**Exercise 5 More React Component Concepts**

1. Moving state into the root component: Open a command prompt and navigate to react-dev-env. Start the development server: npm start
2. Open "src/App.js" from the "react-dev-env" project in an editor.
   1. The title property is currently displayed in the Header component. The variable "title" is in scope so it is used directly.

const title = "My React App" function Header(){

return <h3 style={divStyle} >{title}</h3>;

}

* 1. Instead of relying on scope, let's pass the title in as a property like we did with the "text" in the footer.
  2. Edit the Header component. Pass "props" into the Header component function.
  3. Change the JSX expression that displays the title to get its value from "props.title". Update the App component to pass title as a property to the Header component.
  4. The resulting Header component should look like this:

function Header(props){

return <h3 style={divStyle} >{props.title}</h3>;

}

* 1. And the Header tag in the App component looks like this:

<Header title={title}/>

* 1. Notice how the values for title, author and footerText are coming from variables defined outside the App component. This works because they are in scope but we would like to move those values inside the component.

function App(){ return (

<div className={'boxed'}>

<Header title={title}/>

<Body author={author} />

<Footer text={footerText} />

</div> );

}

* 1. Open the src/index.js file in an editor. You should see the following:

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>, document.getElementById('root')

);

* 1. We want to pass a single object into the App component that includes all the data needed to render it. Add a variable to index.js called 'scope' that holds values for all the variables we previously added in App.js including: 'title', 'footerText', and 'author'. You can copy the values over from App.js. The variable should look like this:

let scope = {

title: "My React App", footerText: "footer text", author: {

name:"John Doe", phone: "800-555-1212",

email: "john@gmail.com"

}

}

* 1. Next, modify <App /> and use the spread operator (...) to pass in the 'scope' object's properties. The properties passed in will be made available on the props object.

ReactDOM.render(

<React.StrictMode>

<App {...scope} />

</React.StrictMode>,

document.getElementById('root')

);

* 1. Save the index.js file. In the App.js file, the App component should look like this:

function App(props) { return (

<div className={'boxed'} >

<Header title={title} />

<Body author={author} />

<Footer text={footerText} />

</div>

);

}

* 1. Modify the values being passed to each component so that they are coming from props:

function App(props) { return (

<div className={'boxed'} >

<Header title={props.title} />

<Body author={props.author} />

<Footer text={props.footerText} />

</div>

);

}

* 1. Save the App.js file. The app has compiled with warnings. Now that we are passing in values through props we don't need these variables in App.js. Remove the 'title', 'footerText', and 'author' variables from App.js. Save the App.js file again. The previous warnings should be gone.

1. Displaying lists: let’s display a list of books in our app.
   1. Open index.js in an editor. Add the following "books" property to the scope object:

, books: [

{isbn:'123', title:'The Time Machine', price:5.95 },

{isbn:'456', title:'War of the Worlds', price:6.95 },

{isbn:'789', title:'The Invisible Man', price:4.95 }

]

* 1. The full scope object should look like this:

let scope = {

title: "My React App", footerText: "footer text", author: {

name:"John Doe", phone: "800-123-1212",

email: "john@gmail.com"

},

books: [

{isbn:'123', title:'The Time Machine', price:5.95 },

{isbn:'456', title:'War of the Worlds', price:6.95 },

{isbn:'789', title:'The Invisible Man', price:4.95 }

]

}

* 1. Save the index.js file. Open App.js in an editor.
  2. Add a BookList functional component to the Body component. BookList should takes props as a parameter and returns an unordered list of book items. Create the BookList component. Edit BookList to return <ul></ul>. Inside the <ul> tags add a JSX expression that uses the array.map function to display a list of book titles. Add BookList right after the author paragraph in the Body component. Make sure to pass all props to the Body component. Inside the Body component pass books to the BookList component. The resulting code changes should look like this:

function App(props) { return (

<div className={'boxed'} >

<Header title={props.title} />

<Body {...props} />

<Footer text={props.footerText} />

</div>

);

}

function Body(props){

return ( <div><p>Author:{props.author.name}</p>

<BookList books={props.books} />

<p>some random text</p>

</div> );

}

function BookList(props){ return (

<ul>

{props.books.map(

(book,index)=>{return (<li key={index}>{book.title}</li>)}

)}

</ul>

);

}

* 1. Save the App.js file. Refresh the browser.

1. Updating input fields. Let’s add an <input> field and update the app so that when users type into the field the changes are shown on the screen.
   1. Open index.js in an editor. Add a "color" property to the scope object with the value "blue".

let scope = {

title: "My React App", footerText: "footer text", color: "blue",

* 1. Save the index.js file. Open the App.js file in an editor.
  2. Add an input field just before the </div> tag in the Body component. The type of the input should be "text". The value of the field should come from the "color" property of the scope object. The name of the input field should be set to 'color'. You will need to pass the "color" property to the Body component where it appears in the App component. You can do this the same way you passed author and books. Modify the text in the <p> above the input field. Change it from "some random text" to "Enter your favorite color:"
  3. The code should now look like this:

function Body(props){ return ( <div>

<p>Author:{props.author.name}</p>

<BookList books={props.books} />

<p>Enter your favorite color:</p>

<input type='text' name='color' value={props.color} />

</div> );

}

* 1. Save the App.js file. Refresh the browser and the input field should be visible in the browser. Note: if you try to enter any text into the field it does not appear on the screen. React keeps supplying the same color value to the input field each time it is rendered. We need to respond to any changes the user makes to the input field and update the color value on the scope object. Before we do this we need to wrap the ReactDOM.render() call in a method so that we can call it after the "color" has been updated.
  2. Open the index.js file in an editor and locate the following render code:

ReactDOM.render(

<React.StrictMode>

<App {...scope} />

</React.StrictMode>, document.getElementById('root')

);

* 1. Create a 'renderApp()' function and copy the above code into it:

function renderApp(scope){

ReactDOM.render(

<React.StrictMode>

<App {...scope} />

</React.StrictMode>, document.getElementById('root')

);

}

* 1. And since ReactDOM.render is no longer executing (since it's wrapped inside a function now) we need to call this new function as the last line of the 'src/index.js' file:

renderApp(scope);

* 1. We are going to need a function to handle changes to the input field. Add the following function into index.js file right after the renderApp function:

function handleChange(event){

scope[event.currentTarget.name] = event.currentTarget.value; renderApp(scope);

}

* 1. This function updates the input field's value on the scope object and then uses the updated scope to re-render the app. 'handleChange' needs to be called from inside the Body component. To make it available there we need to add it to the scope object as well.
  2. Add the handleChange property to the scope object in index.js as shown here:

color: "blue", handleChange: handleChange, author: { ...

* 1. This way handleChange will be passed in along with all the other properties and will be available on props. Save the index.js file.
  2. Open the App.js file in an editor. Add an onChange handler to the text input field in the Body component:

<input type='text' name='color' value={props.color}

onChange = {props.handleChange} />

* 1. Now text entered into the input field should be visible in the input field as you type it and it will be added at the same time to the scope.color property. Save the App.js file. Refresh the browser and test your changes.

1. Click Event Handling
   1. Open index.js in an editor. Add a handleButtonClick function right after the handleChange function. Set scope.message so that it appears like the text below, with the color coming from scope.color:

You like the color Red!

* 1. As the last line of the handler make a call to renderApp
  2. The handleButtonClick method should look like:

function handleButtonClick(event){

scope.message = "You like the color " + scope.color + "!"; renderApp(scope);

}

* 1. Add the handleButtonClick function to the scope object right after the handleChange function:

handleChange: handleChange, handleButtonClick: handleButtonClick, author: { ...

* 1. Add a "message" property to the scope object. For now set its value to an empty string.

title: "My React App", footerText: "footer text", color: "blue",

message: "",

* 1. Save the index.js file. Open App.js in an editor. Add an input element with type=button right below the other input field in the Body component.
     1. Input field's type is button.
     2. Add an onClick handler named "handleButtonClick".
     3. Add a value attribute set to "Click Here".
  2. Add paragraph below the button that gets its text from the "scope.message" property. Remember that all properties of the scope object were passed into the <Body /> component and should therefore be referenced using the 'props' object.
  3. The Body component should now look like this:

function Body(props){

return ( <div><p>Author:{props.author.name}</p>

<BookList books={props.books} />

<p>Enter your favorite color:</p>

<input type='text' name='color' value={props.color}

onChange = {props.handleChange} />

<input type='button'

value='Click Here'

onClick = {props.handleButtonClick} />

<p>{props.message}</p>

</div> );}

* 1. Save the App.js file.
  2. Refresh the browser and try out the app. Enter your favorite color and click the button. The message below the button should update to reflect your choice.

1. Passing Parameters to Event Handlers
   1. Open index.js in an editor. Add a property named "selectedIndex" to the scope object. Set its value to -1.

title: "My React App", footerText: "footer text", color: "blue",

message: "",

selectedIndex: -1,

...

* 1. Add a function named "handleListItemClick" after the "handleButtonClick" function.
     1. The method should take two parameters, "event' & "index".
     2. It should use the "index" parameter to set the "scope.selectedIndex" property and look up a book from the scope.booklist array. After updating the selectedIndex the handler should make a call to renderApp. As the last line of the method, details of the chosen book should be output to the console.
     3. The handleListItemClick method code should look like this:

function handleListItemClick(event, index){ scope.selectedIndex = index;

const book = scope.books[index]; renderApp(scope);

console.log("You chose: " + book.isbn + ", " + book.title + ", " + book.price);

}

* + 1. Add the "handleListItemClick" to the scope object right after "handleButtonClick":

handleChange: handleChange, handleButtonClick: handleButtonClick, handleListItemClick: handleListItemClick, author: { ...

* + 1. Save the index.js file. Open the App.js file in an editor. Add an onClick attribute to the <li> in the BookList component:

onClick = {(e)=>props.handleListItemClick(e, index)}

* + - 1. We are using an arrow function which autobinds "this" and we are passing multiple parameters.
      2. The event parameter is always passed first. After that, you can pass whatever other parameters you need. Here we are passing the "index" parameter.
    1. The BookList should look like this after the change:

function BookList(props){ return (

<ul>

{props.books.map(

(book,index)=>{return (<li key={index}

onClick = {(e)=>props.handleListItemClick(e, index)}

>{book.title}</li>)}

)}

</ul>

);

}

* + 1. Once a list item has been selected we would like it to be styled so that it stands out. Make these changes so that the selected item is highlighted:
       1. Make sure to pass the selectedIndex to the BookList component.
       2. Insert a statement into the <li> that sets the class to "selected" when the index of an item matches the selectedIndex

className={ index === props.selectedIndex ? "selected" : ""}

* + 1. The Resulting code should look like this:

function Body(props){ return ( <div>

<p>Author:{props.author.name}</p>

<BookList {...props} />

<p>Enter your favorite color:</p>

...

}

function BookList(props){ return ( <ul>

{props.books.map( (book, index) => {

return ( <li key={index}

onClick = {(e)=>handleListItemClick(e, index)}

className={ index === props.selectedIndex ? "selected":""}

>{book.title}</li> )}

)}

</ul>

);

}

* + 1. Save the App.js file. Open the "App.css" file in an editor. Add a style for a "selected" class that makes text bold when the class is applied. The style should look like this:

.boxed{

width:200px;

border: 1px black solid;

}

.selected{

font-weight: bold;

}

* + 1. Save the App.css file. Refresh the browser and open Chrome's Developer tools (F12) so you can see the JavaScript console. Click on Console tab.
    2. Click on a book in the list. The selected item should appear in bold and the details of the item should appear printed out in the console.