Class Based Components

Topics covered:

- What are Class Based components?
- What is a component constructor?
- What are props in class components?
  - Props to the constructors
  - Components in the component
- What is the State of class components?
  - Creating state object
  - Using State object
  - Changing state object

1. Class-Based Components:

- A class is a particular kind of component in React that enables state management.
- Until Hooks were introduced in React 16.8, the only way to use and maintain states was through a class-based component.
- This is the reason why the only stateful component available at the time was a class.
- Classes let us handle and alter the state in an efficient manner.

  A common class-based component contains the following:

  - For developing JSX, a React component was imported from the React library.
  - Classes can be created using this component imported from the React library.
  - A class component that extends the React Component.
  - To initialize the props, we need a constructor.
  - To paint/create the UI, we need a render method.
  - Export with the same name as the class.
  - The class component in this sample is named CodeFeast, and it creates a header called Hello Everyone!
  - Example:

```
import React, { Component } from 'react'
export default class Demo extends Component {
 render() {
   return (
     <div>Demo</div>
   )
 }
}
```

2. Component Constructor:

- When the component is started, the Constructor() method is the first function executed, making it the obvious place to set up the initial state and other basic settings.
- You should always use the super(props) method before any other method to start the parent's constructor procedure and allow the component to inherit methods from its parent.
- The Constructor() method is invoked with the props as arguments (React.Component).

Example:

```
import React from 'react';
export class Color extends React.Component {
 constructor() {
   super();
   this.state = { color: "red" };
 }
 render() {
   return <h2>The color is {this.state.color}</h2>;
 }
}
```

Output:

The color is red

3. Props:

- Props are similar to function arguments and are passed into the component as properties.
- Like in this Example, <Demo color =" red"/> added a color prop to the class and rendered it through this.props.color (this.props.propName)

Example:

```
class Demo extends React.Component {
 render() {
   return <h2>The color is {this.props.color} Car!</h2>;
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Demo color="red"/>);
```

Props in constructors:

- Props should always be supplied to the constructor function of your component if it has one, as well as to the React.Component via the super() function.

Example:

```
class Demo extends React.Component {
 constructor(props) {
    super(props);
}
 render() {
   return <h2>The color is {this.props.color} Car!</h2>;
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Demo color="red"/>);
```

Components in components:

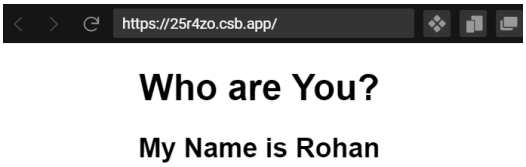
- Components inside other components are referred to as

Example:

```
class Demo extends React.Component {
 render() {
   return <h2>My Name is Rohan</h2>;
 }
}

class Demo1 extends React.Component {
 render() {
   return (
     <div>
     <h1>Who are You?</h1>
     <Demo />
     </div>
   );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Demo1/>);
```

Output:



4. State in class-based component:

- A state object is pre-built into React Class components.
- As you may have seen, the state was used in the component constructor part before.
- Property values for the component's properties are kept in the state object.
- The component re-renders whenever the state object alters.

Create a State object:

In the constructor, the state object is initialized:

```
export class Color extends React.Component {
 constructor(props) {
   super(props);
   this.state = { color: "red" };
 }
 render() {
   return <h2>The color is {this.state.color}</h2>;
 }
}
```

You can have as many attributes as you wish in the state object:

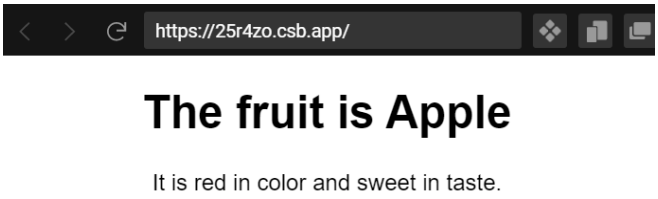
```
import React from "react";
export class Fruit extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
           fruit: "apple",
           color: "red",
           taste: "sweet" };
 }
 render() {
   return <h2>The Fruit</h2>;
 }
}
```

Using State object:

Use the this.state.propertyName syntax to refer to the state object anywhere in the component:

```
import React from "react";
export class Fruit extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
     fruit: "Apple",
     color: "red",
     taste: "sweet"
   };
 }
 render() {
   return (
     <div>
       <h1>The fruit is {this.state.fruit}</h1>
       <p>
         It is {this.state.color} in color and {this.state.taste}
 in taste.
       </p>
     </div>
   );
 }
}
```

Output:



Changing the state object:

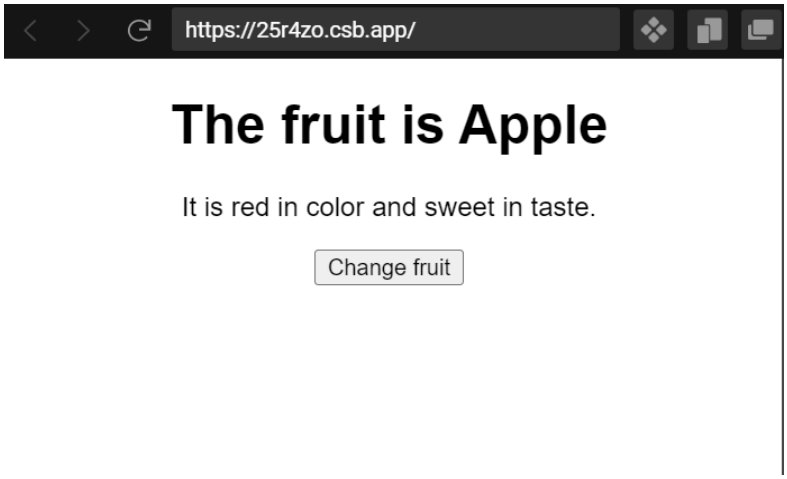
- Use this.setState() to modify a value in the state object.
- The component will re-render when a value in the state object changes, which means that the output will adjust to the new value (s).

Example:

```
import React from "react";
export class Fruit extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
     fruit: "Apple",
     color: "red",
     taste: "sweet"
   };
 }
 changeFruit = () => {
   this.setState({ fruit: "pomegranate", taste: "sour" });
 };
 render() {
   return (
     <div>
       <h1>The fruit is {this.state.fruit}</h1>
       <p>
         It is {this.state.color} in color and {this.state.taste} in
taste.
       </p>
       <button onClick={this.changeFruit}>Change fruit</button>
     </div>
   );
 }
}
```

Output:

Initially before clicking on the Change fruit button.



After clicking on the Change fruit button: