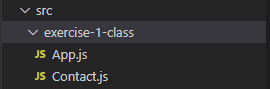
# React Props & State

## Exercise 1: Class Components

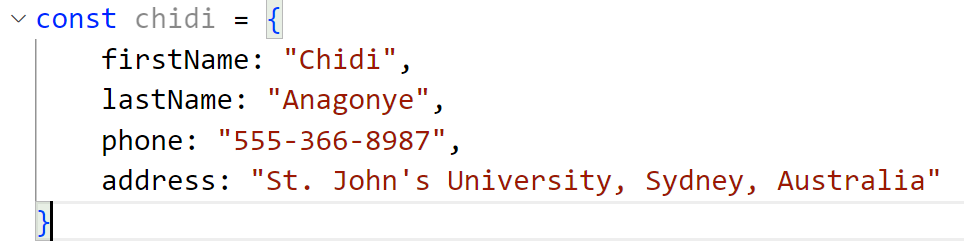
* Expand the **src** folder, then the exercise-1-class folder.



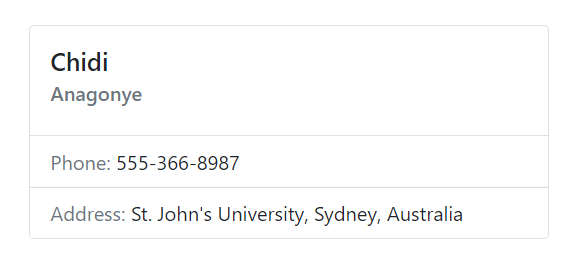
* In this folder there are two class components, the **App** component and the **Contact** component.

**The Goal**

* We want our Contact component to be able to show information about any contact. We need it to work dynamically and react to input data that another component gives it.
* For example, the App component has an example contact



* Our App component should give this contact to our Contact component and the Contact component should display it like this:



## Exercise 1: Functional Components

* Expand the src folder, then the **exercise-1-functional** folder.
* In this folder there are two **functional components**, the **App** component and the **Contact** component.

**The Goal**

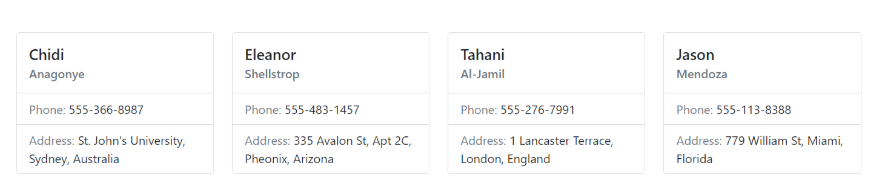
Just like with the class component version, we want our Contact component to be able to show information about any contact. We need it to work dynamically and react to input data that another component gives it.

## Exercise 2: Class Components

* Expand the src folder, then the exercise-2-class folder.
* In this folder there are three class components, the App component, the Contact component, and the ContactList component. You can mix functional and class components, but for this exercise we’ll use all class components to practice with them.
* There is also a data.js file that holds some static data that we will use. In a real app, we might pull our data from a server, but for this exercise we’ll just pull it from the data.js file.

**The Goal**

* In [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/) we set up our Contact component to show data about a contact.
* Now we want our app to actually maintain some internal state data (a list of contacts) and show that data using our Contact List component, which uses the Contact component.
* Our App should react to any changes in the list of contacts and re-render, which will re-render the ContactList component and the Contact components inside it.
* The list of contacts should start out set to the array in the variable INITIAL\_CONTACTS in the data.js file.
* So, when it’s all completed, our app should look like this:



**Tips**

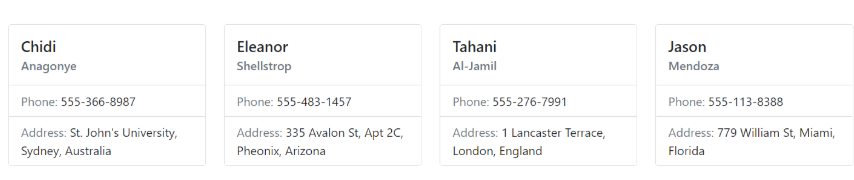
* The Contact component is identical to the one in [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/). You can copy your code from Contact.js in the exercise-1-class folder, into Contact.js in the exercise-2-class folder.
* We will store our contacts in the state of the App component. Our ContactList and Contact components are purely presentational. Presentational means they only know how to present the data, they don’t manage the data or have any data in state. They just react to changes in their props, which will flow down to **ContactList** from App (and to Contact from ContactList).
* React needs to be able to tell exactly what changed when it rerenders your component. Any component that displays a list needs to put a key attribute on the top JSX element (component or HTML element) of each item. The key should be set to something that is unique to each item in the list. It’s most often an ID.

## Exercise 2: Functional Components

* Expand the **src** folder, then the **exercise-2-functional** folder.
* In this folder there are three functional components, the **App** component, the **Contact** component, and the **ContactList** component. You can mix functional and class components, but for this exercise we’ll use all functional components to practice with them.
* There is also a data.js file that holds some static data that we will use. In a real app, we might pull our data from a server, but for this exercise we’ll just pull it from the data.js file.

**The Goal**

1. In [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/) we set up our Contact component to show data about a contact.
2. Now we want our app to actually maintain some internal state data (a list of contacts) and show that data using our Contact List component, which uses the Contact component.
3. Our App should react to any changes in the list of contacts and re-render, which will re-render the ContactList component and the Contact components inside it.
4. The list of contacts should start out set to the array in the variable INITIAL\_CONTACTS in the data.js file.
5. So, when it’s all completed, our app should look like this:



**Tips**

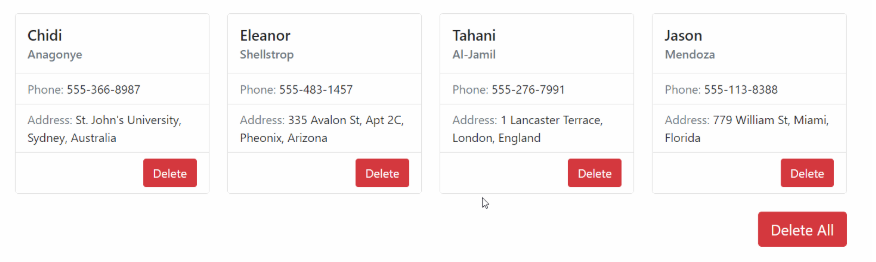
* The Contact component is identical to the one in Exercise 1. You can copy your code from Contact.js in the exercise-1-functional folder, into Contact.js in the exercise-2-functional folder.
* We will store our contacts in the state of the App component. Our ContactList and Contact components are purely presentational. Presentational means they only know how to present the data, they don’t manage the data or have any data in state. They just react to changes in their props, which will flow down to ContactList from App (and to Contact from ContactList).

## Exercise 3: Class Components

* Expand the src folder, then the exercise-3-class folder.
* In this folder there are three class components, the **App** component, the **Contact** component, and the **ContactList** component. You can mix functional and class components, but for this exercise we’ll use all class components to practice with them.
* There is also a **data.js** file that holds some static data that we will use. In a real app, we might pull our data from a server, but for this exercise we’ll just pull it from the data.js file.

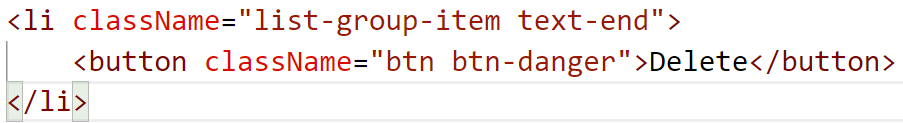
**The Goal**

* In [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/) we set up our Contact component to show data about a contact.
* In [Exercise 2](https://www.codingmadeclear.com/react-practice-with-props-state-exercise-2/) we added internal state data (a list of contacts) to our app and set up our ContactList component to show a list of contacts.
* Now we want our app to be dynamic, and allow the user to change the state of the app.
* There are two buttons: one in the **Contact** component that deletes one contact, another in the **ContactList** component that deletes all the contacts. These buttons should change the internal state data which will trigger a re-render of the app.
* So, when it’s all completed, our app will work like this:

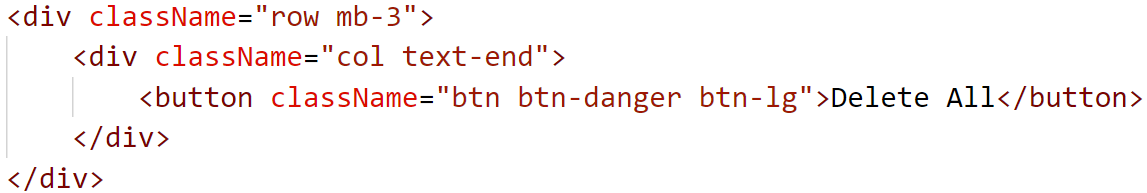


**Tips**

The Contact component is nearly identical to the one in [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/), with an addition. You can copy your code from Contact.js in the exercise-1-class folder, into Contact.js in the exercise-3-class folder. Then add in the new Delete button:



The ContactList component is nearly identical to the one in [Exercise 2](https://www.codingmadeclear.com/react-practice-with-props-state-exercise-2/), with an addition. You can copy your code from ContactList.js in the exercise-2-class folder, into ContactList.js in the exercise-3-class folder. Then add in the new Delete All button.



Because we now want to return two divs from our render function, and React only allows us to return one parent element, we need to use React.Fragment to wrap both divs, like this:



When we update our state, we should not modify the state directly. That means that if you are storing an array in state, you should not add or remove items directly from that array. You can use an array method that makes a copy (some examples are .filter() and .map() and .concat() and .slice() ), or make a copy yourself and then make the change to the copy. You then set the state to the changed copy.

In React, the component that stores the data in its state is the only one that can change the data. State is internal to a component and no other component can change it. Usually the component that stores the data in its state will have functions that modify the state in all the needed ways (like deleting something, creating something, or updating something). If any of its child components need to be able to be able to modify the data, it will pass the needed function down to that child component through props. Then the child component can call that function when it needs to.

## Exercise 3: Functional Components

* Expand the src folder, then the exercise-3-functional folder.
* In this folder there are three functional components, the App component, the Contact component, and the ContactList component. You can mix functional and class components, but for this exercise we’ll use all functional components to practice with them.
* There is also a data.js file that holds some static data that we will use. In a real app, we might pull our data from a server, but for this exercise we’ll just pull it from the data.js file.

**The Goal**

* In [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/) we set up our Contact component to show data about a contact.
* In [Exercise 2](https://www.codingmadeclear.com/react-practice-with-props-state-exercise-2/) we added internal state data (a list of contacts) to our app and set up our ContactList component to show a list of contacts.
* Now we want our app to be dynamic, and allow the user to change the state of the app. There are two buttons: one in the Contact component that deletes one contact, another in the ContactList component that deletes all the contacts. These buttons should change the internal state data which will trigger a re-render of the app.
* So, when it’s all completed, our app will work like this:

**Tips**

The Contact component is nearly identical to the one in [Exercise 1](https://www.codingmadeclear.com/react-practice-with-props-state/), with an addition. You can copy your code from Contact.js in the exercise-1-functional folder, into Contact.js in the exercise-3-functional folder. Then add in the new Delete button.

The ContactList component is nearly identical to the one in [Exercise 2](https://www.codingmadeclear.com/react-practice-with-props-state-exercise-2/), with an addition. You can copy your code from ContactList.js in the exercise-2-functional folder, into ContactList.js in the exercise-3-functional folder. Then add in the new Delete All button.

Because we now want to return two divs from our render function, and React only allows us to return one parent element, we need to use React.Fragment to wrap both divs.

When we update our state, we should not modify the state directly. That means that if you are storing an array in state, you should not add or remove items directly from that array. You can use an array method that makes a copy (some examples are .filter() and .map() and .concat() and .slice() ), or make a copy yourself and then make the change to the copy. You then set the state to the changed copy.

In React, the component that stores the data in its state is the only one that can change the data. State is internal to a component and no other component can change it. Usually the component that stores the data in its state will have functions that modify the state in all the needed ways (like deleting something, creating something, or updating something). If any of its child components need to be able to be able to modify the data, it will pass the needed function down to that child component through props. Then the child component can call that function when it needs to.