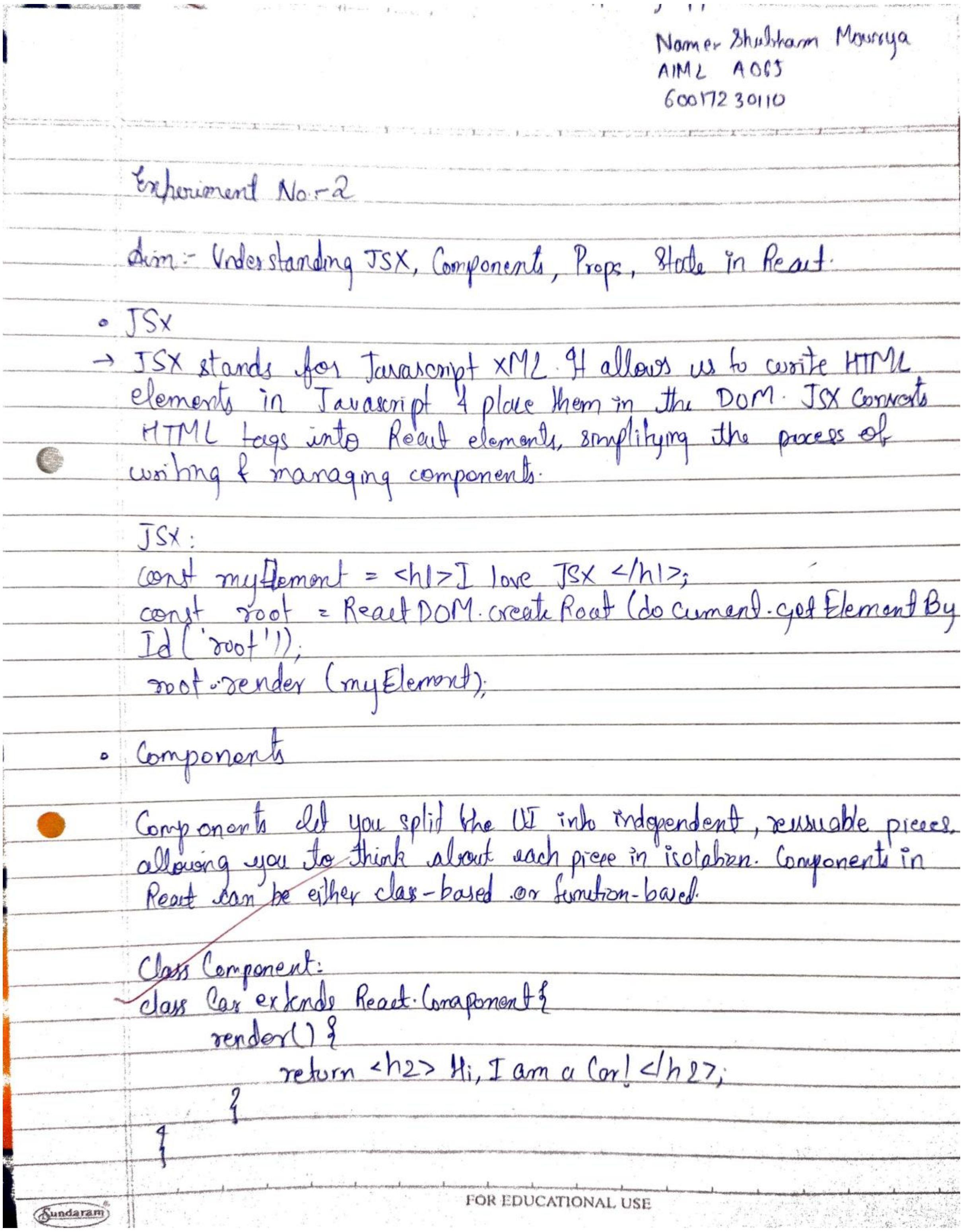
Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Experiment 2

Name: Shubham Mourya **SAP ID: 60017230110**



Name: Shubham Mourya
AIML A065
60017230110

Experiment No - 2

Aim :- Understanding JSX, Components, Props, State in React.

- JSX

→ JSX stands for Javascript XML. It allows us to write HTML elements in Javascript & place them in the DOM. JSX converts HTML tags into React elements, simplifying the process of writing & managing components.

JSX :

```
const myElement = <h1>I love JSX </h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

- Components

Components let you split the UI into independent, reusable pieces, allowing you to think about each piece in isolation. Components in React can be either class-based or function-based.

Class Component:

```
class Car extends React.Component {
      render() {
            return <h2> Hi, I am a Car! </h2>;
      }
}
```

FOR EDUCATIONAL USE

Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Function Component:

```
function Car() {
        return <h2> Hi, I am a Car! </h2>; }
```

* React Prop
Props are arguments passed into React components via HTML attribute. They are read-only & allow components to be dynamic & reusable.

```
function Car(props){
      return <h2> I am a {props.brand} Car! </h2>; }
```

* State
State is a built-in object in React Component used to store property values that belong to the component. When the state object changes, the component re-renders.

'useState' hook allows you to add state to function components

```
import {useState}; from "react";
function FavoriteColor(){
    const [color, setColor] = useState("red");
}
```

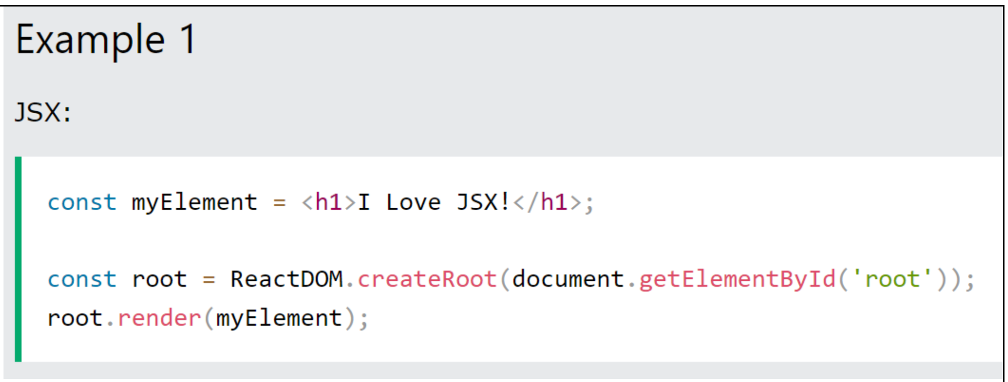
State can hold various data types, including strings, numbers, booleans, arrays & objects.

Conclusion:- this experiment helped us to understand JSX, React Components, props & state management.

[illegible]

Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

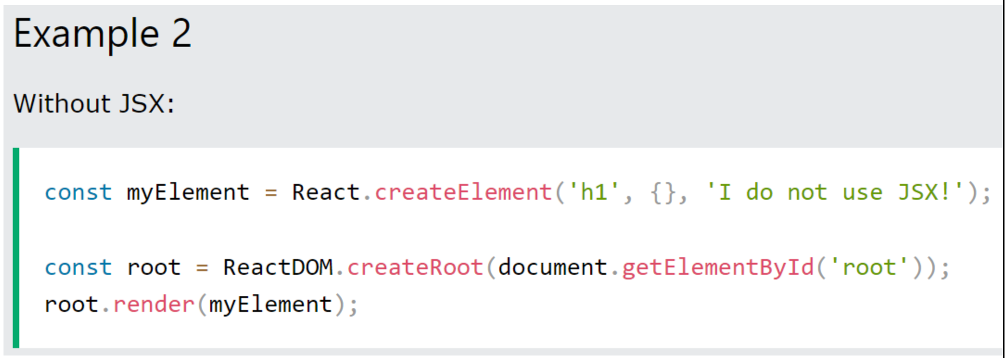
Date:	**6/8/2024**
Aim	**Understanding JSX, Components, Props, State in React.**
Software	
Pre-requisite	Active internet connection
Theory	**Understanding JSX** What is JSX? JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React. Coding JSX JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods. JSX converts HTML tags into react elements. You are not required to use JSX, but JSX makes it easier to write React applications. Here are two examples. The first uses JSX and the second does not:



Example 1

JSX:

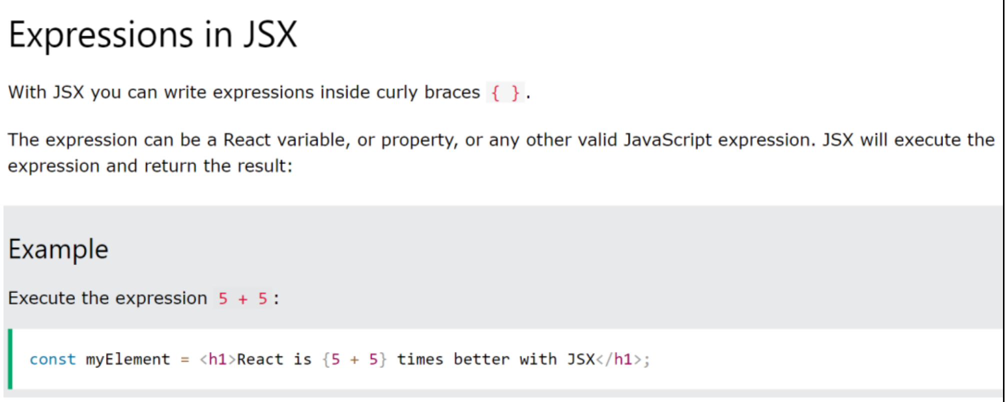
```
const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



Example 2

Without JSX:

```
const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



Expressions in JSX

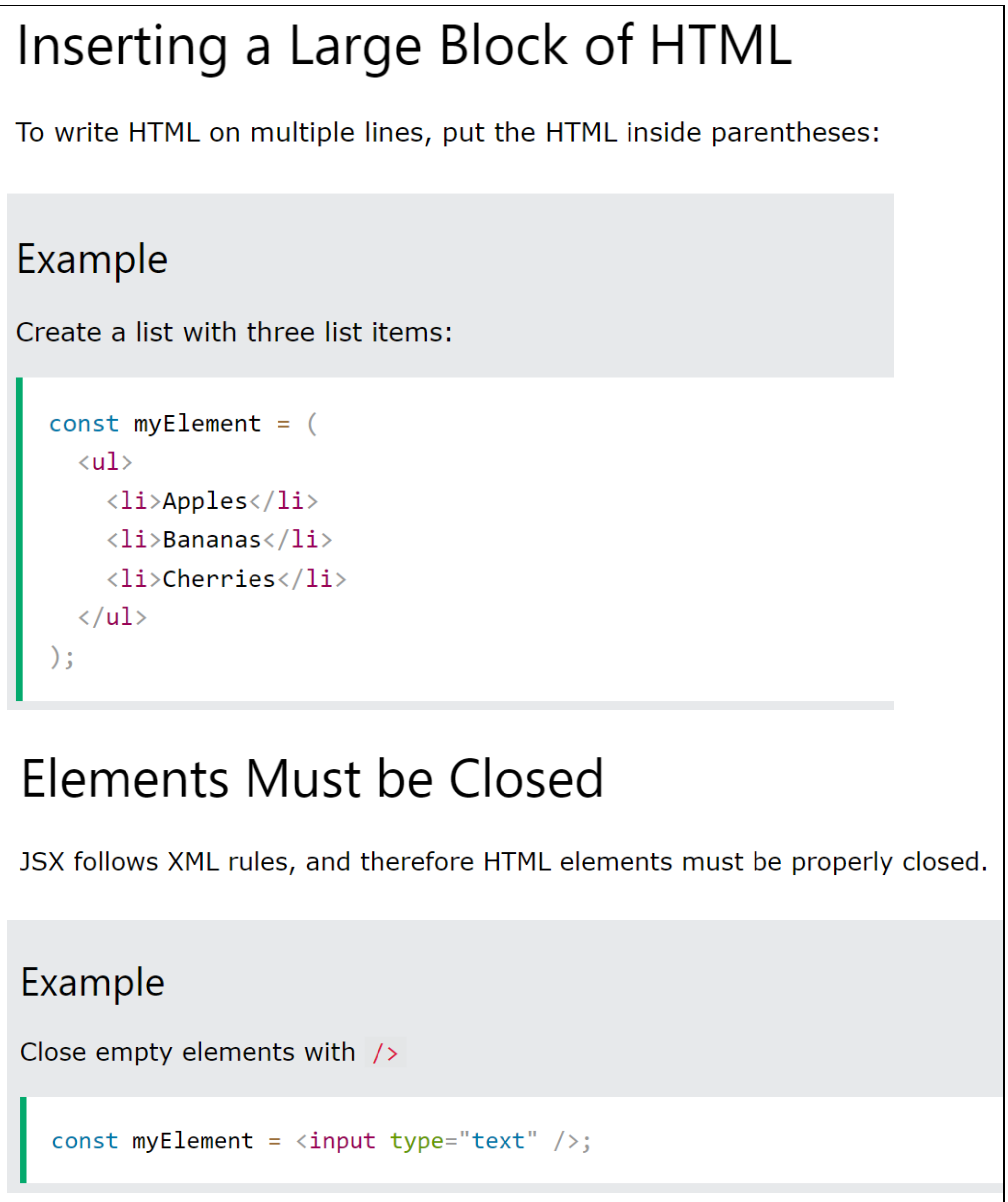
With JSX you can write expressions inside curly braces `{ }`.

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

Example

Execute the expression `5 + 5`:

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```



Inserting a Large Block of HTML

To write HTML on multiple lines, put the HTML inside parentheses:

Example

Create a list with three list items:

```
const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

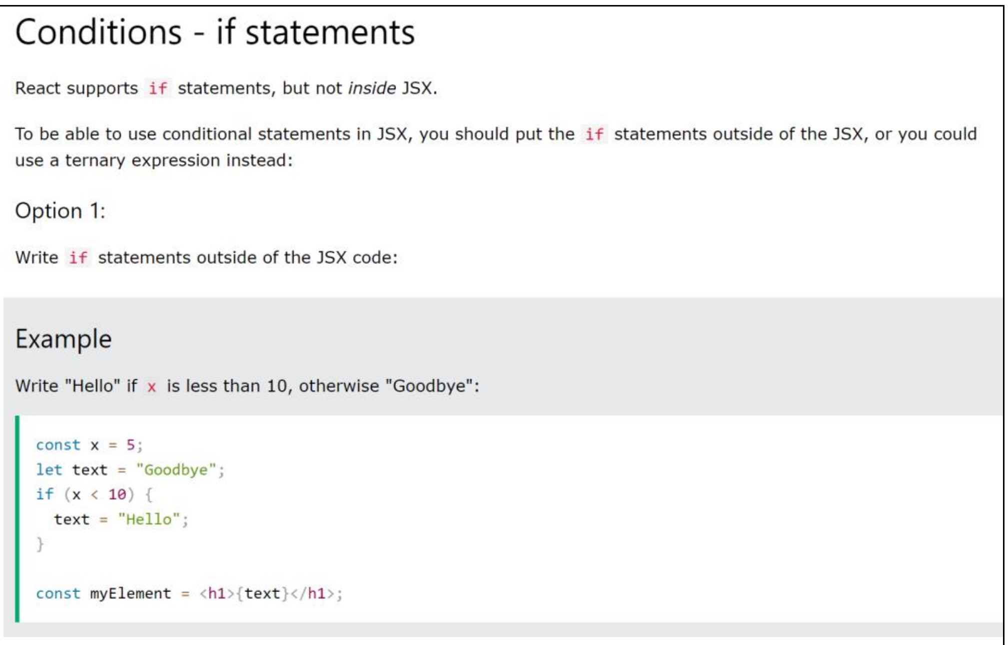
Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Example

Close empty elements with `/>`

```
const myElement = <input type="text" />;
```



Conditions - if statements

React supports `if` statements, but not *inside* JSX.

To be able to use conditional statements in JSX, you should put the `if` statements outside of the JSX, or you could use a ternary expression instead:

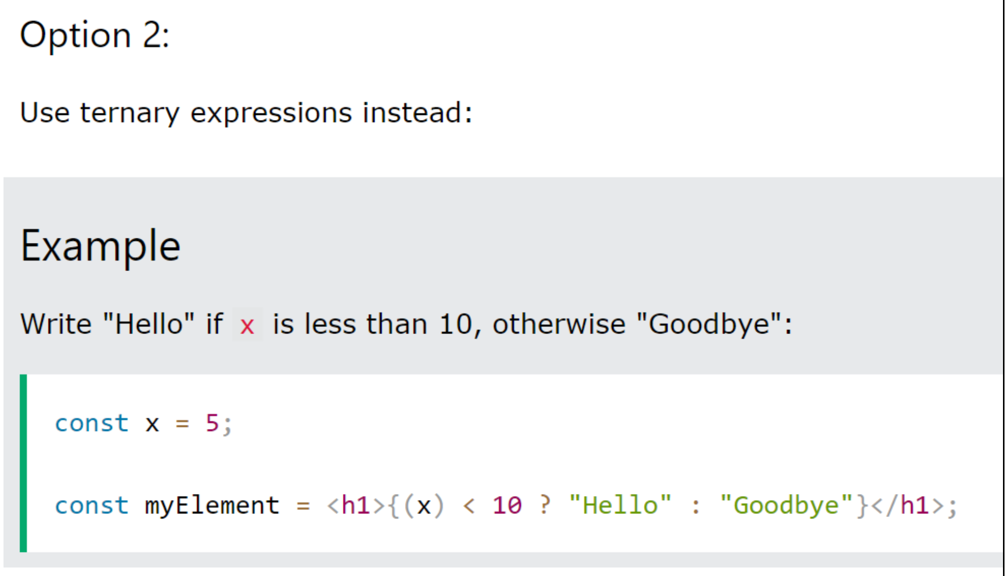
Option 1:

Write `if` statements outside of the JSX code:

Example

Write "Hello" if `x` is less than 10, otherwise "Goodbye":

```
const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myElement = <h1>{text}</h1>;
```



Option 2:

Use ternary expressions instead:

Example

Write "Hello" if `x` is less than 10, otherwise "Goodbye":

```
const x = 5;

const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;
```

Note: that in order to embed a JavaScript expression inside JSX, the JavaScript must be wrapped with curly braces, {}.

Exercise for JSX:

1) Example 1: Texerc

2) Example 2: In this example where conditional expression is embedded in JSX

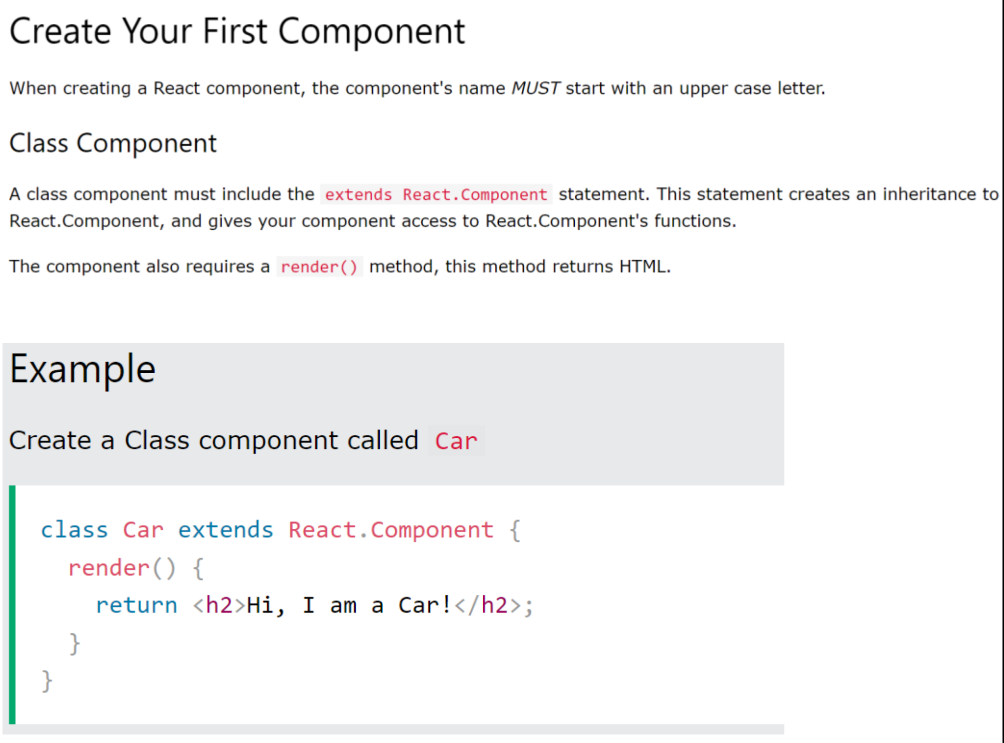
3) Example 3: In this example we have wrapped h1, h2, and h3 tags under a single div element and rendered them to HTML

4) Example 4: Converting HTML to JSX

Components and Props

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components. You can find a detailed component API reference here.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.



Create Your First Component

When creating a React component, the component's name *MUST* start with an upper case letter.

Class Component

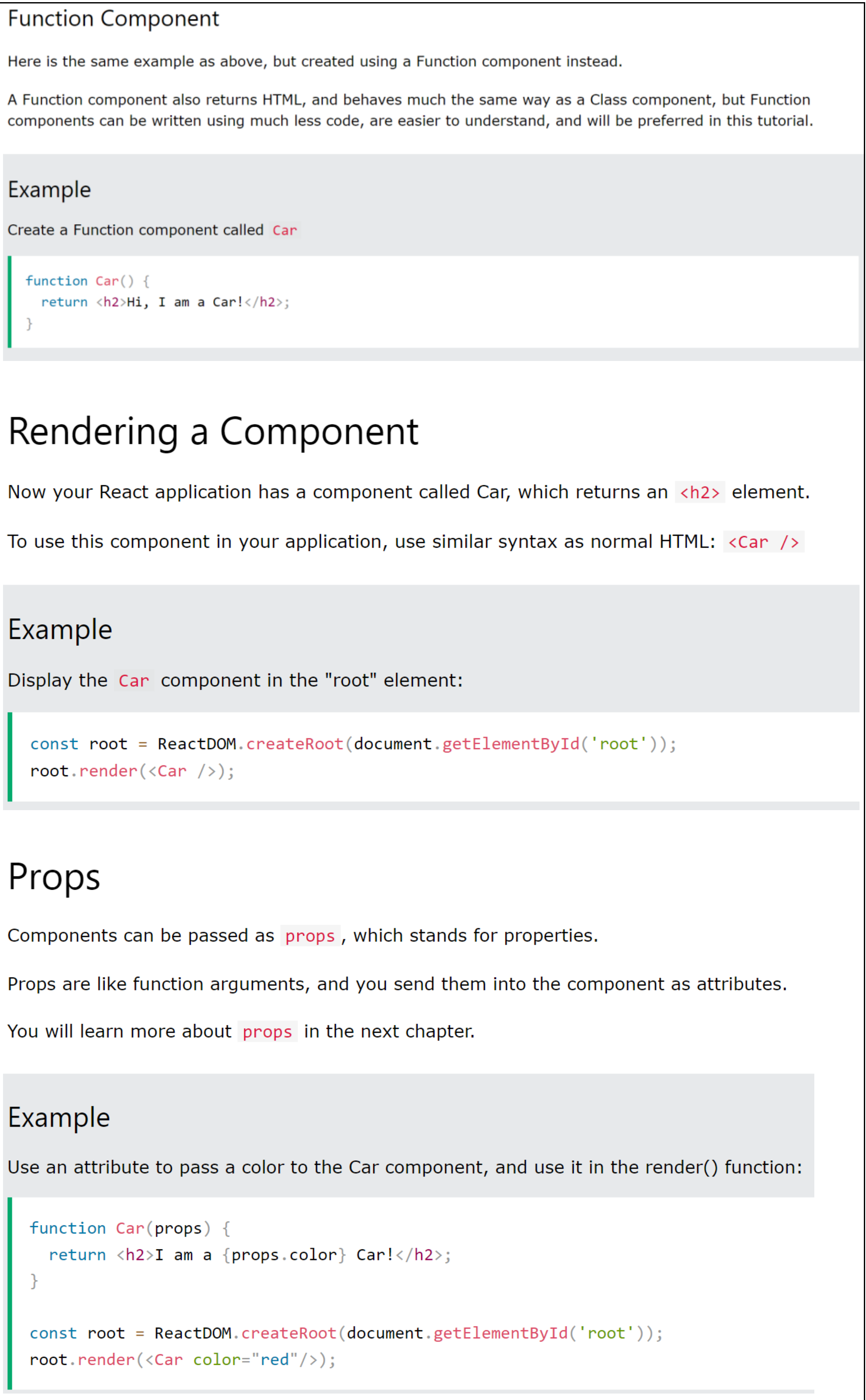
A class component must include the `extends React.Component` statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a `render()` method, this method returns HTML.

Example

Create a Class component called `Car`

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```



Function Component

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

Example

Create a Function component called `Car`

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

Rendering a Component

Now your React application has a component called Car, which returns an `<h2>` element.

To use this component in your application, use similar syntax as normal HTML: `<Car />`

Example

Display the `Car` component in the "root" element:

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

Props

Components can be passed as `props`, which stands for properties.

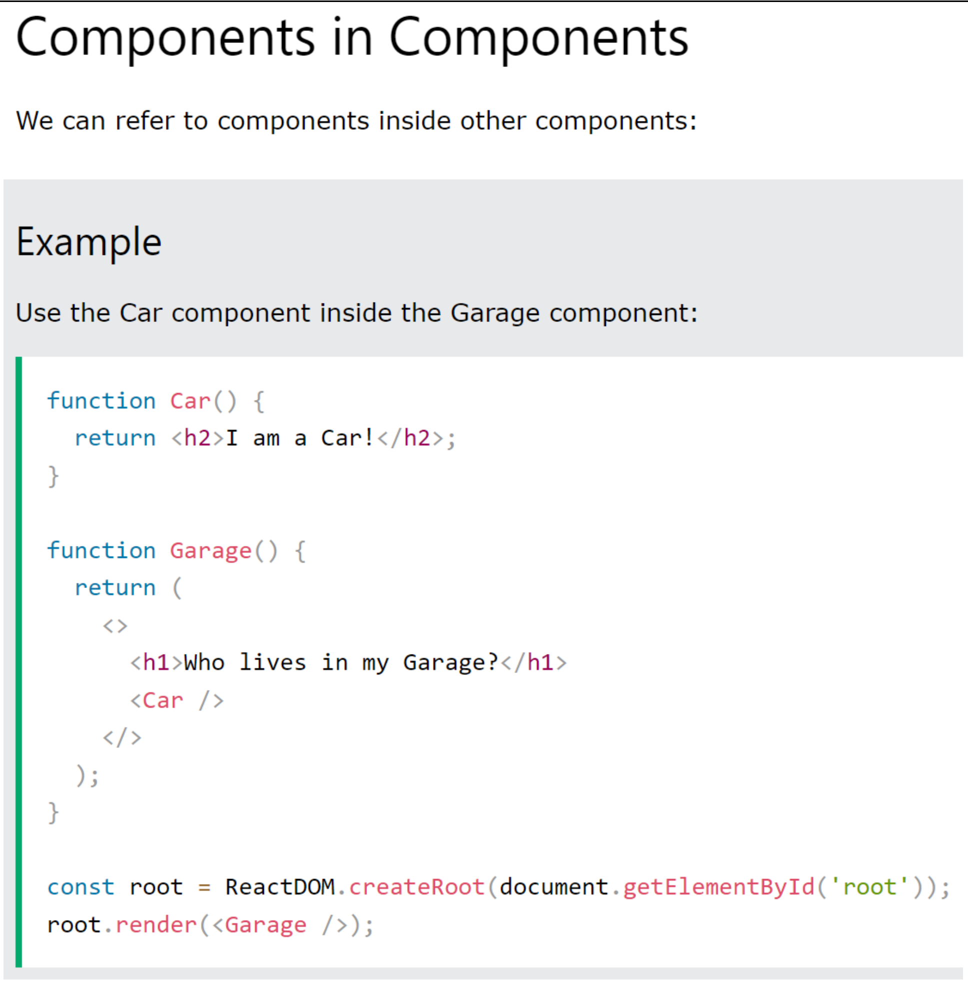
Props are like function arguments, and you send them into the component as attributes.

You will learn more about `props` in the next chapter.

Example

Use an attribute to pass a color to the Car component, and use it in the render() function:

```
function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red"/>);
```



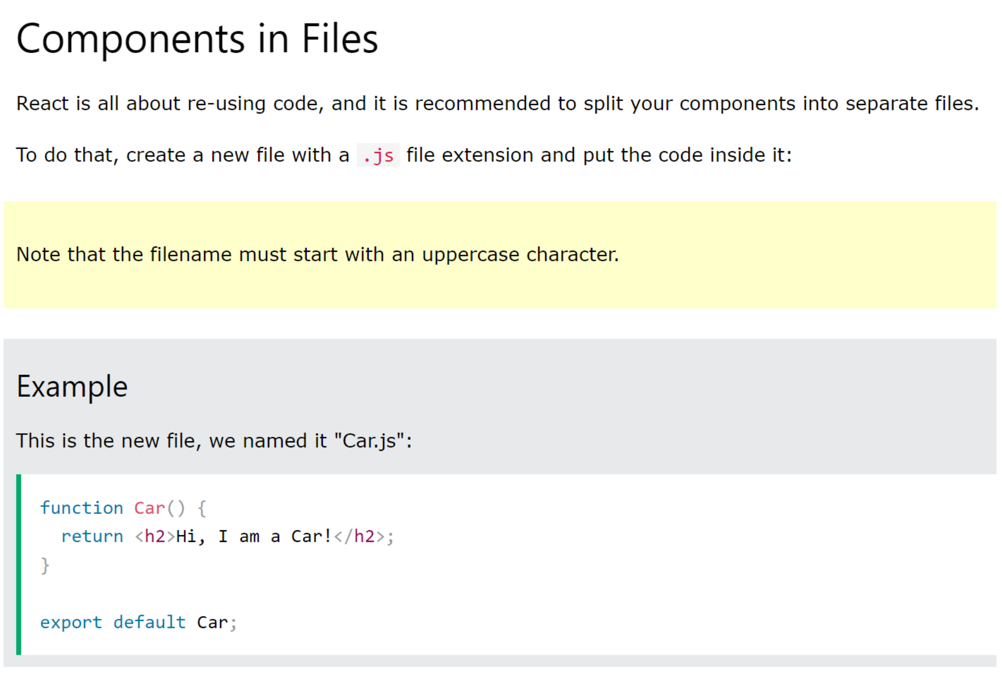
Components in Components

We can refer to components inside other components:

Example

Use the Car component inside the Garage component:

```
function Car() {
  return <h2>I am a Car!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car />
    </>
  );
}

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



Components in Files

React is all about re-using code, and it is recommended to split your components into separate files.

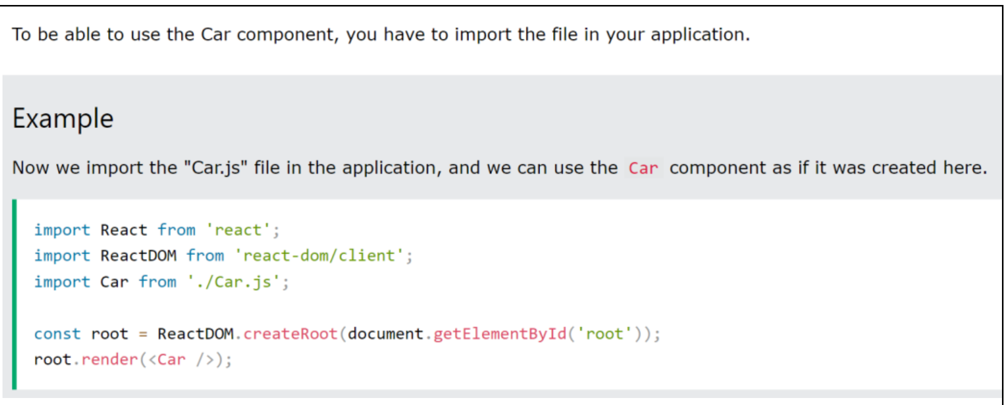
To do that, create a new file with a `.js` file extension and put the code inside it:

Note that the filename must start with an uppercase character.

Example

This is the new file, we named it "Car.js":

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}

export default Car;
```



To be able to use the Car component, you have to import the file in your application.

Example

Now we import the "Car.js" file in the application, and we can use the `Car` component as if it was created here.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Car from './Car.js';

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

Exercise:

1) Example: Create a function component called welcome.

2) Example: Create a class component called welcome.

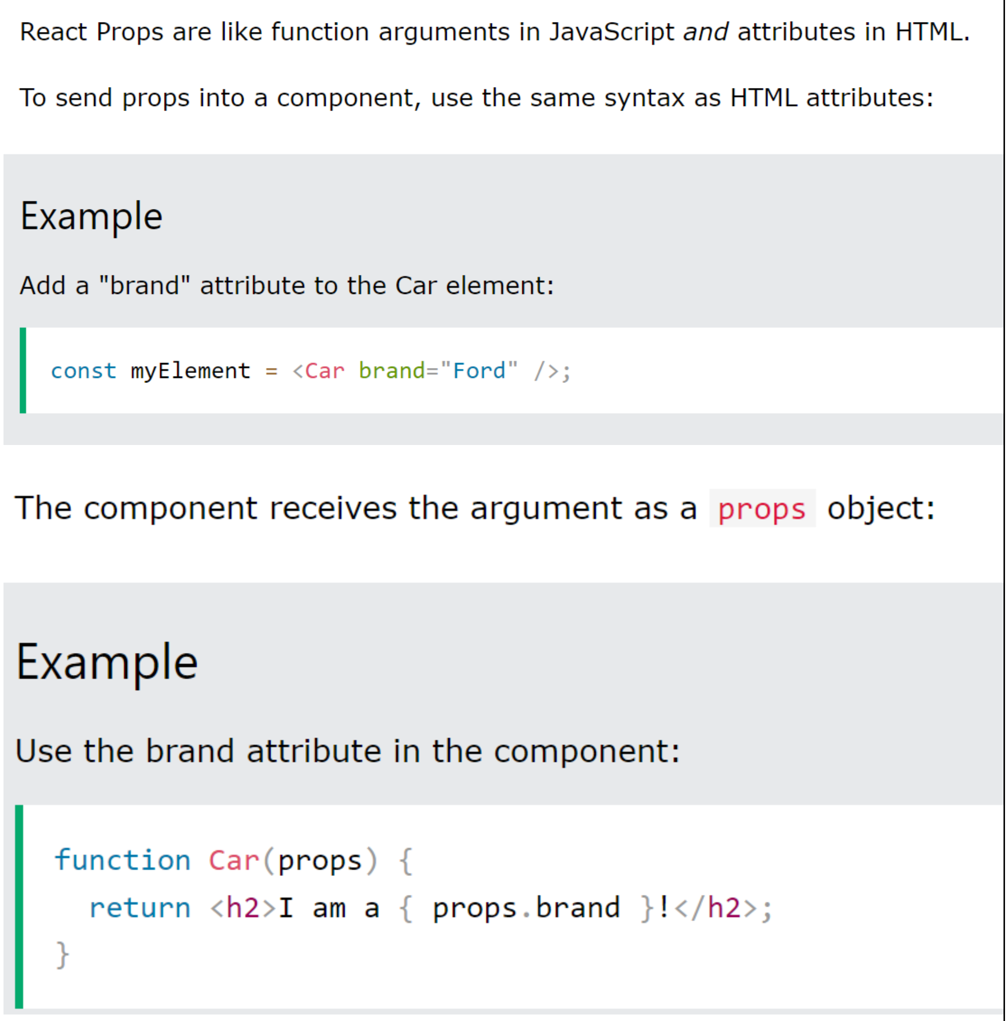
3) Example: Renders a component named Welcome to the Screen.

4) Example: Create the list of unordered elements, where we will dynamically insert StudentName for every object from the data array.

React Props

Props are arguments passed into React components.

Props are passed to components via HTML attributes.



React Props are like function arguments in JavaScript *and* attributes in HTML.

To send props into a component, use the same syntax as HTML attributes:

Example

Add a "brand" attribute to the Car element:

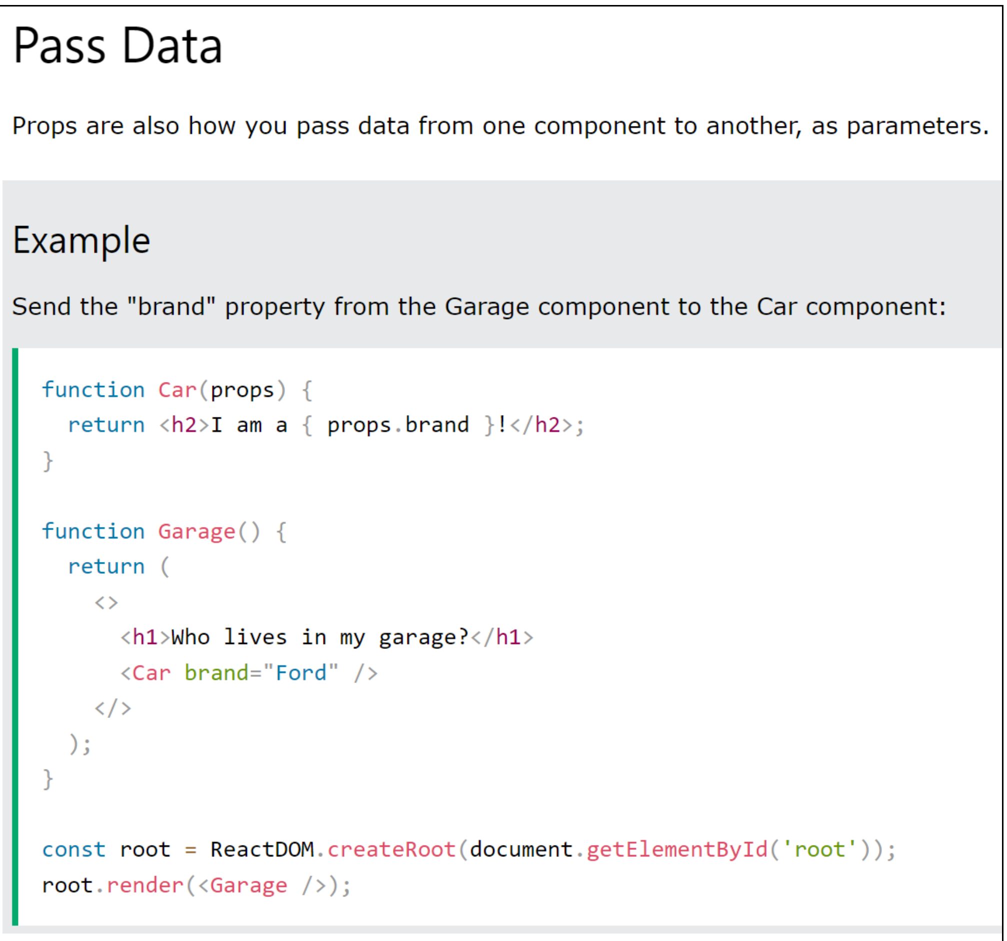
```
const myElement = <Car brand="Ford" />;
```

The component receives the argument as a `props` object:

Example

Use the brand attribute in the component:

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}
```



Pass Data

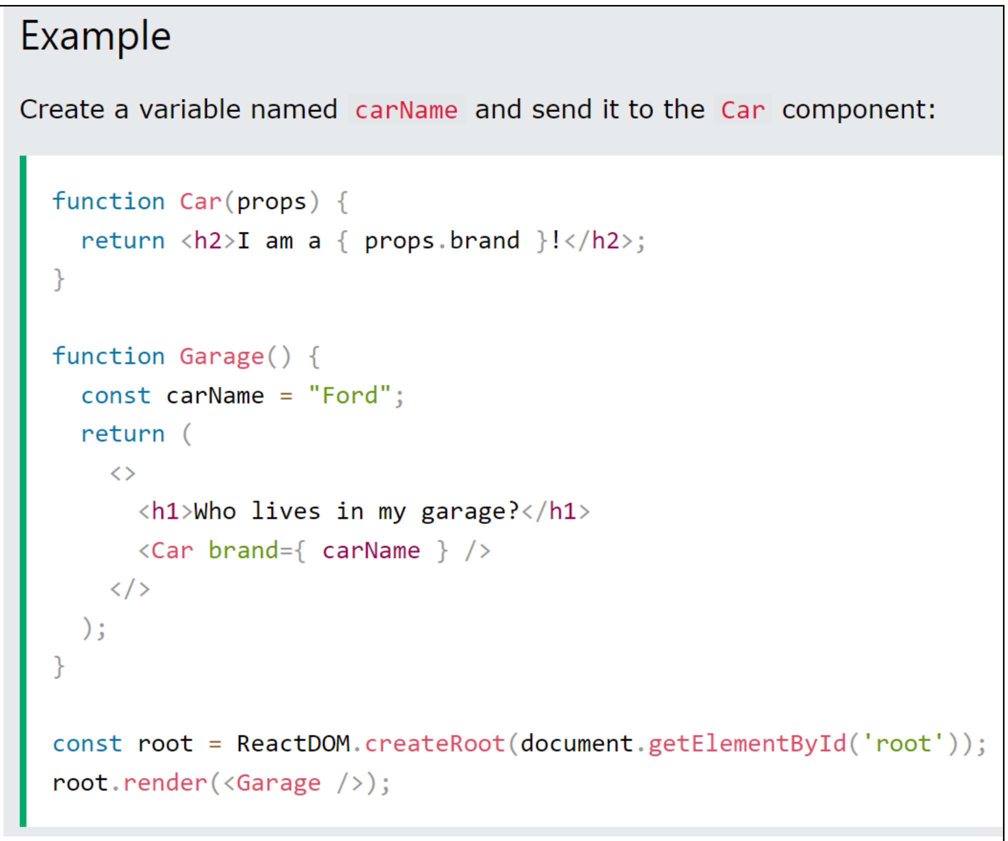
Props are also how you pass data from one component to another, as parameters.

Example

Send the "brand" property from the Garage component to the Car component:

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
    </>
  );
}

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

If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

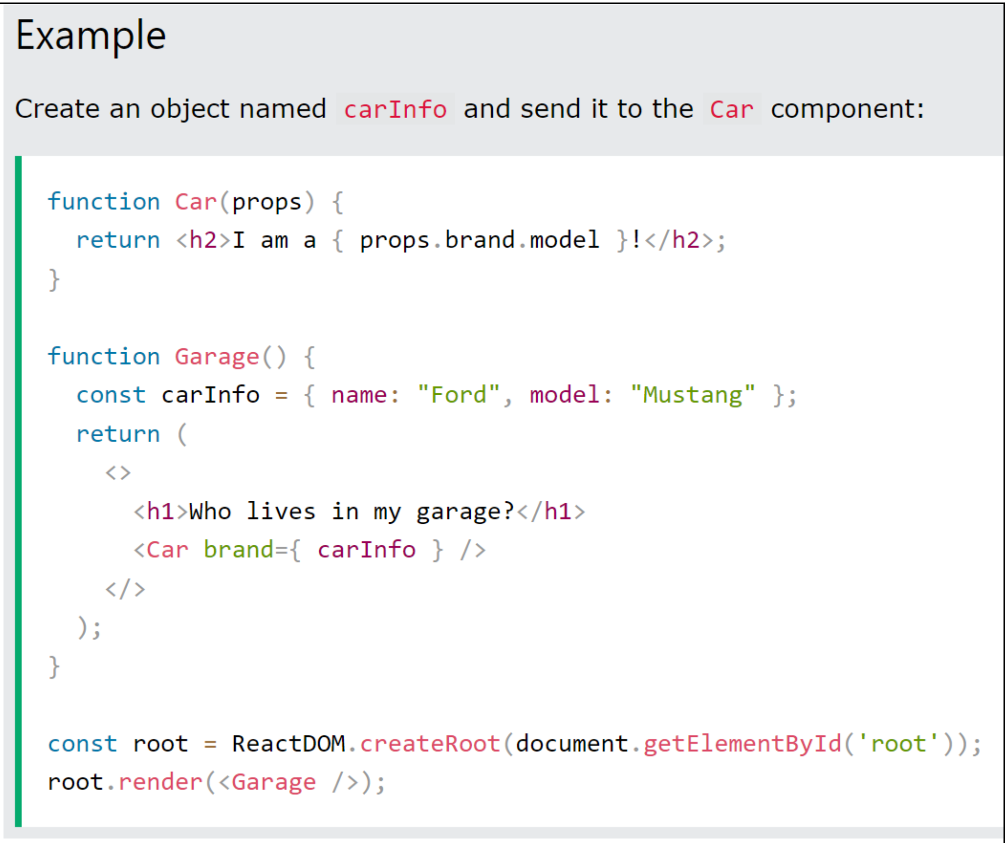


Example

Create a variable named `carName` and send it to the `Car` component:

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
  const carName = "Ford";
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={ carName } />
    </>
  );
}

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

Or if it was an object:



Example

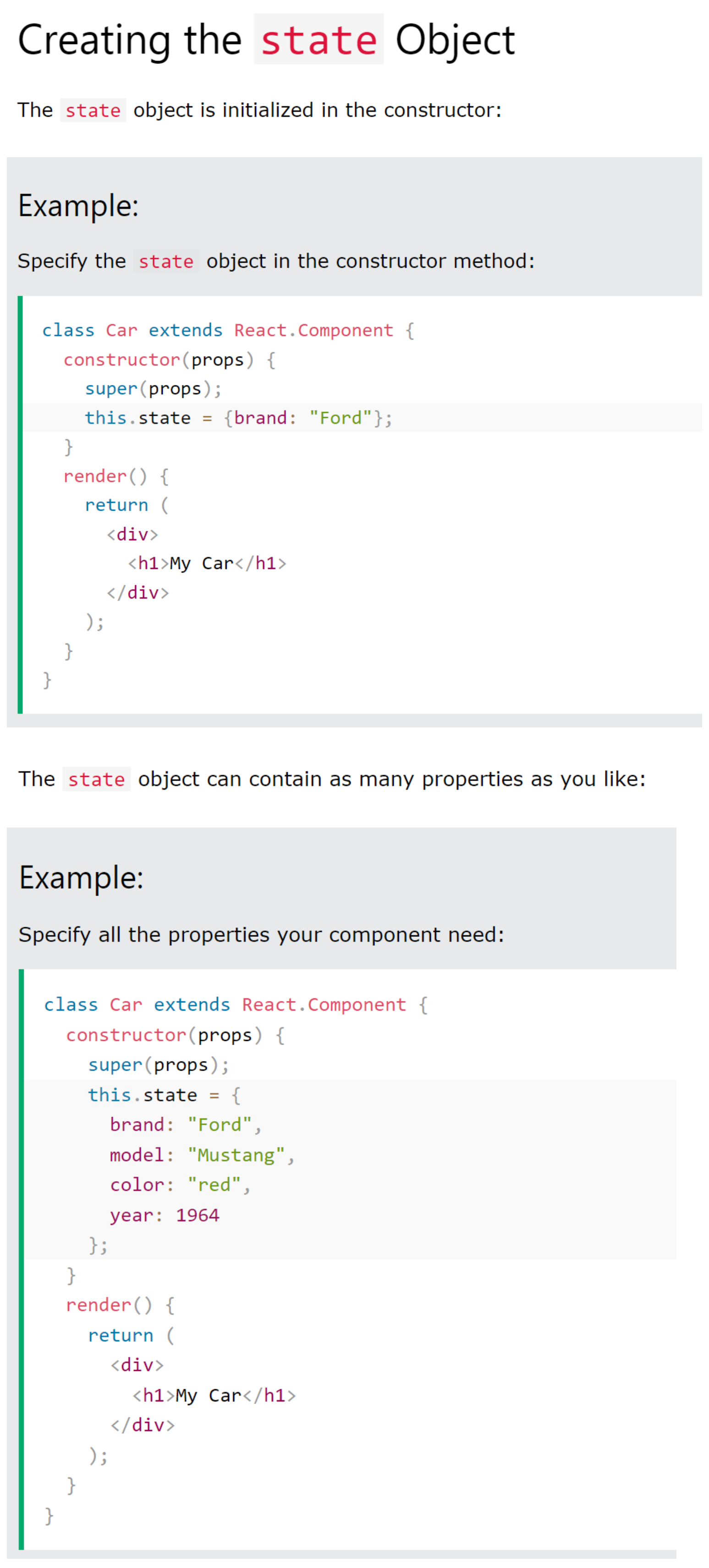
Create an object named `carInfo` and send it to the `Car` component:

```
function Car(props) {
  return <h2>I am a { props.brand.model }!</h2>;
}

function Garage() {
  const carInfo = { name: "Ford", model: "Mustang" };
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={ carInfo } />
    </>
  );
}

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

Note: React Props are read-only! You will get an error if you try to change their value.

State

React components has a built-in state object.
The state object is where you store property values that belong to the component.
When the state object changes, the component re-renders.



Creating the `state` Object

The `state` object is initialized in the constructor:

Example:

Specify the `state` object in the constructor method:

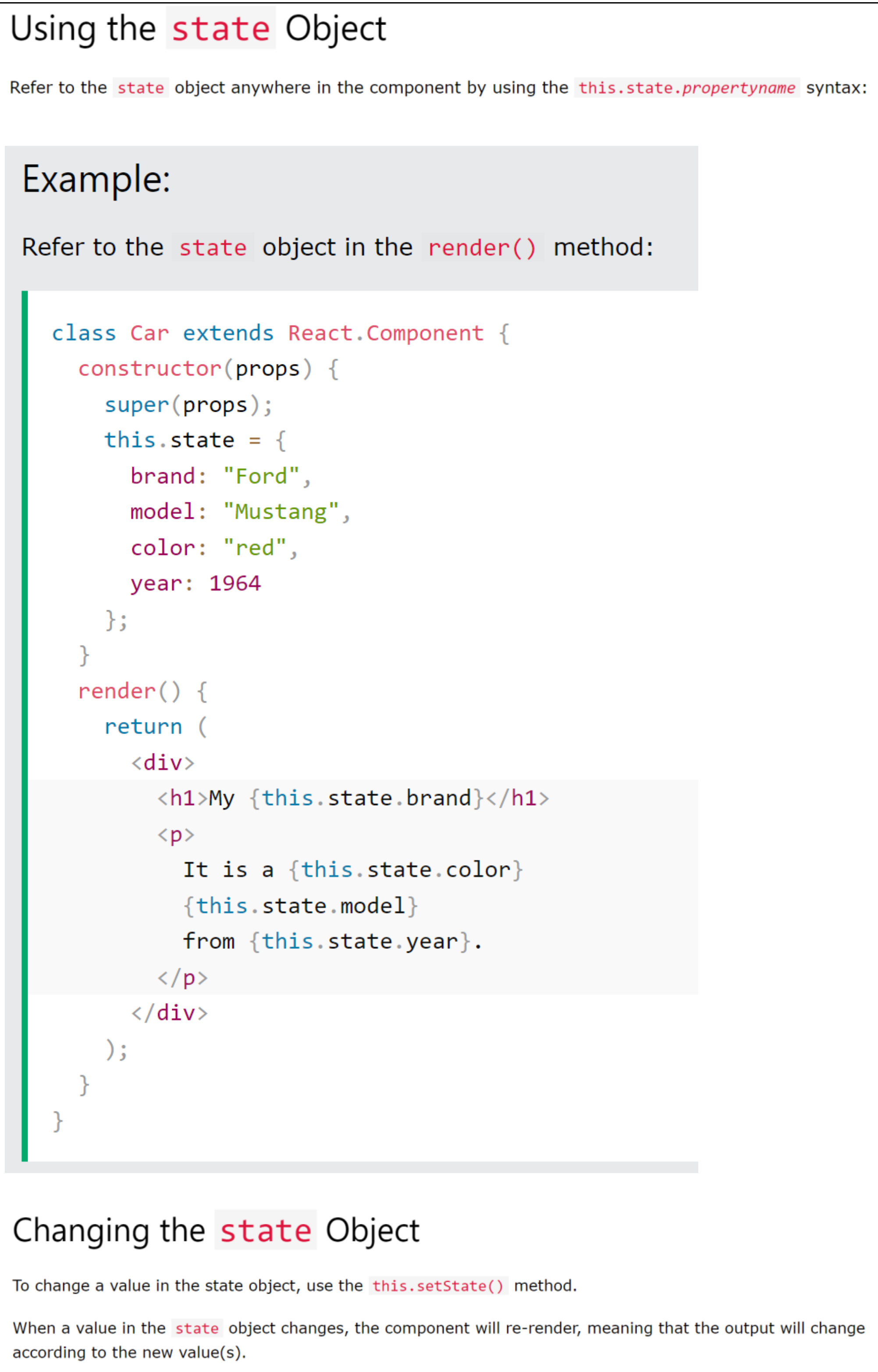
```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

The `state` object can contain as many properties as you like:

Example:

Specify all the properties your component need:

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```



Using the `state` Object

Refer to the `state` object anywhere in the component by using the `this.state.propertyname` syntax:

Example:

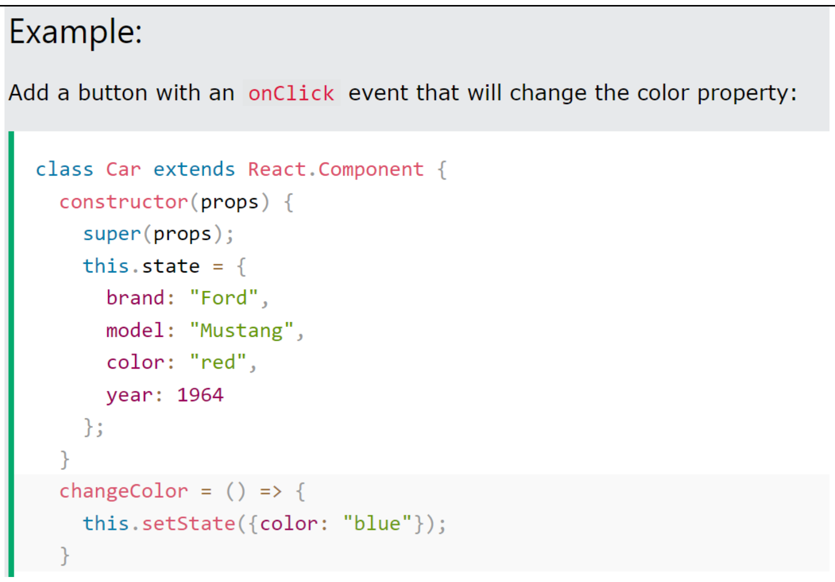
Refer to the `state` object in the `render()` method:

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
      </div>
    );
  }
}
```

Changing the `state` Object

To change a value in the state object, use the `this.setState()` method.

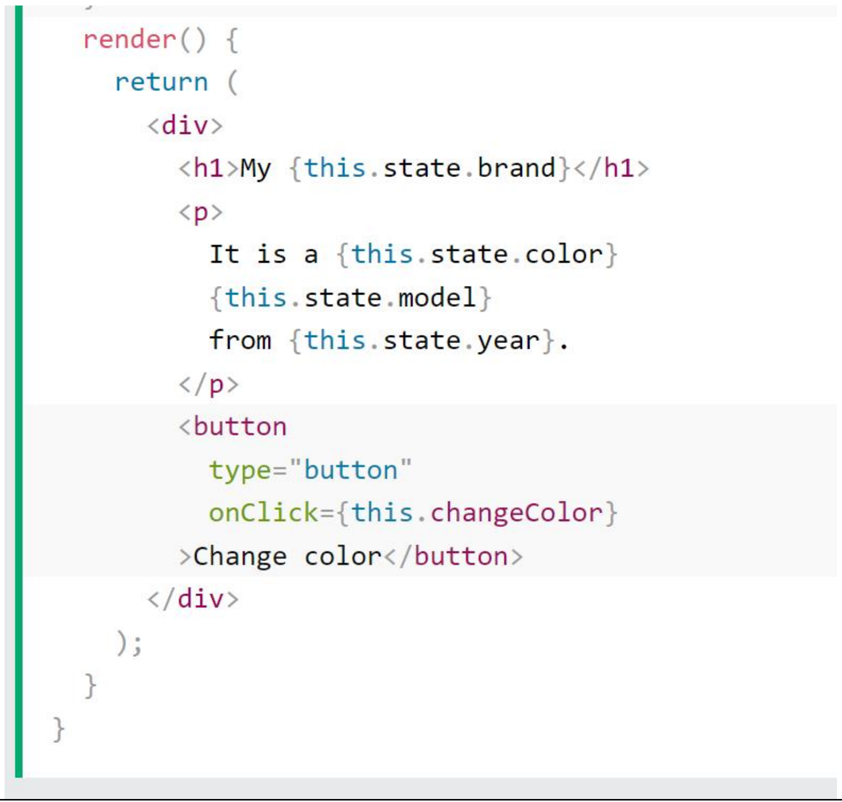
When a value in the `state` object changes, the component will re-render, meaning that the output will change according to the new value(s).



Example:

Add a button with an `onClick` event that will change the color property:

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
```



```
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
        <button
          type="button"
          onClick={this.changeColor}
        >Change color</button>
      </div>
    );
  }
}
```

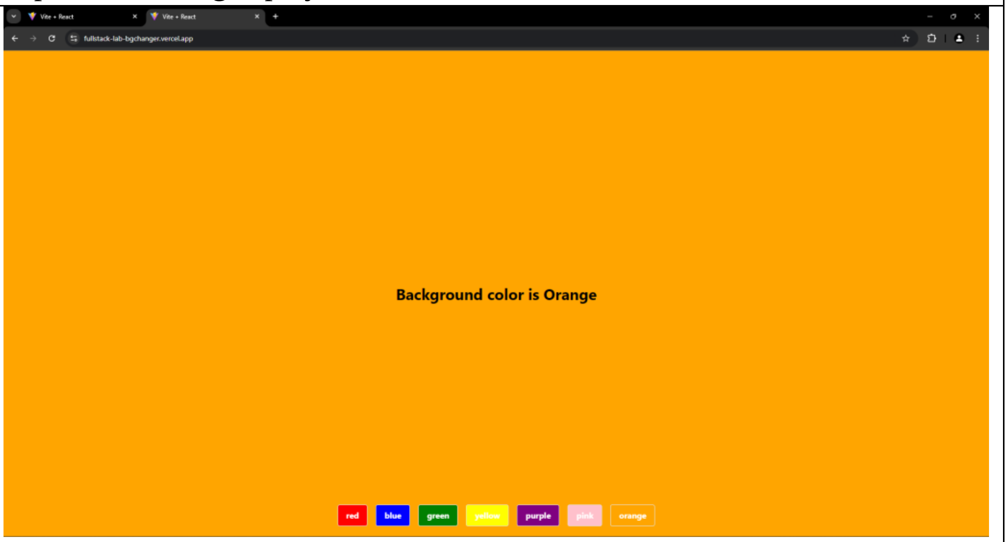
Code

App.jsx

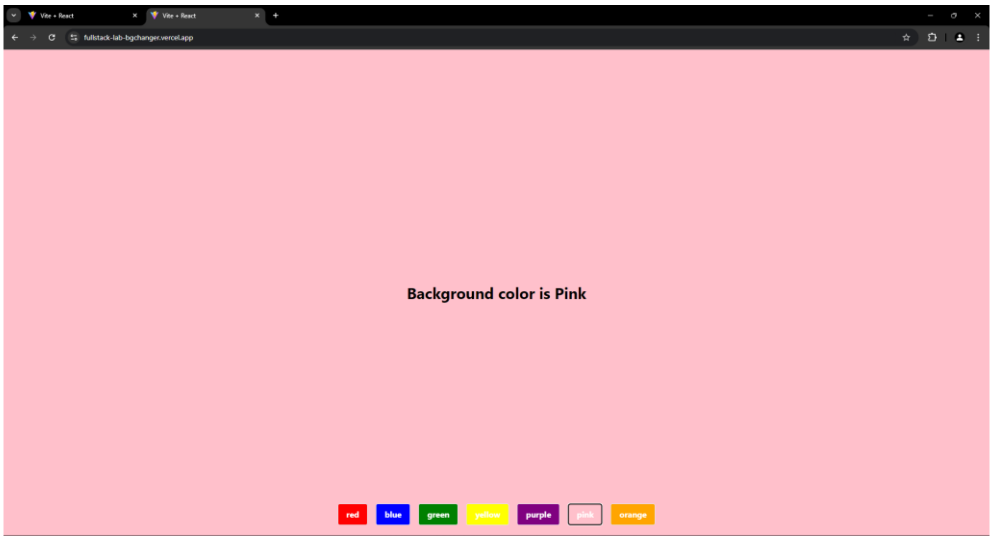
```
import React, { useState } from 'react';
import BgChanger from './components/BgChanger';
import Bgdisplay from './components/Bgdisplay';

function App() {
  const [bgColor, setBgColor] = useState('white');
  const changeBackground = (color) => {
    setBgColor(color);
  };

  return (
    <div
      className="min-h-screen flex flex-col justify-center items-center"
      style={{ backgroundColor: bgColor }}  // Use inline style for dynamic
background color
    >
      <Bgdisplay bgColor={bgColor} />
      <BgChanger changeBackground={changeBackground} />
    </div>
  );
}

export default App;

import React from 'react';

const colors = ['red', 'blue', 'green', 'yellow', 'purple', 'pink', 'orange'];

const BgChanger = ({ changeBackground }) => {
  return (
    <div className='fixed bottom-5 flex gap-4'>  {/* Corrected 'botton-5' to
'bottom-5' */}
      {colors.map((color) => (
        <button
          key={color}
          onClick={() => changeBackground(color)}
          className="text-white font-bold py-2 px-4 rounded border"  //
Corrected 'border-red-500' to 'border-red-500 border'
          style={{ backgroundColor: color }}  // Inline style for button
background color
        >
          {color}
        </button>
      ))}
    </div>
  );
}

export default BgChanger;

import React from 'react';
```

```
const Bgdisplay = ({ bgColor }) => {
  return (
    <div className='text-3xl font-bold'>
      Background color is {bgColor.charAt(0).toUpperCase() +
bgColor.slice(1)}
    </div>
  );
}

export default Bgdisplay;
```

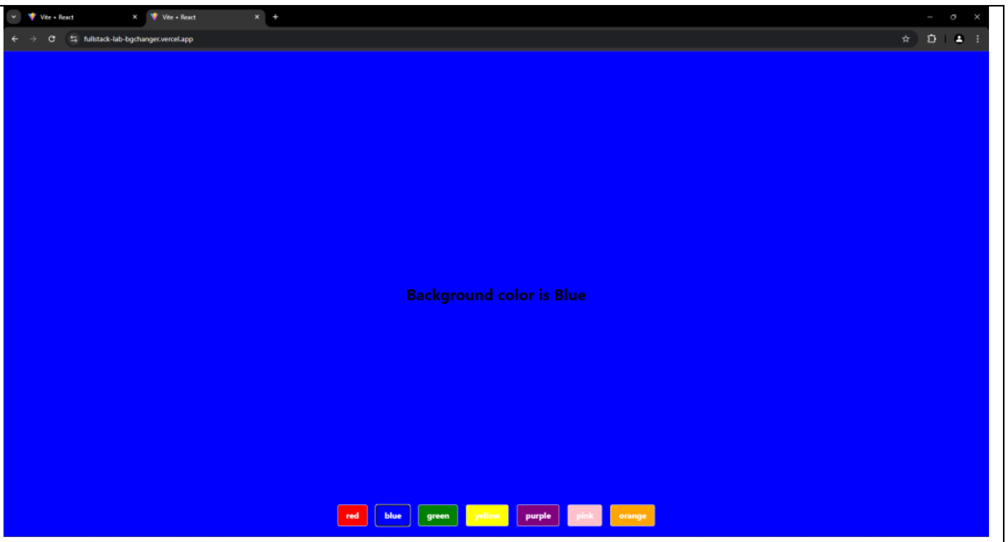
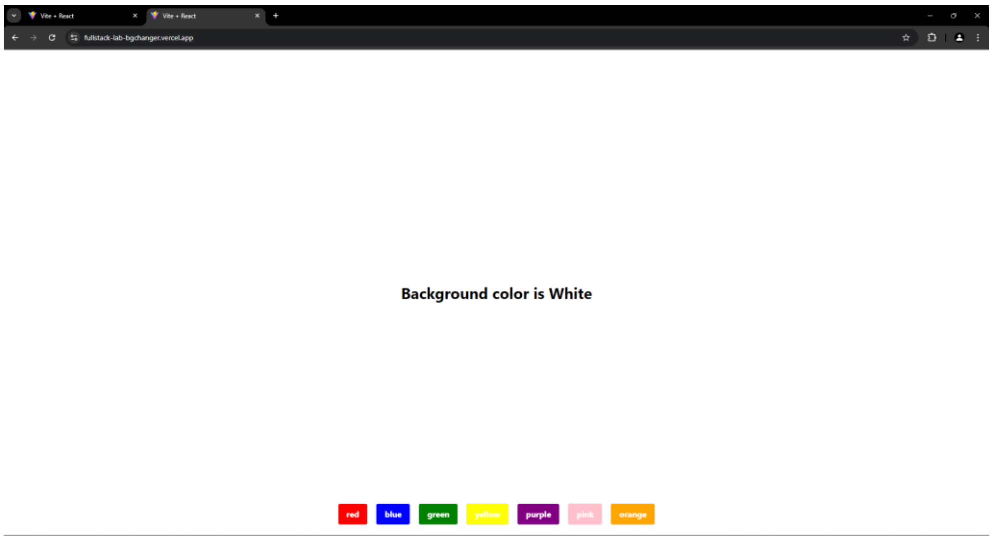
Result



Background color is Orange



Background color is Pink

	 Background color is Blue Vite + React Vite + React fullstack-lab-bgchanger.vercel.app red blue green yellow purple pink orange  Background color is White Vite + React Vite + React fullstack-lab-bgchanger.vercel.app red blue green yellow purple pink orange
Conclusion	In this experiment, we learned to use JSX for writing HTML in JavaScript, created and managed React components, and understood props and state. JSX simplifies UI creation, components help build reusable UI elements, props pass data to components, and state manages dynamic data within components.