**Random Color Picker**

In this project, we’ll build a program that helps designers think of new color schemes.

Our program will set the screen’s background to a random color. Clicking a button will refresh to a new, random color. [Random generators](https://en.wikipedia.org/wiki/Oblique_Strategies) are a well-known tool for breaking a creative rut.

Let’s get started!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/13Complete**

Mark the tasks as complete by checking them off

**1.**

Take a look at the Random component class. Random‘s job is to store a random color, and to use that color to update the screen’s background.

First, let’s store this random color as state.

Give Random a constructor() method. Give constructor() one parameter, named props.

Inside the body of constructor(), write the line super(props);.

Still inside the body of constructor(), on a new line, set this.state equal to this object:

{ color: [x, y, z] }

Instead of x, y, and z, use three numbers between 0 and 255.

Hint

constructor(props) {

super(props);

this.state = {

color: [92, 132, 153]

};

}

**2.**

Good!

Change the three numbers inside of the color array to three different numbers. Click Save. The background color should change!

**3.**

It would be nice to know what color we’re looking at!

In the render() method, inside of the <h1></h1>, add the text, Your color is \_\_\_.

Instead of \_\_\_, access this.state.color!

Hint

<h1 className={this.isLight() ? 'white' : 'black'}>

Your color is {this.state.color}.

</h1>

**4.**

That’s not a very friendly way to display a color!

In Random, find the method named formatColor. This method transforms an rgb array into something a bit more readable.

Inside of the <h1></h1>, instead of simply using this.state.color, call the formatColor function and pass in this.state.color as an argument.

Hint

Your color is {this.formatColor(this.state.color)}.

**5.**

That’s a bit better!

A user should be able to click on a button to pick a new random color. In the code editor, you can see a **Button.js** file. That will be your button!

Select **Button.js**. Add the word export so that you are exporting the Button component class.

Hint

export class Button extends React.Component {

**6.**

Good! Now, if you import Button into **Random.js**, and you’ll get the Button component class that you want.

Select **Random.js**. Near the top of the file, create a new line after import ReactDOM from 'react-dom';.

On this new line, use import the Button component class.

**Button.js** and **Random.js** share the same parent directory.

Hint

import { Button } from './Button';

**7.**

Now you’re ready to render a <Button /> instance!

Inside of Random‘s render method, after the <h1></h1>, add a <Button />.

Give your <Button /> this attribute:

light={this.isLight()}

Hint

</h1>

<Button light={this.isLight()} />

</div>

**8.**

You can see your <Button /> instance in the browser. However, clicking it doesn’t do anything!

You need to define an *event handler* that updates this.state.color to a new random color.

Give Random a new method named handleClick.

Inside of .handleClick()‘s body, call this.setState(). As an argument, pass this.setState() an object with the following property:

color: this.chooseColor()

Hint

handleClick() {

this.setState({

color: this.chooseColor()

});

}

**9.**

You just created a new method, that you will eventually use as an *event handler*. Your new method uses this.

That means that you need to *bind* your new method.

Add a new line to your constructor() method. On this new line, bind handleClick().

Hint

constructor(props) {

super(props);

this.state = {

color: [10, 20, 30]

};

this.handleClick = this.handleClick.bind(this);

}

**10.**

Great! this.handleClick() will update this.state.color to a new, random color.

Now that you’ve defined an *event handler*, you can pass it to another component as a prop. This is a pattern that you’ll see much more of in the next course.

In Random‘s render method, give <Button /> an attribute with a *name* of onClick. Set onClick‘s *value* equal to the handleClick method.

Hint

</h1>

<Button

light={this.isLight()}

onClick={this.handleClick} />

</div>

**11.**

Only one more step!

Select **Button.js**. In the render function, give the <button></button> an onClick attribute. Set onClick‘s *value* equal to the passed-in prop.

Hint

Set the <button></button>‘s onClick value to {this.props.onClick}.

**12.**

Try clicking the button a few times!

If you tried to make sense of the built-in parts of Random, you may have come up confused. This is because Random includes two special functions that we haven’t discussed yet: componentDidMount and componentDidUpdate.

These functions are examples of a powerful React feature called *lifecycle methods.* You’ll learn all about lifecycle methods in *Introduction to React.js: Part II*.

BONUS: Notice that the <h1></h1>‘s text is white if the screen’s background is a darker color, but the text is black is the screen’s background is a lighter color. Similarly, the <button></button> changes colors based on whether the background is dark or light. Can you figure out how that works?