

COMPONENTS, STATE & PROPS

APRIL 2022

YONATAN BENEZRA

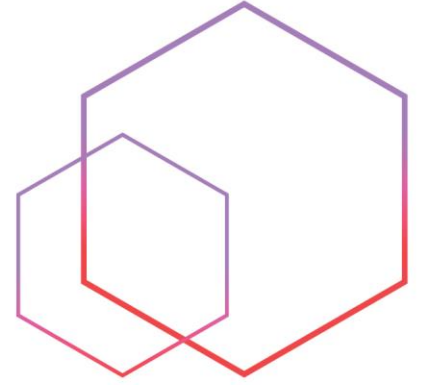
- DO NOT DISTRIBUTE -

- DO NOT COPY - ALL RIGHTS RESERVED TO THE
AUTHOR -

In this module, we will learn about React by creating a project.

React Color Switcher

First, we're going to build a color picker application -- the background of the page will change color based on a button the user selects.



Color Picker



The first step is to build the static user interface. This is how we will do so:

- We'll add an enclosing div which will be the top-level element of our React application.
- We'll add another div inside of that one, which will allow our content to be centered on the page.
- We'll add a header to title our page, and then we'll add three buttons.
- We will add some `className` attributes to our elements.

In JavaScript, the term 'class' is used for creating classes in object-oriented programming, so, React can't use the word `class` to add class names for styling groups of elements. Therefore, it will use `'className'` instead.

We will add the following `classNames` to our elements:

```
import React from 'react'
import './App.css'
const App = () => {
  return (
    <div className='react-root'>
      <div className='centered'>
        <h1>Color Picker</h1>
        <button className='red'>red</button>
        <button className='blue'>blue</button>
        <button className='yellow'>yellow</button>
      </div>
    </div>
  )
}
export default App
```

Since this tutorial is focused on React, simply copy some CSS code into your App.css. Remove what's in there and replace it with the following:

```
html, body, #root, .react-root {
  height: 100%;
  width: 100%;
  background-color: white;
  color: black;
  display: flex;
  justify-content: center;
  align-items: center;
}

.centered {
  text-align: center;
}

button {
  padding: 10px;
  margin: 5px;
  border: 2px solid white;
  color: white;
  font-size: 20px;
}

.red {
```

```
background-color: red;
color: white;
}

.blue {
background-color: blue;
color: white;
}

.yellow {
background-color: yellow;
color: black;
}
```

Your app should now look like [this](#):

Now, we need to make it do something!

Any variables that should have the option to change while our application is running need to be stored in state.

This will cause React to automatically update our component's appearance each time a state variable is updated.

Any questions up to this point?

React State

In order to utilize state, you must import the useState hook from React.

In your App.js component, update the first line of code.

```
+ import React, { useState } from 'react'
import './App.css'
```

Every line that starts with a "+" is a line that you have added.

The useState hook takes one argument: what the initial value of state will be. It then returns two values in an array:

- The first is the value of the state variable.
- The second is a function that will allow us to update the relevant state.

We will use array destructuring to set both of the items returned to their own variables.

```
import React, { useState } from 'react'
import './App.css'

function App () {
```

```
+ const [color, setColor] = useState('')
return (
  <div className='react-root'>
    <div className='centered'>
      <h1>Color Picker</h1>
      <button className='red'>red</button>
      <button className='blue'>blue</button>
      <button className='yellow'>yellow</button>
    </div>
  </div>
)
}

export default App
```

If you console log each item, you'll see the color variable is an empty string because we provided useState with the argument ''.

Change that empty string to 'blue', and you'll see that 'color' will now store the value blue!

setColor is a function, which is used to update the color variable.

In order to continue with this project, we need to learn about event listeners:

Event listeners:

Now, let's add an **event listener** so that when a user clicks on the buttons, the color stored in state will be updated.

- Display the current value of color on the interface.
 - This can be done by writing the color variable in curly braces. This tells React that any code inside the curly braces is JavaScript code.
- Add an onClick attribute to the first button.
- After the onClick is added, add a function that will run when the event fires. This is how event listeners are written in React.
 - For now, simply console.log('clicked').

```
import React, { useState } from 'react'
import './App.css'

function App () {
  const [color, setColor] = useState('')
  return (
    <div className='react-root'>
```

```

    <div className='centered'>
      <h1>Color Picker</h1>
+ {color}
+ <button className='red' onClick={() =>
console.log('clicked')}>
    red
  </button>
  <button className='blue'>blue</button>
  <button className='yellow'>yellow</button>
</div>
</div>
)
}
export default App

```

Check out your JavaScript console and see what's happening!

We will now change the event listener function to change the color state variable instead. This can be done by using the setColor function that useState gave us.

```

<button className='red' onClick={() => setColor('red')}>
  red
</button>

```

You should now see that when you click on the button, the word "red" is displayed on the page! Now, let's make both of the other buttons work as well.

```

<button className='blue' onClick={() =>
setColor('blue')}>blue</button>
<button className='yellow' onClick={() =>
setColor('yellow')}>yellow</button>

```

The last thing that we need to do, in addition to displaying the names of the colors on the page, is to actually change the color of the page.

In your CSS file, you should already have three classes for our colors:

- yellow
- red
- blue

Now, you must add those classes into your react-root element, so that it changes colors to match your color variable.

First, you must make your className take JavaScript code instead of just a string, and then, use string interpolation to add your color class to the element.

```
<div className={`react-root ${color}`}>
```

Your final code should look like this:

```
import React, { useState } from 'react'
import './App.css'

function App () {
  const [color, setColor] = useState('')

  return (
    <div className={`react-root ${color}`}>
      <div className='centered'>
        <h1>Color Picker</h1>
        <button className='red' onClick={() =>
setColor('red')}>red</button>
        <button className='blue' onClick={() =>
setColor('blue')}>blue</button>
        <button className='yellow' onClick={() =>
setColor('yellow')}>yellow</button>
      </div>
    </div>
  )
}

export default App
```

React Props

Now we've used some of React's most important features: JSX and state. There are two more that I want to show you:

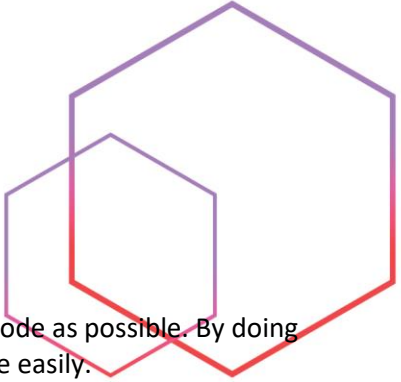
Components and Props

We are actually currently using a component: App.

However, we want to make our components small and reusable. Right now, our buttons follow a pattern.

Each one:

- Displays text
- Has a className
- Has an onClick event.



Create a second ColorChangeButton component so that you can reuse as much code as possible. By doing so, if you want to update the buttons in the future, you will be able to do so more easily.

The first step is to create another file in your src/ folder called ColorChangeButton.js.

Now, create a second React component in this file.

```
// ColorChangeButton.js
import React from 'react'

function ColorChangeButton () {
  return (
    <button>Hi!</button>
  )
}

export default ColorChangeButton
```

Now, return to your App.js and import your ColorChangeButton:

```
// App.js

import React, { useState } from 'react'
import './App.css'
+ import ColorChangeButton from './ColorChangeButton'
```

In your JSX code, create three instances of your ColorChangeButton.

```
// App.js
return (
  <div className={`react-root ${color}`}>
    <div className='centered'>
      <h1>Color Picker</h1>
+ <ColorChangeButton />
+ <ColorChangeButton />
+ <ColorChangeButton />
      <button className='red' onClick={() =>
setColor('red')}>red</button>
      <button className='blue' onClick={() =>
setColor('blue')}>blue</button>
    </div>
  </div>
)
```



```

        <button className='yellow' onClick={() =>
setColor('yellow')}>yellow</button>
      </div>
    </div>
  )

```

Boom! Now you should have three more buttons that show up on the page that all say 'Hi!'. This is how you create and include a second component in React.

However, right now your components are pretty boring, as they all say the same thing. These should eventually replace the three color-changing buttons you have created, so you will need to allow your button to be a different color, and to contain different text.

React uses unidirectional data flow, which means we can only pass data from a parent component to a child component. We will use props to pass data from one component to another.

```

// App.js
return (
  <div className={`react-root ${color}`}>
    <div className='centered'>
      <h1>Color Picker</h1>
      + <ColorChangeButton color='red' />
      + <ColorChangeButton color='blue' />
      + <ColorChangeButton color='yellow' />
      <button className='red' onClick={() =>
setColor('red')}>red</button>
      <button className='blue' onClick={() =>
setColor('blue')}>blue</button>
      <button className='yellow' onClick={() =>
setColor('yellow')}>yellow</button>
    </div>
  </div>
)

```

In your parent component, App, you can use what looks like an HTML attribute to send props. In your case, 'color' is the name of your prop, and the value comes after the equal sign; 'red' for the first component, 'blue' for the second, and 'yellow' for the third.

Now, you need to use those props in your child component. Switch over to ColorChangeButton.js. First, make your function take the parameter props.

```

// ColorChangeButton.js
function ColorChangeButton (props) {

```

```
...  
}
```

Then, you can `console.log` props before the return to see what's there:

```
{ color: 'red' } { color: 'blue' } { color: 'yellow' }
```

It's an object! React combines each prop we send from the parent component into an object, with each key and value in the child. So, to access the color in your child component, enter `props.color`.

Let's make your button display the color as its text, and also add the color as a class to the button so that the correct color will be displayed.

```
// ColorChangeButton.js  
import React from 'react'  
  
function ColorChangeButton (props) {  
  return (  
+   <button className={props.color}>{props.color}</button>  
  )  
}  
  
export default ColorChangeButton
```

Your buttons now look as they are supposed to! The last thing that you need to do is make the click event work. In your `App.js`, you wrote the following code to change the current color:

```
<button className='red' onClick={() =>  
  setColor('red')}>red</button>
```

The one issue we have is that `setColor` is defined in our `App` component, so we don't have access to its `ColorChangeButton`.

Good news though: there is a way to pass data from a parent component to a child component, which we learned in the past step: props!

Let's pass the `setColor` function down as a prop to our `ColorChangeButton` component. You can also delete your three original buttons, as you no longer need them.

```
// App.js  
return (  
  <div>
```

```

    <div className={`react-root ${color}`}>
      <div className='centered'>
        <h1>Color Picker</h1>
+ <ColorChangeButton color='red' setColor={setColor} />
+ <ColorChangeButton color='blue' setColor={setColor} />
+ <ColorChangeButton color='yellow' setColor={setColor}
/>
      </div>
    </div>
  )

```

Now, if you go back to the ColorChangeButton and console.log what the props are, you'll see that you have a second item in the object. For example:

```

{
  color: "red"
  setColor: f ()
}

```

Let's use that setColor function:

```

function ColorChangeButton(props) {
  return (
+ <button className={props.color} onClick={() =>
props.setColor(props.color)}>
    {props.color}
  </button>
  )
}

export default ColorChangeButton

```

Now, each button should work as expected! This pattern of passing the state change function down from parent to child components is called inverse data flow.

It allows us to circumvent the unidirectional data flow nature of React.