# Introduction to React

## Video Transcript

### Video 1 – Introduction to The JSX Render Element

Let's quickly review the React extension to JavaScript called JSX. So, if we end our filename in .jsx, it allows us to extend JavaScript to include statements like this. Here, I've defined an element const element =, and we have now < h1 > Hello < /h1 >. This is reminiscent of an HTML element or tag. We'd like to be able to render that and to do it, all we have to do is call ReactDOM.render, and specify what we want to render. Now, we have to tell it where we want to render it. And we do that by having a root div defined. So, like I'm showing here below, we have an id= "root". Now, in our standalone, that's defined in the standalone.html file.

And I'll show you that in a moment. So, now we have a way, if you like, of injecting into that root div our element. So, we've injected the < h1> Hello. Now, we can extend that by making it a variable Hello and some name. So, here I define name as being "Mary". And now, we're going to evaluate that name inside the element. And to do that, we enclose it in curly braces. So, now {name} refers to this. It's evaluated. So, now we'll have Hello Mary in that element tag, and that will be rendered into the root div. Now, when we program this way, React takes care of our elements or as we will see, the web components as well.

And it does it by running what we call a Shadow DOM. So, we're now rendering into the Shadow DOM, and React will take care of synchronizing the Shadow DOM with the real DOM. So, now we shouldn't be reaching into the Browser DOM, what I'm calling the real DOM. We should be programming in the React DOM. Okay, let's take a look quickly at some code. So, here we've got our standalone, and inside the standalone, we've got the ' "root" < /div >'. And here we're specifying the file, the jsx file where our program resides. And here is that program.

So, it's very short. But we've got our element. And as you know, this is neither a string nor pure HTML. But what will happen is that this in the curly braces will get evaluated and inserted into that element and then rendered. So, let's take a look at that in action. And so, here we have Hello Mary. If we bring up the IDE, we'll see that it will look like this. But inside "root", if you expand it, you will find the one that we've, the element that we specified, and the " Hello " "Mary". Okay, so this is the programming environment that we're going to use to explore React. And we'll do some more examples for you to practice.

### Video 2 – Introduction to JSX Components

So, let's review using JSX. And this time we're going to take our code and write a web component. And we're going to use the functional component form. So, we're going to call our component Hello. We need to start the component name with a capital letter. And when we finished it, we're

going to put it into the ReactDOM.render. And you'll see we've got a new tag here < Hello/>. So, the way we need to do this is we need to return whatever we want injected into that Hello. So, we're going to return < h1> Hello, and we're going to have the variable {name}. And in this case, we're going to make it 'Fred'. So, that is going to be then injected as before into the root. Okay, so let's go code that. So, here we are with our original element and now we want to develop, instead, we want to make this a component.

Okay? So, we say 'const', and now the name 'Hello =='. At the moment, we won't pass in any argument but we could. And now, that's our function. Now, what we need to do is we need to 'return'. And usually, we'll put whatever we're returning in, if it's multi-line, we'll need to put round brackets like this with a single line it probably doesn't make much difference. But we're going to take that '< h1 >', and we're going to return it. Okay? And we need to define the name. So, 'const name =', and let's make it ' "Fred" ' this time just so we can see the difference. So, that's our component. And if I save it, it's going to reformat it.

It took out the round brackets, but we need to remember to go and change this to '< Hello/ >'. That's our tag and we can end the tag like that. So, we save that. And let's render it. Hello Fred. Okay, so that's good. Let's just change this a little because we can define attributes of our tags. Let's put 'name =', for instance, here, as an attribute. And now we can pick that up here in, we'll call it properties '(props)'. And instead now of name being that, why don't we make it 'props.name;'. And ' "Francis" ' is being passed now as 'name'. So, it'll be an object and to get out of 'props' now we'll access that name. Okay, let's see this running. Okay, and we've got Francis. Let's take a look in the elements here and it looks very much the same.

## Video 3 – Passing Properties into A Component

Okay, let's take a look at passing in properties. So, when we define our component, our web component Hello, we can add attributes to this. So, for example, name = "Albert". Now, these get passed as the argument to Hello. So, we're calling it (props) because it's an argument, it's an arbitrary name. It's going to pick up the object that gets created here. So, the properties, all of the attributes are stored as objects. So, this will come as an object, and then we can access its elements by doing dot. Now, we need to know what these are called. So here, .name, for example. But then this will render out Hello Albert into that root. So, we will see our Hello Albert. Okay, let's take a look at coding. Let's do some coding here. And let's not only have the name passed, but we're going to pass a function in and we're going to call it 'action', arbitrary name.

But we're going to write it and it's called '{getRandomColor}'. And what we're going to do is pass, both of these, are going to get passed in to the 'props'. We're going to pick out the function, '{getRandomColor}' by it's called 'action'. And we're going to fire it. And we're going to get a random color. And we're going to put that into the style of our h1. And so, the 'backgroundColor:' is going to change. And then we're going to render out the name, which should render out '{props.name}' will be ' "Lisa" '. Okay. Let's take a look at the 'getRandomColor'. We're going to define an array and we're going to get a random number between 0 and 3 and we're going to round it down. So, 'Math.floor' takes that random number. We're multiplying by 3 because 'Math.random' gives a number between 0 and 1.

So now, we'll have a number between 0 and 3 and round it down. And we'll get a number 0, 1, or 2. And we're going to pick out of the 'palet' either the ' ["red", "green", "blue"];' green or blue 0, 1, or 2. And return that. We'll write out the color. And we're going to put that into the 'style'. So, we've got the curly braces that say execute this. And now inside here, we will get the color that is calculated by this 'props.action();'. Okay, let's see if it works. So, let's load it up. We get red, reload it. Red again. Red, green, red again, red is popular. Let's see if we get blue. Yep, there's blue. So, it's random. You'd think there's patterns but there's not.

Okay. So, that's passing into our component, some attributes that we're setting on that component. And we're picking them up in that argument. And so, we're picking them up. Let me collapse this. The important thing is we're picking it up here. We need to match. This will match whatever they're called down here. So, 'action' and 'name', So, '{props.name}' here, 'props.action();'. Okay, so we'll use this for passing functions into our components. Usually, from the parent component, we'll pass a function into a child. Okay, try this out, type this in yourself, and get it to execute. Maybe change these colors.

## Video 4 – ES6 and React

Hi! In this lesson, we're going to look at how ES6 is used in React. React makes heavy use of some of the things that we've covered in ES6. We're particularly going to look at array, and object destructuring. We're going to look at how the spread operator is used. We are going to look at callbacks at map, reduce, and filter because these are heavily used. So, we're going to give you some practice in these, and this is going to be very useful for you when it comes to some of the projects. So, let's enjoy using ES6 and in developing React components.

## Video 5 – Destructuring an Object

So, let's take a look at how ES6 is used in React. And in this case, I want to look at destructuring an object because that's often be used. So, suppose we've got an object. So, suppose it's something like we've got an object literal here called 'product'. And it's got a 'name:', a 'cost:', and an 'inStock:'. Let's suppose we want to just get at, let's suppose we want to print out its '{name,' and its 'cost}'. Then we could do it just like this. So, that would get us this part. Now, it doesn't have to be 'cost', we could to 'inStock'. The order doesn't matter. It's going to use the actual names. So, let's do this. And now, we can have that, put '=', say 'item', and we'll put '{name' and 'inStock};'. And let's see if that works. I'm going to take that here.

I'm going to put it into a web component, and I'm going to print it out here. I'm going to use 'item'. So, 'alert('item')' sorry, let's write ' 'name:' '. Now, I'm using backticks, so you remember we can put '$', and in this case, I put '{item.name}', okay and it also have 'instock: ${item.inStock}'. Okay, that's good. Let's save it. And I want to see if the destructuring works. That's my goal here. So, we'll click Click Me. And you'll see that, comes up name: is pear, instock: is 7 So, it worked as we wanted. Now, I want to actually, let me undo that. So, let's suppose now what I want to do is to have a child component. That's a button. So, I'm going have 'MyButton'. That's a component, the component and it needs to start with a capital letter.

All web components needs to start with capital letters to react convention. Unfortunately, you can type them with smaller letter and it won't complain. So, that's not good, and what we want to do is return In fact, we want to return the small 'button', this 'button' here. This is somewhat artificial, but I just want to do it as an exercise. So, now 'MyButton' is going to be a 'button'. It's not an 'onClick', but it doesn't have a 'handler', and that's the problem. Now, here I'm going to use 'MyButton' So, I'm going to return the web component '< MyButton >' and it's going to be this. Now, the problem is I need to move this 'handler' down into the child, to pass it into the child. Now, I can do it as a property on this web component.

I can make 'onClick' a property and I'm going to put it '={handler}' to the handler. Okay. So, 'MyButton' now has a property. Now, that property is passed in as the argument here '(props)'. That's just the convention. So, an object is created, and inside that object, the name of the object is '(props)' and inside that is going to be 'onClick'. So, now down here, I can do '{props.handler}', sorry '{props.onClick}', okay. So, let's see if that still works. So, now I click and it says hello. Good, it's working. So, this is now 'MyButton', and everything works. Now, we can use destructuring here, 'props'. There's something, so 'props.onClick'. So, 'onClick' is a property of 'props'. We can get at it by just using destructuring here.

Now, we need put braces around it, use the curly brackets, we could use it otherwise. So, I can save that. Let me get rid of that. And it's pulling 'onClick' out. So, now I don't need to do 'props.' I can just use 'onClick'. And it's going to get me the 'handler'. Let's check that it does. So, yes. I'm just going to put, just to check that all of them is being sent out. So, I'm going to do that. Reload, hello world. So, this is well working. So, that's showing you two things. One, destructuring of an object. And two, passing props between parent and the child.

And you can have any number of properties here. So, for example, we could have something like, so, let's suppose it's, we get index 9. We could pull out index here. And so, we could have. This would get us index as well well. We pass it down as part of props, and we could now retrieve it, but we're not using it. So, let's just pull out the handler. But it could be passed down and still working. Okay, so that's destructuring and also passing props between a parent and a child.

## Video 6 – Destructuring and Renaming an Object

So, let's continue with our examination of ES6 and react. We've seen how to destructure, props being passed in, destructure objects. Now, let's use that to get hold of the bootstrap '{Button}'. So, bootstrap has been loaded in the stand-alone. And I just want to get the button part of it. So, we to do 'let {Button} = ReactBootstrap;'. Now, I'm going to change that to 'B'. And let's reload. Beautiful. So, now we've got our 'ReactButton'. And it's still working. Now, let's do something that I don't recommend, but you can rename.

In destructuring, we could rename that button. So, there's a property in Bootstrap. So, that's why now we're getting that property. Now, here we can rename it. Let's just call it 'Abutton'. Now, I'm going to use 'Abutton' here. I've renamed it, it's pointing to the right thing. Let's see if it still works. So, this renames it, and now, we're using it. Yep, let's just check that I'm not having it crashed.

Yep, hello you. So, as I say, I don't think that I do this in the project, but that's renaming a destructured property.

## Video 7 – Map to Generate Child Buttons

So, let's see how ES6 can help us create lots of buttons, like here on the right. So, we saw how to create one button. Here it is. Now, let's see how we can alter this code to create lots of them. So, let's define a list, an array, 1, 2, 3. And we can loop over this list using ‘foreach’ as they said there. But I prefer to use map. So, we'll do 'a.map()' and that takes. We loop over each item. And then each item, we'll make a button. So, we take this. Delete it here, and put it here. So, now we've got a long list of buttons. We have to think about how they're going to be laid out. Let's put them. Well, we can do that later. That's okay. We've got a number of buttons here. So, 'list' now is going to be a lot of these. I'll get rid of that since we don't want it.

So, we're going to return this and we're going to do that multiple times. And we're going to get our list of things. Okay, that seems reasonable. Now, what do we want to 'return' here? This is returning from 'App'. Well, we want to return at least that '{list}'. But we have to enclose it in something. We can just put it in a frag. So, that's just those angle brackets like that. And that should do. Now, I know it's going to complain, but let's see what we have here. So, we have to reload. Yeah, we've got four buttons. Okay. So, that's quite nice. If I look over at the console, it's warning me that each child should have a key. What it's saying is, I'm putting in a lot of buttons over here.

So, that mean I'm going to put this on the next line so you can see it. I'll put it in. Here we go. Okay. Now, it's complaining that we need a key here. So, we can put the 'key='. Well, we could get it, let's see. We can use item or probably preferable 'map' can take two arguments. So, let's put in the second argument. The second argument is and 'index'. It takes three arguments. Actually, we can actually pass this array in as well if we wanted it. But the 'index' tells us how many items we been through. So, the 'key' we could put as '{index}'. We could make it a string, which it should be, but this will do it.

This will get rid of that problem. So, let's save that. Let me reload. And I'm looking to open my console. And now, I see I've got a clean window. It's compiled correctly. Now, you might wonder why I don't put the map in here, in the 'return' statement. I could do that. They're not allowed to put for loops in the 'return' statement of a web component. But we could put the map in there. But I prefer to leave the map here outside of the return statement because I think this makes things clearer and easier to read. Okay, so now let's take a look at how we might catch the click event and perhaps, for example, delete a button.

## Video 8 – Using Props Index to Identify Buttons

So, we've seen how to use map on an array to create a number of buttons. And we returned a 'Mybutton' with an 'onClick'. And 'Mybutton' is a web component we've written here and we're passing the '({ onclick })' event in. And that's '{handler}' which we've defined up here that says '(`Hello`);'. And so, when we click this button, we should see Hello. Let's just check. Let's click this one. Yep. It says Hello. Let's click this one. Hello. Okay. Now, let's suppose we want to know which button has been clicked. Whether that it's this one, the first one, second one, third one, or fourth one. Okay. So, let's take a look how we might do that.

Well, the 'onClick' always creates an event and we can catch that event in our handler. So, we can catch the event '(e)'. Let's just call it '(e)'. We can call it whatever we like. And let's print out something about '(e)', I'll call this '(`button:', and then, I'm going to use the backtick notation. And inside here, I can look at the event '${e.target'. I happen to know that there's a '.getattributeNames()}`)' function. So, let's run that and see what we're getting. Okay, let's load that up, and let's click. So, the names of the attributes type, and class. So, that confirms that which button has just got type, and class. Well, this button has 'key' as well. So, it's not that button, this one only has 'onClick'. So, it has type and class. Let's add an index to this. How can we do that?

Well, we have to do it up here. So, to 'Mybutton', if I had an 'index =' and I'll make it '{index}'. So, that's this index in the map. So, we have '(item,' and 'index)', so we're looping through. So, this will get the values 0, 1, 2, 3. Now, how do I pick them up down here? Well, I need to put a comma and put ',index' in here. And now, I've got 'index'. I can put it in this button as well. Okay. So, now when I click up here, we should see different attributes. Let's take a look. And let's reload. Yes, index, type, class. So, now we're picking up this index here. So, we can take a look at that. Instead of 'getAttributeName()', we can do 'getAttribute'. And in here we can put '("index")'. Okay, let's see if that does what we want. Reload, Click Me, button: 1, button: 0. Perfect. So, that's one way to get at which button has been clicked.

## Video 9 – Using Closure to Generate ID Functions

So, our problem is to try and figure out which button has been clicked. And we saw there was a good solution if we added 'index' to the attributes of the button, passed it down in the properties. Then we could specify it here. And our 'onClick' then could take the event and could use '${e.target.getAttribute("index")}`);'. And that would be the index of the button that was clicked. Now, let's see if we can find a more elegant solution. So, one option would be to pass an argument here to the 'handler'. If we could pass '(index)' to the 'handler', then instead of having to deal with the event here, we actually could pick up that '(index)'.

Then we could just say the '(`button:' was '${index}`);'. Now, we can't do that because 'onClick', the function that you passed to 'onClick' can't take an argument. It doesn't allow it. But maybe there's a workaround what if we created an anonymous function here, and the anonymous function would just call our function. And 'index' is known to that function. Here it is. It's an

argument. So, here. And what will happen is this function. We'll use closure to set the '(index)' so that the first button will be passed to functions with index 0.

The second button would be passed to function with index 1, etc. So, this is a perfect example of closure. And now, we don't need to pass index. We don't need this index. So, we don't need this. Delete that. And we don't need to pass it down here. So, we don't need that. We don't need to specify it here. So, we are locking into this 'onClick', an anonymous function, which we're picking up here, and it knows which button it is. So, this is beautiful. Let's see if this works. So, let's reload the page and we'll hit Click Me button: 0, button:1, 2. I think this is one of the most elegant solutions I've seen and it's a perfect use for closure.

## Video 10 – Delete Button Exercise

So, in this exercise, I want you to destroy buttons when they're clicked on. We've spent some time figuring out how to know which button has been clicked on. So, now it's your time to actually do something with that information. And now when I click on one of them, it goes away. And you'll see now I've got [1, 3, 4] left. Let me click on this one which should be 4. Yes. Now, I've got [1, 3] left. Let me destroy 1. I've done that, and I finally got [3], and I destroyed 3. So, this is what I want you to do. That's the exercise. I think you will need to use filter. I want you to keep state. So, let me show you that now.

So, I want you to use, '{ useState }', and to do the following we'll 'let [state,' and 'setState]', where they'll be equal to 'useState()' with the initial setup. So, what I want is to put the numbers of the buttons in there. So, that's going to be your initial state. Now, every time you destroy the button. In here, this is where you'll be able to destroy the button and you update the 'state'. So, what you'll need to do is instead of 'alert' in here, you'll need to put some code. So, in here '// code to update the state and remove destroyed button'. So, what you'll need to do is probably, the hint would be to filter out.

We know this 'index' here, is the button you need to destroy. So, out of this, let's suppose it's 2. Then we need to end up with a state that looks like '// [1,3,4]'. So, the state will get updated and you'll need to 'setState' to whatever the '(newState)' is. Okay. So, and then, of course, it's going to be re-rendered. And to make sure that it's re-rendered, we'll need to make this 'state'. Okay, so that's, copy this code. This is your starting configuration. You've got to work in here. You may have to change some things elsewhere, but you should be able to do it. Okay, so that's the exercise. Copy this code.

## Video 11 – Delete Button Solution

So, our goal is to have these buttons destroyed when they're clicked on. So, we'll need to know which button is clicked on and which one to destroy. Now, we're keeping track of 'state' here and obviously, using '{ useState }'. Now notice '{ useState }' is using destructuring. Let me show you. So, here's the code. Let's put a breakpoint here and let me reload that code and you see we've broken here now at this point. Now, let me show you React;. So, if I click on React;, so now we can list the children, and the one that we're after is 'useState:', here it is. And we're picking out that one attribute out of all of these attributes of this object. Notice the colons?

So, you know, and you can see that 'useState' is some kind of function. Okay. So, that's destructuring. Now, let's continue to solve our problem. So, we know we're going to get 'index', which will be the index of the button. Now, what index will that be? Well, it's going to be returned to us by whatever this 'handler' function has. Let's change this a little to keep track of the number rather than the index position. So, here I'm going to put '(item)'. So, this is now going to be the button number. It's going to be '[1, 2, 3, 4]'. The index would be 0, 1, 2, 3. Okay. So now, I know what I'm getting back here is actually the button number that I need to destroy. Let's figure out how to get rid of that.

So, let's suppose it's button number 2. We'd have 'state.filter' and this will give us the '(item)'. So, this will give us the actual number that we want to get rid of. So, if that number is equal to 'index', we want to get rid of it. So, we want everything left. If it's not equal to 'index'. That seems like it should do it. Now, I need to make sure this is a. Right, that's better and we need to end it with, okay. We need just that. Okay. So, that now will filter out, and let's call this 'newState' and that now will be set. And we should be rendering it out. Let me, I'm going to put a 'console.log' in here so we can see if we're doing it correctly. And let me do '(newState)', okay. And now, that'll remember React?

When it sees that something is being updated in useState, it will re-render. So, this now will automatically call this and we've got our 'newState', and that will be used to re-render. Let's see if that does exactly what we want. So, let me reload the code and I seem to have I'm checking the code here just to see that I've got my updated code. It looks good. Let's click on the buttons and see. So, okay. That one worked. You can see here we've got buttons [2, 3, 4] left. Let me click on 3 and now, you see, we've got [2, 4] left. Let me click on 2 which will be this button. Okay, we've got button [4] left. Let's destroy ourselves totally. Everyone's gone. So, that is quite nice. We've got control of everything. We've managed to destroy all our buttons. We actually only needed to write one line of code. Beautiful!

## Optional

### Introduction to ES6 Exercises

So, in this exercise, we're going to be looking at using JavaScript and some of the new syntax that ES6 has introduced. We really love ES6 because it allows you to write very compact code and is very powerful in the extensions it's provided. So, let's go and warm up like an athlete we need to train. And these exercises are going to get us going with ES6 and JavaScript.

### ES6 Warm-Up - Shopping Cart

So, in this ES6 warm-up, we're going to look at some data on "fruits":. And you'll see that this data has an object. And inside that, there's a property "fruits";, which is an array of all of these. So, I'm asking you to write your own fruit. So, choose a different one to this and fill in all the properties, change the price to whatever you like. Now, add your fruit to the array of fruits. So, and put it at the beginning. So, that it's going to be right at the head here of this array. It's not difficult. And you should be able to do that. Now, I'm assuming that I've gone shopping and I've put various fruits into my shopping cart, and here are the ids of those fruits. 0, 3, 4, 5, and I put 3 in again. Now, what I want you to do is to write a function to calculate how much I'll pay at checkout.

So, here's my 'cart'. Now, I suggest you write 'getCostOf' of one of the ids. So, this will be say 0 for 3 or 4. And now, fill in here what that cost will be. So, I have to pick out the price of say, item 0. Now, I suggest you write a function 'getTotal' total that takes the 'cart' and tells us what the total is of all the items in the cart. So, it gives us the price that we have to pay at checkout. Now, this is a little trickier. I want you to write a function to fill the cart up. But I've only got 1000 to spend. And we're going to choose and we're going to choose items at random. So, we need to write a random function. Here. I suggest you call it random and write your round.

So, write something called 'random', say. Let's put a capital. Let's call it 'getRandom' So, that would be equal to, and you'll need to scale it So, you can pass in a '(scale)' because we need to return numbers between 0 and however many fruits we have, and I think it's seven. So, you need to figure out this function. You will use it in 'fillCart'. So, you've got to fill it up so that we don't spend more than our total of 1000. So, we keep choosing items to put in the cart and at the end, we can return the cart. I believe we should pass the cart in.

So, why don't we do that and we probably should start it off as empty? And then we'd know how much we have to pay. And it should be a number below 1000. Okay. So, to execute that, we should be able to type in 'fillCart', and pass in the '(cart)', say. And if we're passing in the cart, we should make sure it's empty. We need to pass in if we don't pass the argument. This says, there's the default argument. Okay. So, I hope you enjoy this and you should learn quite a lot. It requires that you're thinking exactly by what you're doing. You might have to look things up, but that's, there's plenty of documentation on ES6, and examples.

## ES6 Warm-Up - Shopping Cart Solution

So, let's take another look at ES6 JavaScript and continue our warm-up. This time, I gave you 'data' and it's on ' "fruits":'. So, 'data' itself is an object, and then ' "fruits":' is a property in that object and it contains an array. So, 'data.fruits' is an array with these entries. Now, we want to make our own entry. So, we've done that. And now, we need to create a bigger array with our entry as the first item in it. So, we want to go to 'data.fruits'. That's the array. And we want to add to it. Well, we want to replace it actually with a new array with our fruit at the front. And now, we can spread out 'data.fruits' the old. All right, so what's happening now is, this is now going to point to something totally different.

We've got a handle on this one. But now it's going to point to somewhere different with our new array. And it's '(7)' long and we can check that it contains ours, ours is the ' "plum" '. Yep, it looks good. So, we've got that. Now, we want to create a cart and find out the price of items in the cart. So, we can create a 'cart' easily enough. It's an array, and these will be the ids of the items in this array. Okay. So, we've got that. Now, we want to create, we'd like to get the cost of one of these items for each of them eventually. But first, let's write. I get cost of, and given an id, we want to return the cost of an object. So, 'getCostOf', we're going to pass in an 'id'.

And we're going to go to the data. Now, we, let's suppose we, I don't want to type 'data.fruits' all the time. So, before we do that, let me, so let me put 'data1 = data.fruits;' So, I don't have to keep retyping 'data.fruits;'. Okay, now let's do the 'getCostOf'. So, let's, I seem to have typed this before and 'getCostOf' by passing an 'id'. Now, we'll do 'data1.filter'. And 'filter' will return to us the 'item'. So, to return to us one of these fruits, if the 'item.id' matches the 'id' we're passing in. So, we doing that check. Now, filter, remember returns an array. So, we need to get the '[0]' element to that array to get hold of one of these fruits. The fruit that matches the id.

Now, we've got hold of that. We can get its 'price'. And we need to put brackets around here. That'll match up this bracket, yep. Okay, so that's 'getCostOf'. Now, let's get the total in the cart. So, given this cart, let's get the total. So, the 'getTotal', we pass in the cart. We set the 'total = 0;'. Now, we're doing a 'map' on those integers in the cart. They're in an array. So, we can do 'cart.map'. We've got the 'item', and now we get the cost of that item. So, we just wrote that, and add it to the total, and return that total. And that should give us the cost of all the items in the cart. Let's try. And we pass in the '(cart)'. We define the cart.

Yeah, it gives us '660'. And if you want to check, that's the right amount. Now, we need to be able to fill a cart so that we're always spending less than a 1,000. So, we're going to fill the cart with random items. So, let's get a random function. We're going to pass in a 'scale' because 'Math.random()' creates a number between 0 and 1. And we want a number between 0, and there are 7 items here. So, we'll pass in the 'scale' of 7. And that'll give us a number between 0 and 7, and then we'll round down so that we will always have a number, 0, 1, 2, 3, 4, 5, 6. And that matches what we have in our data. So, let's 'getRandom'. We can check 'getRandom'. Let's do it with '(2)', so we can just check it out. It'll basically give '0' and '1'.

So, let's just check out the '(7)'. Makes sure we're okay with the '(7)'. So, the first one was '2', '1', '6', '5'. It looks like it's working. That's pretty nice. Okay, so now, let's see if we can fill the cart up. So, we need to write a function to fill up the cart so that we always have less than a $1,000. Let's take a look at it. 'fillCart'. We're passing in the cart, we set the 'total = 0'. I think we might want to have a default of '0'. Sorry, the default would be, let's put a default of an empty cart, in case, we forget to pass a cart in. Now, it's going to go around this 'while' loop. While the total is less than a thousand, we're going to get a random item from, of those fruits.

So, that'll be our 'nextItem'. We see if the '(total+getCostOf(nextItem))' is within a budget. 'i<1000'. If so, we 'push(nextItem)' into the 'cart', and we update the 'total' so that will either jump out of this loop next time if it's over the total. And this is go, we'll see the total as we go along. So, let's do that. Let's put our 'cart = []'. 'fillCart'. And we need to pass in '(cart)'. Okay? And it fills it up to '940'. Let's check again, 'fillCart'. 973. 924. 887. Okay. It looks like it's doing a really good job. So, we've had pretty good time with this one. It's quite tricky.

I'm sure it'll exercise you, so, or rather that it has exercised you since you'll be watching this after you've done the exercise. But hopefully, you've learned a lot about ES6. Watch out with filter. Remember it brings back an array, often mentally, I'm thinking, oh, I'm filtering for one object. And I'm thinking that I've got that object, that it's buried in that array. So, we have to be careful with that to go and dig it out of the array, which we're doing up here with that '[0]' there. So, I really hope you've enjoyed these exercises and that I'm sure they'll help you with the challenges of react.

## ES6 Warm-Up Code Interpretation

So, in this exercise, I want you to work through each of these statements and give the answer as to what they're doing. Okay, so this is just for you to understand for yourself. And then I'll give you the answers to these statements.

## ES6 Warm-Up Code Interpretation Solution

So, let's go through the exercise I gave you. So, the first one, we had 'A' as '[1, 2, 3]'. So, that's easy. That's just an array. And now, we did 'B = A.unshift'. And we put a '(0)' in there. And what this does is it inserts at the beginning of 'A', whatever this is. So, now it puts a '0' into the 'A'. So, we'll see the 'A' now has a '[0, 1, 2, 3]'. But it returns to be the length now of 'A'. So, 'B' is the length of the new array. So, this 'unshift' updates the array. Okay, let's take a look at 'B' now being '= [0,'. Now, we have the spread operator, 'A'. So, it spreads 'A' out. And now, we added a '4'. So, remember what B, what A is now. A is of length '4' now. Now, I'm replacing B that is equal to '4', at the moment, with this.

So, B now is an array, and it's going to have two '0's at the beginning. '[0, 0, 1, 2, 3,' and then we added. So, '0, 1, 2, 3', which is the spread operator because we'd already done the 'unshift'. And

then we have '4' at the end. Then let's look at some functions. So, we had 'f1=' and then we have no arguments '()'. Now, we have the '=>', so we know it's a function. Normally, we'd put brackets, and we'd put maybe '1'. That's what I want to do is 'return 1;'. So, that would be fine. But equally as good is if we're just going to return what's on the right-hand side. In one line, we don't need those brackets. And we don't need the 'return'.

So, we can write it just like that. That may be a little confusing. But if we do 'f1', you'll see its value is '1'. So, it returns '1', okay. And now, we did some others. 'f2 =', and in this case, they return '2'. 'const f3 =', and we're going to return '3'. So, if I 'f3', it gives us the value '3'. So, those are fat arrows for functions, and it makes it easy to pass functions around. Let's take a look now at 'B = [f1, f2, f3]. So, now we've got an array of references to functions. We've got 'f1' is a function, 'f2' is a function and 'f3' is a function. Suppose we say, 'let [a, b] = B'. So, what is a? Well, it's a function, and we fire it and see what the result is. '1'.

So, a is 'f1'. So, this pulls out the first two elements of the array. And we fire 'b()', we'll get '2'. Okay. So, there's a little bit of destructuring there, of that array that we're taking out of a three-element array, a two-element. Okay, let's take a look now. What do you think? What are we going to get with this? We'll write 'a()'. So, we know what that is. '+ 1', and now we'll say, '== b()' or fire 'b'. Suppose we didn't fire 'b', just did that. What would this be? Well, go to 'a', and that's 'f1', and so that returns '1', '+ 1' is '2'. And then we're saying, '2 =' and 'b'. Well, 'b' is a function now, 'f2'. So, that'll be 'false'. But if we do and fire 'b()'. What do you think now? Well, we should get 'true', and we do.

Let's do well on destructuring. So, let's suppose we've got, I've got some variables, 'color = red, ', and I've got 'width = 200,'. And length or 'height = 300'. Okay, so those are all defined. Now, suppose I say, 'let props = {color,width,height'. What am I doing? Well, this is shorthand now, for I'm writing really this. I'm creating an object. So, you recognize that. But now with ES6, we don't need to specify the names. It'll know the key. Here, the key is 'color', ' 'red' ' is the value. So, color will be red there. So, let's go back to our original, and we just do '{color, width, height}'. So, we've got that 'props'. 'onClick' a function equals, and let's pass into it. We could pass into it '(props)'. Okay?

And we could then write out that who the, this we could do 'console.log', say, '(prompts.width)'. That would be fine. Another way we could do this. We call it with 'props'. We could just say, I only want width from whatever you're calling width. I want you to destructure whatever you're passing into me. So, let's fire 'onClick'. And we pass in '(props)', which contains 'width'. So, out of it, it picks up the 'width', and prints it out. And it's '200'. So, if we. So, this is really destructuring 'props'. You don't have to keep repeating 'props', for example, in the function. Let's do 'const onMove =', so we pass in the '(props)'.

And now, we might have something like '(props.color, props.width, props.height)' So, we'd have to repeat them, and that is a little tedious. Of course, this would work and this would be fine. But let's pick out '{color}'. And now, let's print at that. So, instead of 'props.color', we can just now, print out '{color}'. And we could, ' 'color is', and then '${color}' '. That's quite nice. Good. That's cool. 'onMove' So, we call 'onMove', and pass in '(props)'. And it should just pick out the 'color'.