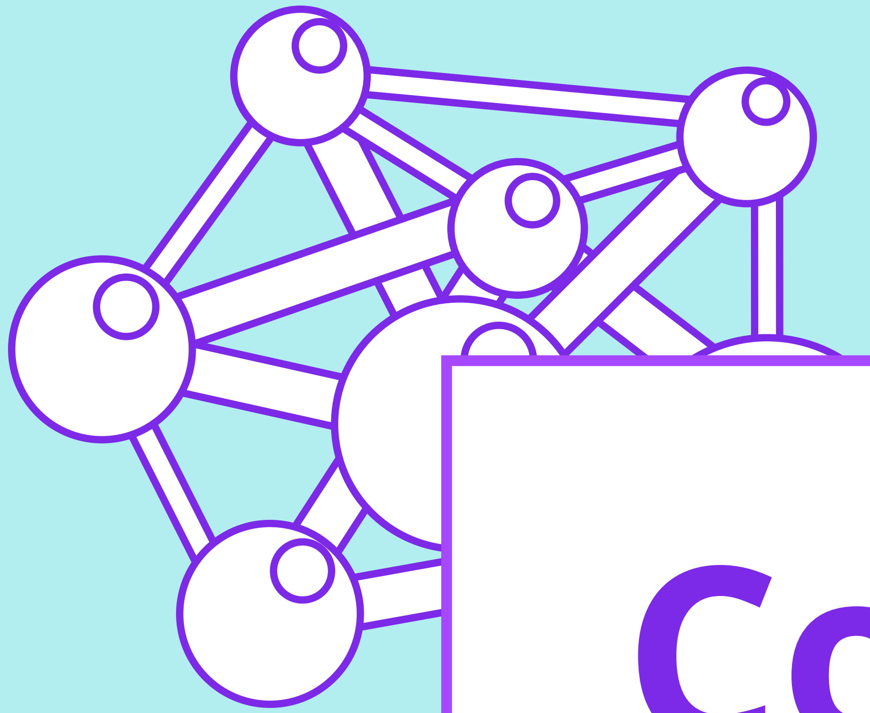
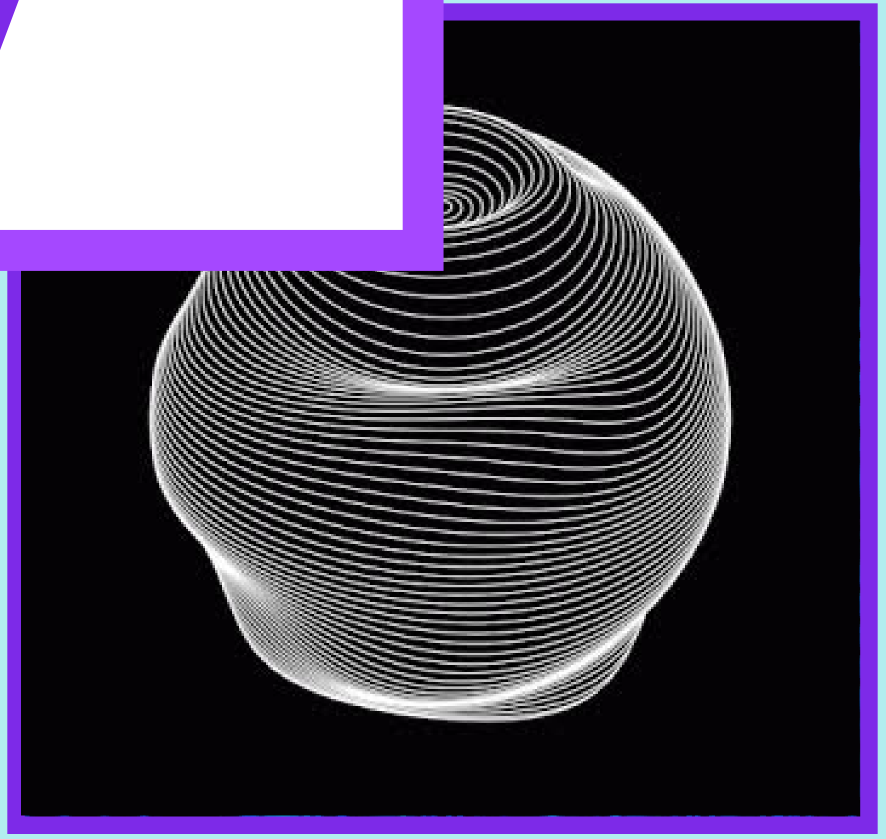


Components & Reusability

Written by **Fahad Rehman**, presented by **Hayden Smith**



Why Reusability?

"Don't reinvent the wheel, just realign it"

- Efficient to implement new features
- Consistent
- Easier to test
- Easier to debug

Component Driven Development

"Don't reinvent the wheel, just realign it"

Components are a set of web platform APIs that allow you to create new custom, reusable, encapsulated HTML tags to use in web pages and web apps. Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Component-Driven Development is a development methodology that anchors the build process around components.

It is a process that builds UIs from the “bottom up” by starting at the level of components and ending at the level of pages or screens.

React Components

In React, we can build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript/JSX instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Basic React Component

```
function Button () {
   return <button>Hello</button>;
}
```

Hello

```
<Button />
```

Hello

```
<Button />
```

"Don't reinvent the wheel, *just realign it*"

Q

That's all good but my components won't always have the same text and UI. How do I control the state and UI of my component?

A

Different UI have different needs.
For example, in our button component, we would have scenarios where the text or the background colour is different. Components would only be useful if somehow we can accomodate all of these different UI and behavioural states. Luckily, React has a mechanism just for this purpose

Introducing Props

</>

React components can be customised when they are created, with different parameters. These created parameters are called props, short for properties.

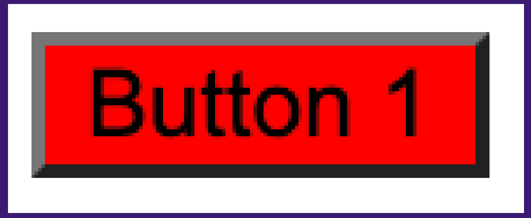
For example, in our basic Button React Component, you can use props to control it's text, background colour as well as click callback

"props" is an object defined by React. This has access to all the properties passed to the component.

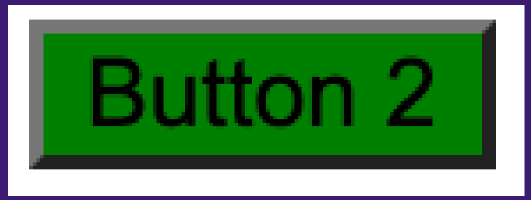
```
function Button (props) {
    const { btnText, btnBackgroundColour, onBtnClick } = props;
    return (
        <button
            onclick={onBtnClick}
            // "backgroundColor" is JSX for "background-color"
            style={{ backgroundColor: btnBackgroundColour }}>
                {btnText}
        </button>
    );
}
```

Props Usage

```
//Clicking on this button will give an alert with "Btn1 click"
<Button
   btnText="Button 1"
   btnBackgroundColour="red"
   onBtnClick={() => alert("Btn1 click"} />
```



```
//Clicking on this button will give an alert with "Btn2 click"
<Button
   btnText="Button 2"
   btnBackgroundColour="green"
   onBtnClick={() => alert("Btn2 click"} />
```



All React components must act like pure functions with respect to their props.

```
//Pure function as it doesn't change it's parameters
function sum(a, b) {
  return a + b;
}


//This function is not pure as it does change it's
parameters
function withdraw(account, amount) {
  account.total -= amount;
}
```

Other types of Components

Categorising a component based on its purpose and behaviour can lead to more cleaner and reusable components. Consider following three types of commonly used components:

- **Dumb/Stateless Components:** A component that doesn't have any state. Or in other words, a component that just display some static content. They are much more efficient because they don't require much resources to render
- **Presentational Components**: These are components that don't have their own state. State gets passed to them and they conditionally render UI based on it. They do not bother with the management of state. Presentational Component mainly concerned with how things look.
- **Container Components**: They provide the data and behavior to presentational or other container components. A container does data fetching and then renders its corresponding sub-component. This component mainly concerned with how things work.

Dumb/Stateless Component

```
function DumbComponent() {
  return <h1>Hello, I am dumb</h1>;
}
```

Presentational Component

```
function PresentationalComponent(props) {
  return <h1>{props.title}</h1>;
}
```

Container Component

```
function ContainerComponent() {
   const title = "Hello";
   return <PresentationalComponent title={title] />;
}
```