# Intro to Props Handout

## What are Props?

Props are a method of passing data from one component to another. This data can be of any data type, or even a function. Props are incredibly simple to use, and are great for functionality as the data can be changed dynamically, meaning we can re-render whenever necessary due to a change in the application.

## Example Uses

There are many cases where you may need to send data between components, for example if assembling a list on a page you will want to send different data to each list item, props are the easiest way to do this. You could do this for any page which uses API data as props would allow you to quickly and easily send the data to the individual components that require it.

## How do we use Props?

Take the following example, where the App component is rendering a message of hello world on the page:

import React, { Component } from 'react';

import './App.css';

class App extends Component {

message = "hello world";

render() {

return (

<div className="App">

<p>This is the app Component</p>

<p>the message is: {this.message}</p>

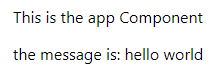
</div>

);

}

}

export default App;



This is fine and works well, but what if we want to use this message in other places? Or re-use the same html showing the message? In this case we will have to have this same message shown in a separate component.

To do this we will first define the other component:

import React, { Component } from 'react';

export class SubComponent extends Component {

render() {

return (

<div>

<p>This code is within SubComponent</p>

<p>The prop passed to SubComponent was: {this.props.message}</p>

</div>

)

}

}

This code will return HTML showing the messages ‘This code is within SubComponent’ and will then display the value of the Prop message that was passed to it.

If we then remove the paragraph tag in the App Component that displays the message, and add the SubComponent to the display you will see the following:

class App extends Component {

message = "hello world";

render() {

return (

<div className="App">

<p>This is the app Component</p>

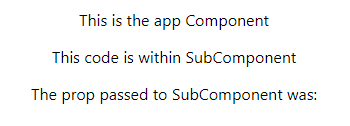
<SubComponent></SubComponent>

</div>

);

}

}



This is because we have not yet passed the Prop to SubComponent, it does not have access to the values defined with the App component, they must be specifically passed. We do this by simply changing the definition of SubComponent in the JSX as follows:

{/\* From this \*/}   
<SubComponent></SubComponent>

{/\* To This \*/}

<SubComponent message={this.message}></SubComponent>

This is gives SubComponent access to a new Prop called message, which has the same value as the message variable defined in the AppComponent. We can call this prop anything we like, it does not have to share the same name as the variable it takes its value from, as long as we use the same name when we use it in SubComponent the name used does not matter.

Here is the full code from the example:

**App.js:**

import React, { Component } from 'react';

export class SubComponent extends Component {

render() {

return (

<div>

<p>This code is within SubComponent</p>

<p>The prop passed to SubComponent was: {this.props.message}</p>

</div>

)

}

}

import React, { Component } from 'react';

import './App.css';

import { SubComponent } from './sub-components/SubComponent';

class App extends Component {

message = "hello world";

render() {

return (

<div className="App">

<p>This is the app Component</p>

<SubComponent message={this.message}></SubComponent>

</div>

);

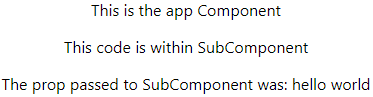
}

}

export default App;

**SubComponent.js**

**Result:**



## Passing functions with the use of Props

We can also use props to pass functions from one component to another. An example of this is shown below:

export class SubComponent extends Component {

componentDidMount() {

this.props.onMount();

}

render() {

return (

<div>

<p>This code is within SubComponent</p>

<p>The prop passed to SubComponent was: {this.props.message}</p>

</div>

)

}

}

class App extends Component {

message = "hello world";

render() {

return (

<div className="App">

<p>This is the app Component</p>

<SubComponent onMount={()=>this.logHello()} message={this.message}></SubComponent>

</div>

);

}

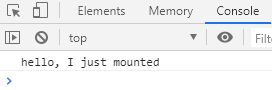
logHello() {

console.log('hello, I just mounted');

}

}

Now, when the SubComponent is created, it calls the function passed to it, when the console is opened after the page has finished loading you will see:



As you can see we used a fat arrow function to pass the function we wish to be called when the component is mounted, this is the more modern way to do this. However in the past we would write:

onMount={this.logHello}

This is because we would not put braces at the end of logHello, as this would call the function instantly, we instead leave them off to pass a reference to the function so that SubComponent can execute the function in the reference. This method still works, however we will cover why we now use fat arrow functions in the next section.

## Advanced passing functions using props

The functions passed into the SubComponent can also manipulate the data Stored in the app component. For example if we wanted to change the value of the message variable within the App Component we could do this by passing a function to change this value into SubComponent, and having SubComponent execute this for us. An example is shown below:

class App extends Component {

message = 'hello world';

render() {

return (

<div className="App">

<p>This is the app Component</p>

<SubComponent onChangeMessage={() => this.changeMessageValue()}

message={this.message}></SubComponent>

</div>

);

}

changeMessageValue() {

this.message = 'changed message value';

this.forceUpdate();

}

}

As you can see, in the App component we have defined a function which simply changes the value of the message variable, and then forces the component to update, which updates the view to show the new value on the screen.

After we have defined this function, we then pass it as a prop to the SubComponent using a fat arrow function, we have named this property onChangeMessage.

Many changes to variables and values after the page has loaded will come from things such as HTTP requests which can take time, to simulate this, and better demonstrate the change, in the SubComponent we have defined a setTimeout, so that the value of message is not updated until 2 seconds after the component has loaded. You will see this below:

As a result of this, when the page first loads, you will see the original value displayed as follows:

export class SubComponent extends Component {

componentDidMount() {

setTimeout(this.props.onChangeMessage, 2000);

}

render() {

return (

<div>

<p>This code is within SubComponent</p>

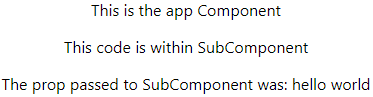
<p>The prop passed to SubComponent was: {this.props.message}</p>

</div>

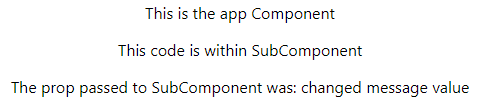
)

}

}



However after two seconds, this page will change to display:



Showing that we have successfully changed the value of a variable within the App component, from the SubComponent.

Earlier we mentioned that fat arrow functions are now the preferred way of passing functions in Props. This is because of JavaScript closures. When a function executes it must have access to all the variables used within it, when a function is passed between components, or used outside of the component it is defined in this may no longer be the case because it is being used in a different closure to where it was defined. There are two ways to avoid this happening, the older way is to use the .bind() function we showed earlier, this binds the function to the current enclosure, so that it has access to every value is it previously. The other simpler method, is simply to wrap the execution of the target function within a fat arrow function, which is quicker, simpler and has the same effect.