Module Outcomes

After completing this module, you will be able to:

Use JavaScript for loops for repeatedly performing actions
Achieve advanced loop control with break and continue in JavaScript
Perform advanced manipulation of arrays in JavaScript
Understand and control the DOM structure
Extend your ability to handle mouse events, as well as timer events
Use advanced function handling techniques

1. For Loops

Let's have a look at for loops in JavaScript. So that's the focus of this presentation, there's several different types of for loop. We're gonna look at them and they look like this.

So, for loops, well, for loops, the clearly show the start and end values all in one line. It's a little bit different to a while loop. A while loop, perhaps you might have something at the top, something at the bottom. And it might possibly be a little bit less clear when it finishes, how it starts, that kind of thing. With a for loop, they're all in one centered place. So there are some advantages to using this for loop.

For loops are especially good for handling a series of data, for example in an array.

Now in this next example on the next slide, we use this little piece of code so I'm just gonna quickly remind us what it does. And basically if you have some data structure such an array and you want to know how long it is, how many items are in the array, you just say name of the data structure.length. And that tells you how many things are in it. So here's our first example.

So what I have here is an array, and that has five things inside it. That array has a list of continents in the world, and then we have a loop and that loop is going to ask the person have you been to each continent. One by one, the person using the web page will be asked, have you been to Australia, Africa, Antarctica, Eurasia, America in a loop.

Now we're focused on a for loop. So let's look at our for loop. And basically, there's three parts. One part there, and another part there and another part there and the first part is setting up the initial values. We're going to use a variable called index. We could use any name for the variable and then that's getting started. And then when will the loop keep repeating? It will keep repeating while that variable index is less than the total length of that array. So, we're going through all of the elements in the array. And so we wanna continue until we get to the end.

Now, every time at the end of the loop, what are we gonna do? We're gonna increase the value of that variable by one, index++.

So we have a loop. And it will keep repeating and it will repeat until this is not true any more. And that is when it will stop.

Okay, so in the loop, we use confirm("Have you been to") and then we pick out that particular continent, and it says, have you been to Australia, get the result. The result will be true or false, depending on what they do when they see the little window. And then we can check. If the response is

true, we could just increase a simple count for counting how many continents they have been to. That's not really related to the for loop. It's just an idea of using a loop for a particular purpose, okay. This is what we're focused on, we're focused on our for loop. So there's our basic for loop. And then at the end, we can show a message, you have been to. And then we can show how many continents you have been to.

All right, so let's run this example. So let's run this example.

First question. Have you been to Australia? And I could say, yes, I've been to Australia. So I could do OK. And if I haven't been, I press the other one, Cancel, which is basically sending a no message. But let me say, yes I have.

Have you been to Africa, all right. Well I'm gonna say I haven't been to Africa. So I'm gonna do the second button. Have you been to Antarctica? Well, let's pretend that I have. I'll say OK. Have you been to Eurasia? Well, perhaps I haven't. I say Cancel. Have I been to America? Okay, I'll just do OK, and then there's a result being shown. You have been to three continents? So it's a fairly simple loop. But just illustrating the for loop. Let's go back to our presentation. All right this is what we just did. We had a loop, and it did those one, two, three, four, five continents, taking them out of the array. And then I pressed something like, no, yes, no, yes, yes something like that. And then it does a little summary you have been to how many continents you have clicked OK.

All right so that was a typical for loop right there now let's look at a couple more variations of for loop. First one is for in loop. So, this is taking the previous example and then rewriting it so we use for...in. Now straightaway you notice that the code looks nicer, right? The for loop doesn't have so much kind of complicated stuff. It doesn't have plus plus, it doesn't have any semicolons, it looks nicer and it's easier to type. And it's easier to manage as well cuz you're less likely to make mistakes. So this is our for in loop. Much nicer.

Everything else is the same. We have the same array up here, we have the same loop content, we have the same message at the end, you have been to three continents. All of the content is exactly the same.

Alright, so you can see straight away what's happening. This will, by itself, take the array list continents which is this array. And then it will do the counting and it will put the number in here automatically. So it will put zero in here, which is this one, and it will put one in here for this, two in here, three in here, four in here, then it will stop. So it automatically has the index number, zero, one, two, three, four.

That's great. That's a much nicer code.

Now let's look at another example. This is going through not an array but it's going through an object.

So here's an object and it's got a series of left hand sides and right hand sides. If you wanna use that language. Left, right, left, right and so on. That's our object. And we're gonna use a for in. Same thing we just looked at. And we go through that object and automatically that for in structure gives the left hand side to this variable. So it's not an index anymore. It's just the left hand side. All right, so initials will go here and then you could use it in the loop. Age will go here, property, and then you could use it in the loop. Job will go here and then you can use it in the loop.

So, in this particular loop, we use the left hand side, initials, age, job, but also, we use that and we can

access the right hand side as well. Okay? So this is taking that variable and then using it so we can access the right hand side. In other words, we can access everything. It looks like this, so we've got left-hand side, right-hand side, left, right, left, right. Initials = DR, Age = 40, Job = Professor, and that's the end. So, pretty straight forward.

Let's look at the final type of, for, for of. All right? Now for of is is slightly different. It gives you each individual item.

So we've taken that example which we've used before. And then we've used for of, for of. Now, again we don't have to have all those semi-colons and plus plus and all of those things that we used in our very first example, so that's nicer.

But there's a slight difference to the for in. So what's happening now is it goes through the array, there's our array, same as before, go through the array and take the actual items. Take Australia and put it in this variable. Next loop it does take Africa, and put it in this variable. Next one it does Antarctica, next one Eurasia, next one America. And the actual values, these words, are given to the variable continent. That's different from the index number. It's the actual values going into this variable.

So now I can slightly adjust my code and I can say okay, please have you been to this continent. And I can simply use the word continent, which is this variable here, because it has the words Australia and then Africa and then Antarctica and so on, and that is what is contained in that variable. So I can simplify my code here, and also, compared to the first for example, I can simplify this code as well.

All right, now, a few more things you can do with for loops cuz they're quite flexible. So one thing you can do is not include some of these pieces here. So what happens if we don't include any of them, right? There's a empty space, semicolon, empty space, semicolon, empty space. That is an infinite loop. JavaScript says, well, you haven't told me when I should I keep doing the loop. Remember that's the second part. You haven't told me that. So, I'm gonna keep doing this loop forever. It will never stop. So this message, a little window, it will keep coming up in a loop forever. Infinite loop.

Alright, now, sometimes people do that. Sometimes they leave out just part of the structure, the for loop structure. So, here we're leaving out the first part. Now, remember the first part is setting up the initial values. So, instead, I just chose to do it outside the for loop up here. It's just personal preference, right. So I'm not doing here, I'm doing it outside the loop here but I've also, I've got the usual test when do I keep doing my loop. And also what do I do at the end of the loop, those two parts are included in this example. I've just removed this part and I put it outside.

Okay, you can do that sometimes people do that. Sometimes people do this. So this one is, let's say you have two variables being used in your loop, which keep getting updated during the loop process. Not just one variable, but two. Well what you can do is set up both of those variables in the first part, like this. There's a comma there, it's not a semicolon. So I'm setting up variable one. I'm setting up variable two in the first part.

Then the second part, like usual I need some test. When do I keep repeating my loop? That's kind of the same as before. And then the third part, what do i do at the end of each iteration of the loop? I update two variables and I can do that. I've just got the first variable being updated, increase by one and then comma, not semicolon. The second variable which gets multiplied by two, which gets doubled.

So that's an idea of handling two variables. And then in my loop itself, of course, it will actually use

those variables in some way.

All right, so that's the end of our discussion about for loops.

2. Loop Control

Let's look at some other ways to control loops in JavaScript.

So, these are the two commands we're gonna look at, break and continue.

So break, basically, totally stops the loop. This could be a while loop, do while loop, for loop. Whatever the loop, if you say break, then your loop will completely stop, permanently okay? The other side of that is continue. It's not quite the same. That one just stops the current iteration. Remember we used the word iteration, when we have a loop, okay? So let's have a look. This one is using break. It's a pretty simple example. Let's run it. So it asks you, okay, you have lots of bank accounts with different types of different amounts of money. So how much is in this one? And then you keep entering your money, and it will tell you the total. Then how much is in this account? I don't know, maybe like $100. Press Enter. Okay, how much in this account? Okay, another account, maybe $90, something like that. And you keep going and it will repeat and you keep entering your sums of money. And eventually you just don't have any more money, no more bank accounts, so you just simply enter nothing. You press OK. And then it gives you the total. Your total amount is, and then it adds up all of those amounts, $270. So let's see how this example works. Let's go back to our presentation. So here's the code for that example. And we have a loop, and that loop keeps repeating forever. It says while (true). There's no variable here. It just says, while (true). Now that is one way to do a loop that never ends. So we have a loop that never, never ends. So how can we end it? Well, the only way for this particular structure, is to use break.

Break will completely stop the loop, and jump out of the loop, and carry on with JavaScript code after the loop.

All right, so in this particular example, somebody types in a number and then we convert it to a float just before we start adding it. And then we start adding it to another variable, total amount. Whatever they type in, whatever number, it keeps getting added. So we do +=, to keep adding on top of the previous total. Now that's not very important for this example. What we're really looking at is this one here, break. So, if what they type in, the number they type in, is a positive number, then we just keep adding. Otherwise, then they haven't entered a proper number, bigger than zero. So that's the signal to stop the loop. So we simply stop the loop. Infinite loop never stops. Let's stop it with a break instruction, and then it jumps underneath, and it carries on with the JavaScript. All right, so that's an example of break, and this is illustrating what we just saw. Type in a number. Type in another number. Type in nothing, so don't type anything, and just do okay, and then it comes up with the total.

All right, continue. So continue skips the current iteration. We talk about iteration, right? A loop that keeps going round and round, each one of those is called an iteration. Continue just stops the current iteration and jumps to the next iteration. It doesn't stop the loop permanently.

The next example, we use this language, array.push, so I just quickly explain it here. And that is push will add an item to the end of the array, okay? That's what array.push is, not related to loops or such, just a line of code in this example.

So here, what we are doing, an interesting kind of loop, let's go through it, okay, so it says okay, 2014, was great for you? And you can choose was it great or and if you can give a positive response or a negative response and you can say, okay, yes, that was great. 2015, was that a great year for you? Or well, you can say no, wasn't that good maybe. Cancel. 2016 was it was great for you? Oh yeah, maybe it was great. And then it shows a summary of your great years, the ones where you said okay.

Straightforward. Let's go back to our presentation. So this was illustrating a loop and for this particular example we used the for loop. Start with the year 2014. Go up to the year 2016. Keep adding one to the year and at each time, show the message, was that year great for you? Get the result. It's either gonna be true or false. If the result is not true, cuz that's what's happening here. If not correct. Correct is the result coming back from that little window, and it either is true or false. If it's not true, then skip this iteration of the loop and jump straightaway to the next iteration of the loop. That is what this says here.

So, in other words, if this was not a good year, then don't do this line of code. Instead, jump straightaway to the next iteration to carry on with the next year. However, if that was a great year, then let's make a list of great years. Here we are, this little list of great years, a little array. And start with empty array and then let's add that year to that array list. So we build up a list of great years. That's the idea. All right, so that's the example of continue. It doesn't permanently stop the loop, it just tells the browser to go to the next loop. Okay? The next iteration.

All right, so here's an illustration, 2014, OK, 2015, Cancel. It didn't like that year, 2016, good year. And then it shows a summary, 2014, 2016, and all of that was done using continue. So, that's the end of our discussion of break and continue.

3. More On Arrays

Let's look at some more things you can do with arrays in JavaScript. That's what we're focused on. We've called it advanced array functions. But they're not very hard. These are the different functions we're gonna look at.

So, first thing, sorting. There's an example array. Dog, Cat, Rabbit, Hamster. If we simply say take that array, and then sort it. Then, in this particular example, it will give us a result in alphabetical order. Cat, dog, hamster, rabbit. So that's C, D, H, R. Alphabetical order. If these are numbers, that will work as well. It will do increasing number order.

Reverse. You can guess what it does. Name of the array, .reverse. It's gonna swap the order from the back to the front. The first element will become the last element. The last element will become the first element. And it will swap them all around. So Dog, Cat, Rabbit, Hamster is going to become Hamster, Rabbit, Cat, Dog. Okay? It just does a simple reverse order.

In fact, you can combine these together. All of these functions, you can combine them together. So, for example, you could say take the array and then sort it. And then, when you're finish sorting it. Reverse it and it will correctly do those both operations to the array.

Here's the result rabbit, hamster, dog, cat. So we took the dog, cat and then it got sorted in to alphabetical order. And then it got reversed. So we have reverse alphabetical order.

All right, one way to find something in an array is to use indexOf.

So in this array, search for whatever you are searching for. So we've got our animals here, like before. Search inside the array. Look for one particular example, rabbit. Okay? Search for rabbit.

And, this will give us the first place where it finds rabbit. It won't give you the second or the third. It just gives you the first place that it finds rabbit. Okay, in this particular example, it's gonna be not here, not here, here. So it's gonna print the number two. Because this particular thing returns the value two. Zero, one, two.

If it can't find it, it returns a value of minus 1, meaning an error.

All right, what else could we do? We could do same thing indexOf, but we could start at a particular position. There's no reason to start at the beginning and go to the end. Search for this, but start at a particular position depending on what you want to do.

So this example here has lots of animals in an array.

And then we are searching for rabbits, okay. For some particular reason, you're searching for rabbits. And we're using push which adds the position of the rabbit. Every time it finds a rabbit it adds the position.

And then it keeps looking. So we have a loop. It has a loop. It has a loop, until we don't find any more. Okay?

So we are using the index. It will find rabbit. And we have lots of rabbits. Rabbit here, here, here, here. We have our code. It goes through each one of these. It starts looking inside that array. It looks for rabbit, and the start position keeps being updated. So that's the clever use of this second parameter. We keep updating the start position. So we search for rabbits. We found it. And then, we remember where we found it. And then, we start searching again, but it would be a waste of time to start searching from the start. That would be silly. So instead, we start searching from the next one. And then, we keep searching, oh there's a rabbit, ok. Then we start searching from the nest position. There's a rabbit. And then, we start searching from that position. Each time we search from the next position. That's what happening here. And so that illustrates the use of the second parameter. At the end, if we did show us where all the rabbit positions are, we're gonna to get 2, 4, 5, 10. And it shows us, there's a rabbit at two and four and five and ten. Remember, the indexing starts from zero. All right, so we just saw indexOf. There's also lastindexOf and that will search from the end going backwards.

So going backwards from the end, search for rabbits. All right, not here, here. There's rabbit. It will give us the number 0,1,2,3. It will give us the number 3 in this example, because it started from the end. And it will ignore this one. Let's look at slice. So slice is extracting part of an array, really a series of items in an array. So let's look an example. Array name, .slice(1). So that says go to position 1. And then from that point onwards, if there's only one parameter here, from that point onwards extract all of those items. Cat, rabbit, hamster, and it extracted them. And we can put them into a new array name. The original data is not altered. Okay, the pet array, it has not changed at all. But we have copied part of the series of that array.

So here item one. Dog, cat, rabbit, hamster right up to the end is put into our new array. Now, we could add a second parameter just like this. And that is, the end position does not get included. So let's go to item one, which is here. And then do item two, as well. And item three, we will not include it. So it does item one, item two. Cat, rabbit. Those are included in the result. Let's look at splice. So splice is

different to slice. Okay? It's got a p in there, splice. And that is when you are effecting the original array.

So as to that, dog, cat, rabbit, hamster. Let's go to position one. Here it is. And then let's extract one single item, that's quantity. And so we are extracting cat. Take it out. And that has affected the original array. That original array now has three items, not four.

Okay, so, the thing that we have extracted is cat. That goes into result. The remaining array just has three things, dog, rabbit, hamster.

All right, let's do splice a little bit more, more control. So, let's go to, for example, position two like this. And then, we'll say for one particular, or let's take all these parameter for now. And let's do an addition of the word rabbit. Okay? So for position 2, 0, 1, 2,. We're gonna add the word rabbits. And that's gonna be the new position 2. That's gonna be the new position two, which is here.

As a result we're going to have dog, cat, rabbit and hamster. This is insertion. Okay? Dog, cat, new position 2, rabbit, and the old position 2 is now position 3, hamster. Okay, what is this operation return? Well, it doesn't actually return anything. It's doing an insertion. So it doesn't actually return anything, cuz it didn't grab anything out of the array.

Okay, let's use splice again, a bit more complex example. Let's go to position one, which is this one here.

And then it's going to do a replace of one item. So cat is gonna be replaced, and what is it going to be replaced with? It's gonna be replaced with rabbit and fish.

So, we're gonna get dog, and then replacing rabbit, fish, and then hamster, okay? That will affect the original array. So now, we're gonna get dog, and then rabbit, fish, and then hamster by doing a replacement at position one of one of those items.

All right, so that's the second set of examples about array handling in JavaScript.

4. Array Functions

Let's have a look at some more advanced array operations in JavaScript.

So that's what we've focused on, advanced array operations and there's two particular commands we're gonna look at. There's actually many commands for manipulating arrays in JavaScript but these are the two we're gonna look at. So, this is the traditional way of going through every item in an array, right? There's a typical array, dog, cat, hamster, stored in an array. And you have a for loop, start at the first item, item zero, and then keep going until we get to the end, each time go to the next one. That's a typical for loop right there. That will give us access to each individual element in the array like pets 0 dog, pets 1 cat, pets 2 hamster. And then here we're just giving it to an alert, and alert will show a little window, and it will show those things. Dog and then cat and then hamster, one by one. Okay, that's a typical use of a for loop. That's a lot of typing, you could make mistakes. You can forget the semicolon here, here. You could forget one of the pluses, forget the close brackets, whatever. There's loads of ways to make a mistake with a for loop.

Now, what you can do is this. This will do exactly the same result.

Name of the array.forEach, and that says go through each one and do this little function here, which is the same alert. Just a simple function, just to do something, show it in a window.

Okay, so that is our for each. It has the same result as this, but much easier to handle. Less chance of making mistakes as long as you remember there's a capital E here.

So, what is it doing, that forEach? Well, we can understand it, but actually it has a bit more power than what you think. That forEach, you don't have to type this, all right. This is not for typing. This is just to illustrate the way for forEach works. Okay, so we've got the array which we're working on and then we've got a function which is applied to each item one by one. And then we have our for loop which goes through each of those items. And then, it applies that function there. We just saw it using array alert, okay. So there's alert and it is taking dog and then cat and then hamster and it's displaying it. That's what we just saw.

However, there is actually a couple of extra parameters which that function, if it's clever, could also use, it could use those extra things. Now alert, which we use it's not clever, it's not gonna use this extra things to do something clever, it's just gonna ignore them. But it is given them, what are they? Well, that is the index number by zero, one, two, and that is the entire array. Those three things together are given to the function every time when you use forEach, okay? That's the way forEach works. This is illustrating the way forEach works. Now, if you use alert, it doesn't help you. But if you make this function yourself, then this could help you. You have everything and the index and the actual dog, cat and so on. So your function, if you make your own function, could do a lot of things. So your function could be down here. You're given the actual element, like dog, cat, you're given the index. And in fact, you're given the entire array, every single time and all you have to do is use those if you want to use them. If you don't wanna use them, okay, ignore them.

So here's an example, there's our array, one, two, three, four, five and then we say, all right, go through the array. For every item, forEach, I want to apply this function. And I actually write the function right there, right there I write the function. No need to do it somewhere above. What does the function do? Well, this function is taking the first thing, which is the number like 1, 2, 3, 4. That is 1, then it is 2, then it is 3, then it is 4, then it is 5.

And also it has those extra items. And here, just to illustrate the idea, it's using those two extra items. Alert is not clever enough to do that, it's using these items here. So what does it do? It takes the item, like 1, and then it multiplies it to get the square. And then it puts it back in the same place by using those second and third parameters, which is quite clever. In other words, it's directly affecting the original array and it's replacing those elements with their square.

And then, when it's finished, we can check the result, and the result shows lots of square numbers, 1, 4, 9, 16, 25. They're all the squares of the original array.

Okay, so let's look at map, so that example we just looked at, it changed the original array. Map is quite similar, perhaps, but it doesn't change the original array, okay.

So let's have a look at an example. This is how map works. So don't think that this, you have to type this stuff. No, it's just illustrating how map works. So what does it do? It runs that function. It runs the function, one by one, to each thing in the array, same idea, one by one, dog, cat, hamster, whatever. It runs a function to those elements, one by one, there's our for loop. But when it's got the answer, it runs

some function and then it gives it those parameters and it gets an answer. It adds the answer using push. And it adds the answer on the end. Every time you do push, it adds on the end of an array, okay? So, it's basically going through every element and then, it applies that function, gets the answer and adds the function to an array. And that is a new array. It is not interfering with the original array. And then finally, it can return the results.

So here we have it, another example. There's some array elements. And we got a little function up here, and that function takes a number and returns a square of the number.

And so, what are we doing? We're saying, oh, let's take that array, and then let's do a map.

And that is using a function, which is up here. Square. Okay, square. All it does is returns the square. If you give it a 9, it will return 9 times 9. So, what we do is we use map and it will go through everyone of those elements and it works out the square. And it returns a completely new list, a new array, and it gets stored in there. And then we can see the results. And it hasn't interfered with the original array.

If we ran this, this alert would show us 1, 4, 9, 16, 25, which is just the squares of this. The original array would not be altered. Okay. So, that's the end of our

further discussion on how to manipulate arrays.

5. The DOM - Basic Concepts

So now let's look at the DOM- Basic Concepts.

So after this presentation, you'll appreciate the concept of a DOM and how it gets used. You'll understand the role of text nodes in a DOM. And you'll appreciate the concept of whitespace and how it is stored.

So DOM, what does it mean? Document Object Model. When you load something into a browser, it gets converted into this DOM structure. Whatever you load. You load HTML, it gets converted.

XML, if you were doing that will get converted. SVG, all of those things get converted into this DOM structure.

Now we're focusing mainly on HTML.

Let's see an example. This is some code we've already seen. It was our first web page. Very simple. Header section, body section. Couple of things, couple of things here. Pretty straight forward HTML. If you load that into a browser, it gets displayed as this. Okay, but we're not really focusing on how it gets displayed now. We're focusing on how it gets stored. Anyway, that's how it gets displayed. We've seen that before.

In the memory, it gets stored in a certain way. And this is how it gets stored. As a what we call a tree structure. In computer science, a tree structure is basically start from something and then split and then split and maybe do some more split. And it's like an upside down tree. So sometimes we call this kind of thing, a tree structure.

Now, so I've got the idea of that webpage stored in memory.

And it's stored in this kind of graph structure, a tree structure called a DOM. So the top level, we have our HTML. Inside that, I'm sure you remember, was a head section and a body section. Both of those were under the HTML. And then inside the head section, we had a couple of things. So those go under that part of the DOM structure. In the body section, we had a couple of things as well. Now there's a couple of keywords, so let's just explain those keywords. I use the word node a lot. N-O-D-E, node. All of these things here, all of these are nodes. And that language is just from discussing graphs, okay? I use that a lot, the word node and we have a top node. That top node is called the root, so sometimes I use the word root. It simply means the top, if you like, the top of the graph. Okay, that's the root.

What else do we have? We have some language where I use the word child. So for example, this particular node here is a parent of two child nodes, okay? So we use the language parent, we use the language child. All right? This particular node here is a child of this node here.

We also use the language sibling, kind of like brother, sister, okay. But there's no gender concept, so we use the word sibling.

This particular node here is a sibling of this one, meta. They are on the same line, if you wanna use that language. And so they are siblings, okay? So this kind of language is very commonly used when we talk about this. And then finally, we have this language branch. And so that again, that's related to this language of trees, we have the language branch. And that means kind of area of this graph structure. Where you have one thing at the top and then it probably splits off into other things. We call that a branch, okay. You can think of it as an upside down branch, just like the whole thing is an upside down tree.

Okay, so special language branch, node, child, parent, all of these kinds of things are very common when we talk about the DOM structure.

Okay, so here we have the DOM structure which is representing that webpage we just saw.

Okay, let's look at another DOM structure just to get more familiar with what we are talking about. So here's another webpage. And there is a table in this webpage. And it has a table body which sometimes people don't write, but let's write table body. Inside the table body is a row.

There's another row. And we've talked about tables before. And inside the row is a box and another box. And the second row has a box and another box. And so in your mind, you can build up the structure. All right, what does it look like? Okay, it looks like this. Okay, very simple. There's one column, there's another column. But we're not really talking about what does it look like. We're talking about the DOM structure. So we load that table into memory. This is the DOM structure. We have table node. Inside there is table body node, okay. Inside that was one row and another row. Inside that was two boxes, td, table data. Inside the other row was two boxes. Inside those boxes were pieces of text, okay. Each box had a little piece of text.

Now in the DOM structure, that piece of text is its own node. It is a separate node underneath the thing which contains the text.

So this box has this text, this box has this text and this box has this text and so on. These are text nodes, okay? They are text nodes. You don't actually type that when you type your HTML, but in memory they are represented as separate nodes.

Let's look at another example. So we've got a little bit more detail now, a bit more complex. But really, the main thing we've added is just some attributes. Okay, so we have an attribute here, we have an attribute here. The body has an id, the paragraph, the first paragraph has an id and then what do we have? We have a piece of text, that text is not in any paragraph. It's not in anything really. And then we have a break. And then we have another paragraph. That paragraph has an id attribute as well.

Now these attributes are useful when we start doing things with JavaScript, but we'll talk about that later. We're just saying, what happens when we load this particular webpage into memory? So all of these attributes, they don't get thrown away, they get preserved, okay. So, what does it look like? Okay, we just saw paragraph, text, paragraph, nothing very special. That's what it looks like. The DOM structure is what we're focusing on. So we have our body element, let's focus on that, downwards. That one had an attribute, id is the body in this case. It could be any name, nothing special about this name. What was inside there? Well, we had a paragraph. We had a piece of text, which wasn't inside anything, okay? And you can see that visually. We have a paragraph, that was the first thing. That paragraph has an attribute, id like name. And then we had a piece of text which was not inside anything. It wasn't in the paragraph, it was just by itself. So that is a separate node by itself. And then we had a break, okay. That doesn't have any text in it. That is what we called a void element, we talked about that before. So it's just by itself. And then we have our last paragraph and that one also had an attribute. That paragraph had some text inside it. The very first paragraph has some text inside it, okay. So that is the DOM structure for that example here, okay. Basically, that's what happens. Now, we've got some familiarity with the DOM structure. Loading a page, we know what happens. There are some surprises sometimes. So I wanna talk about one surprise and that is this, this thing called whitespace. And sometimes, people say whitespace nodes.

So what is whitespace? Whitespace basically, is anything you can't see like spacing or an instruction to go to the next line. If you have an instruction which says go to the next line, then that is something you can't see. That is also whitespacing.

Now, you may have a text node, right. We just saw examples of the text nodes. And those text nodes may have only whitespace, all right. In other words, they don't contain anything that you can actually see, right? They just have some kind of maybe tabs or spacing or an instruction to go to the next line, something like that, right?

Those text nodes are called whitespace nodes.

Sometimes they are a little bit of trouble, because programmers who are accessing the DOM. Doing something clever with the DOM, they don't think about these whitespace nodes and there code doesn't work. So you have to be aware of them.

So here's an example. , body, this is the start of the code. Body, inside the body is a paragraph with some text. No problem at all. Very simple. That does not have a whitespace node. Here, it does not. What about this one? Okay, body and then next line, paragraph, bit of text inside it. That has a white space node. All right, it has a text node right here after the body. Before the p, it has a text node which contains nothing visible. But it contains a message which says, go to the next line.

All right so, that is one, two, three, actually four nodes there. That is four nodes. This one is one, two, three nodes, all right? Three nodes, four nodes. Because of the white space.

So, here's an example and we've got our body which we were just talking about and then we've got a message to go to the next line and then we have a paragraph. What happens in the DOM structure? Well, it looks like this, okay? Nothing too surprising. In the DOM structure, we have our body and then the very first thing that we just saw was actually a text node containing whitespace.

People if they're not used to this, they may think, oh no the very first thing was a paragraph. And yes, the paragraph had a text node, we have a message inside it. That's what people might think. But you have to be careful. The first thing is actually in that example, a text node with some emptiness in there. An instruction to go to the next line. All right, so that's the end of our introduction to DOM.

6. Node Relationships

So now let's continue our understanding of the DOM structure by looking at node relations.

So after this presentation, you'll understand the relationship between nodes and how you access that in code. You'll be able to visualize the path to a node. And we'll talk a little bit about using event handlers as well. So we're gonna look at these ways of accessing the relations between different nodes.

So, basically we have our DOM structure. Remember? Anything loaded into the browser memory gets converted to a DOM structure. How are all these nodes related to one another? So we can use specific code which let's you explore that relationship. And this applies to many different programming languages. Don't think this is just JavaScript. Perhaps, you're doing PHP or Java. All of these code that we're gonna see could be used in those languages, all right? So these are the keywords which can work in multiple languages. And lets look at that visually, cuz that's the best way to understand it.

So this is our DOM structure. And what we have is the same things we've seen before. Body, there's a paragraph with some text. There's a piece of text by itself, there's a break element. There's a paragraph with some text in it. Okay, typical kind of DOM that we've seen before. But what we have now, is all the relations written in this graph to help us understand.

For example, if we say that is our top element right here, how can we use some code, which might be JavaScript or any other language, to refer to this element, starting with this element. Well, that's what's shown here. So from here to here, what is the relationship? It is the first child.

It is also childNodes. So from there to there, that is childNodes of this, all right. What about this one here? How can I refer to this one from here? Well, that one is childNodes. Okay, so you're gonna get the pattern, pretty obvious. This to this, is childNodes. This to this, is childNodes and so on for every child.

However, this particular DOM has this paragraph as the lastChild. So from here to here, I can also say that is the lastChild for this particular DOM structure.

So we've got firstChild, we've got lastChild, we've got childNodes, childNodes, childNodes, childNodes and so on.

What else do we have? For some siblings, which are on the same row, if you wanna use the language row. Then we have the relationship, Sibling. So from here to here, this is the nextSibling from this element. And here to here, this one is the nextSibling of this element. Here to here, nextSibling. We also have previousSibling. So if I wanna go from here to here, I can say go follow the previousSibling

link.

What else do we have? Well, we have the parentNode relation. So from here to here, if I'm starting here, I wanna find the parent. Then I can say from here, give me the parentNode and it will give me this. Same idea here. From here to here, give me the parentNode and it will give me this.

So those are the main languages for talking about the relations between all the different nodes.

Now let's say we wanna find something in the DOM structure, which is very common. We wanna find a particular node in the DOM structure.

So we can do that and we have a little example here. And what we do in this example is we click on a node and then it shows the path to that node.

So how do we set that up, right? What we wanna do is to click on a node. And we're gonna see the path to that node. We're gonna show it in a text format. In fact, let me quickly show an example right here. Then we'll come back to the code. And so here's an example. What we have here is just lots of HTML, all right. Random HTML. It's got some lists, lists inside lists. And we've got some anchors, remember anchors for links, things like that. This is just some random HTML. When I click on one of these items, it will run a JavaScript function. That JavaScript function will go up the DOM structure and tell me all of the elements that lead to the item I click on. So let's show that running HTML. I'm gonna click on something.

And there's the result.

Okay, it shows me that the thing I clicked on down here was starting from the end, an anchor, in other words, a link. It was an anchor which was inside a list item. Which was inside an ordered list. Which was inside a list item, which was inside an unordered list, which was inside the body of the webpage, which was inside HTML. Which was inside the document. All of that shows you the path which brings me to this particular element, all right. Anchor, list item, ordered list, list item, unordered list, body, HTML, document. Now if I click on something else, I get the same idea. I get a message to show me where in the DOM structure this is. So let me click on this item. What does it say? It says, the thing I clicked on was an anchor inside a list item, inside an unordered list, inside a body, inside a HTML, inside the document. So all of these things are just basically things to play with. Things to click on to give you an idea of the path to that element.

For example, if I go down the bottom, there's even more things to play with. So I could have some items here, I could click on this input item, click. And it gives me a long sequence, which tells me the thing I clicked on was an input element inside table data, inside table row, inside table body. Inside table, which was inside a form, which was inside another table. And that table was inside the body. And that was inside the document. And so that is very deep down the DOM structure, the thing that I clicked on here.

So it's just illustrating the path of things that gets you to a realworld example. In this case, it was clicking on one of these input items. So that was created by using JavaScript. So let's have a look at that JavaScript.

So here is the JavaScript which created that message. It was a little piece of text. That's all that we saw. Piece of text. Right at the end, we have an alert message. That alert message is just showing us the text.

What is the text? Well, we click on something and then we have a loop. That loop starts from the place where we clicked. We clicked on a particular node, right? We did clicked on a particular node. And then we have a loop. And we keep going up the DOM structure. Up, up, up, up, up until we do not have any parent. Where is it that we don't have a parent? It is the root node, right at the top, that's the only one which doesn't have a parent. So we have a while loop. While there is a parent, while there is a parent that exists, that's what this is saying.

Then go to the parent and add this current node. Add the name such as list item, td, table, input, whatever type it is. Add the name on the left-hand side, add the little arrow. And add the thing that we have so far on the right-hand side. So it basically on the left-hand side, we keep building up the name of the node as we go up the DOM structure. And this is just a piece of text. That's a piece of text starts with the thing that we clicked on, the node. Start with the name of the thing we clicked on, such as input. And then, start adding an arrow and then start adding the parent. And then go up to the parent and then add the next parent. Go up to the parent, add the next parent and we end up with a long piece of text, which gives us the path from the node right up to the top of the tree. And then we can just show that piece of text.

Now there's perhaps some code that we're not totally familiar with, but we don't have to understand everything. The main clever part is this one here, the loop which keeps going up to the parent.

This here is event.stopPropagation. Now that is a little instruction which just makes sure that this code is only executed once and it's not executed every time. So for example, let's say I click on an input element in a table. I don't want the input element to trigger the function and the table also to trigger the function. I don't want that. So I say, make sure it only happens once by stopping the event after its been handled once.

Okay. So, some clever code there.

Let's move on.

So how do we trigger that code? So what we do is, after we've constructed that function we saw, we add that function to almost every element in the DOM structure.

Now we ran an example just now which was HTML. And it shows you the path to the node that we clicked on. There's another example, a few seconds later, SVG which is exactly the same code and exactly the same idea. Simply to help understand the point that we're making, it's not related to HTML. The DOM structure and the JavaScript code for manipulating DOM structure is really the same regardless of what's being stored.

So how did we add that to every element? Well, we have our DOM structure. And again, here's some code which is a little bit kind of clever. So don't worry too much if you can't understand all of it. But the task here is to add that little function that we already saw, which is called handleClick. And make sure whenever I click on something, it will run that function that we just saw. So this code here goes through every element in the DOM structure. Doesn't matter if it's HTML or SVG or XML, doesn't matter. Goes through every single element and in a clever way, it says whenever someone clicks on this element, make sure we run that function. Now I won't go to this in detail, but it's a clever function which actually calls itself to make sure that it goes through every element in the DOM structure. Don't worry if some details escape you. It's not super important.

So we saw this example, that was our HTML example and it produces something like this. We saw that. Here's another example, SVG example. And again, same thing. I've got something in the DOM structure. And when I click on something, it will go up, up, up the parent until it reaches the root. So let's try that. So in my SVG example, I'm gonna click on something like, let's click on a star. And it says, all right, you clicked on a polygon. Remember, this is not HTML, this is SVG, the graphics language. Polygon, you clicked on the polygon, the polygon was inside a g.

G means group. So the polygon was in a group of stars. That group was in another group, which was maybe the background group. And that group was in another group, which is perhaps the webpage group. And that group was inside the SVG area which was inside the document, which means the webpage. So same idea, basically I clicked on something from that node, it goes to the parent, parent, parent, until there is no more parent. Which means you are at the top of the DOM structure. That's the idea here. Another language, but exactly the same code.

So let's just move on to the last slide or two. So there's an example where we just did the same idea. This time, I clicked on a rectangle, which is inside some groups. And it went up the DOM structure.

What else do we have? That's really the end, I think, of my second discussion about DOM relationships.

7. Locating Nodes

So now let's look at locating nodes in the DOM structure.

So basically, after this presentation, you'll appreciate how to find a node, how to locate it. And also, how to change something in the node.

These are the three instructions we're gonna look at.

So we know the DOM, everything gets stored in the DOM structure.

We can actually add anything in the DOM structure, we can delete anything in the DOM structure, copy, we can change anything in the DOM structure. Basically if you’re a programmer and you know how to do it, you can do anything you like with things in that DOM structure.

First thing that we have to do, is we have to understand how to locate something before we can delete it, change it, whatever we want to do.

So, there's three methods for locating something in a DOM structure. First method is, use the exact DOM path.

So you go right from the start, from the root, and you go down, down, down. And you say, on this particular node here, after giving the exact path to get to that node.

That's one way to find something in the DOM structure. Not the best way, really, sometimes hard to work out the exact path to that node. And if you make a little mistake then it totally doesn't work.

Also, it is possible, although it's a bit rare these days, it's possible you build some code and it works in one browser and then you try the code in another browser and it doesn't work. And the reason for that,

is because the DOM structure may be slightly different, especially at the very top level. Usually not an issue these days, but it's an issue with older browsers.

All right, what is the other possible ways of locating something in the DOM structure. Well, another way is getElementsByTagName(). That's really saying, find something in the DOM structure by using the type.

So for example, let's say you have a h2, okay? Get elements by tag name h2, and then the JavaScript code will find the h2,. Or maybe h3, okay, get elements by tag name h3, we'll find the h3. However, there maybe five h3's or seven h2's. And so you actually need to know which particular h1 or h2 or h3 you want to find.

So you have to actually say, with some clever tricks, which one you want to refer to. The first one, the second one, the third one. So, there's a little bit of extra work with this one.

This one is probably the simplest one.

Find something using the id of that element.

So when we write our HTML, we say element_name and then we give it an id, as you know. And then we finish the element_name. And then there's probably something inside. So, that is typical way of constructing some HTML, we simply search for whatever that id is. getElement in the DOM structure by searching for the Id. That's what happens. And in many ways, this is the easiest way to search for one particular thing in the DOM structure.

So, here we have an example and it uses all three methods.

So, we're gonna construct an example, how do we construct an example to experiment with this? Well, here's some HTML code. We could have a little input button, one, two, three input buttons. You click on one button. It runs one piece of code. You click on another button, it runs another piece of code. You click on the third button. It runs a third piece of code. Where is the code? It's on the next slide. And there's our first function, second function, third function. That is using the first method we talked about. That is using the second method, the third method, and so the first method was, the exact path through the DOM structure. So start from the top level, go to the first child. childNode, meaning the first child, go to the third child, childNode, meaning the third child. Underneath there go to the second child. Inside that, we want to change the color to red. This is a kind of random thing, doesn't really matter what we want to do. The main thing is, how are we locating something in the DOM structure? So that is following the exact path.

Second method was using the TagName. The element name. So here I'm searching for the h2 in the webpage. I'm searching for the first h2.

0 means the first one.

First h2, change the color to yellow. Doesn't really matter what we do with it. The third type was using the Id.

So, search for cute_text. That was the Id. And then change the color of that.

So let's go back to the HTML to remind ourselves, h2 and then cute_text and that was down here, so there's our h2 inside our webpage body slash body. H2 and it had the name cute text so that's our third example. It searches for something with this name, and it changes it. Our second example, we're searching for h2, and then it wanted to change the very first h2. Now, there isn't a second h2, there isn't a third h2, but still you have to say, find h2. And I want to refer to the first h2 in this example, and I want to change it.

All right so there's some code, webpage and three functions, which get triggered when I click on these buttons. So let me now demonstrate that.

So here is that example, nice and big. So we're gonna run the first method, which is exact DOM path, and then the second method, which is TagName, search for h2, and then the third method, which was use the Id.

It won't look very spectacular, but it's enough to prove that the code works. So the first method was change the color, and you can see it's changed into red color using the exact DOM path.

Second method was searching for h2 and changing the first h2 we find. Yes, it's successfully changed into yellow color.

Third method was search using the IED and change it to blue color. So you can see all three methods totally work fine, the exact DOM path, the TagName, and also the ElementId, so all of these are working fine.

So we just saw an example and everything was working fine, we were changing the color of something we had located. Another way to change something after you've located it is setAttribute. So, I just wanna mention that, after you find something, and this is the third method, getElementById, search for something with a name. Then after you found it, you can change something using setAttribute. Very common way to change things, using setAttribute. So here I'm changing something, I'm changing the style. There's lots of things that you can change, in "style" here, I'm changing the color, very similar to the last example. So that goes in the style property. Immediately in the webpage, you will see it change. So, it's more common to use this method of changing things setAttribute(). And that's the end of our discussion about locating things in the DOM.

8. Creating and Adding Nodes

So now let's look at creating and adding nodes to DOM structure.

So after this presentation, you'll be able to create different types of node. And then add them at the appropriate place in the DOM structure.

So these are the commands we're going to look at. Let's get started.

So the basic step is first, you create some nodes, or possibly one node. And when you create them, they are not yet in the web page. They're just floating around in part of the browser memory. They are not yet in the proper DOM structure. Then, step two, you add them at the appropriate place.

So, how do we create things? Well, the first method is create element. So you could make a div in this example, or a paragraph. You put p in there. H1 to get heading one, whatever you want to do. You can

use create element. But there are some things that, remember, you can't type such as a text node. You can't really type that. So, there's some special commands for handling those, createTextNode. Similarly, there's createComment and other things like that. So, you could create something that you can type or use one of these special commands to create something special such as a text node.

So, you create it and then you have to add it into the DOM structure at the appropriate place.

In this example, we create a hr. Remember, hr makes a kind of horizontal ruler, a kind of line across the page. And then we add it to the DOM structure at the appropriate place.

In this example, we search for the body. We search for the main content of the web page, the body. We search for it using tag name, just to give another example of using tag name. So, get elements by tag name, body. Now, body, there is only one body of course. So you have to say, give me the first body. And how do you say first? Like many programming languages, you use 0. So give me the first body and that gives us the parent. So the bod is the parent in this example. And then, I say at the position of the parent, firstChild. So, in other words, the start of the webpage. Rght at the top of the webpage, I will insert before the parents first child. I will insert this new thing that I have created which is a line.

So, let's look at that example.

So here is that example, so let's click on the web page. And yes, you can see it has added an hr, the line.

I could do it again, and it runs it again. And adds the hr at this top of the body. In other words, the first child. Each time I click, it runs the code. And it adds a first child, which is the hr. All right, let's continue with our notes.

So, let's think what we just did just to be extra clear. We had our body element, and then we searched for it. Remember, there's different ways for searching. For an element, we use to get elements by tag name, just as one of the ways to do it. We found it. And then before we did anything, it already had some text. That text was actually h1. Okay, so that's what it looks like in the DOM structure, some text in the h1. Then, we ran some JavaScript, and the JavaScript did an insert. It did an insert before the first child. That was the first child. Then, we've added this in the document memory. And then, we can see it immediately in the web page. If I run the code again, it will add another one over here, hr. If I run the code again, it will add another one over here. Every time it's adding it as the first child.

All right, that's the whole code that we just saw. Web page, body. And you click on the body. It runs the code, insert new text. There is our code, insert new text. First thing we did, creat an hr, the line. And then we added it as the first child of the body using the insert before command. So if you're inserting something before the first child, well, that means it's gonna be the new first child. And then we saw that at the top of the web page in front of this h1. So that was our first example.

Let's show another way of adding something to the DOM structure, and that is appendChild( ). So here we have create a text node, another example. Some text here. And then, we search for something using the name, using the ID. We search for something called my texts. And then, that thing, we append the child. And that is the text node that we have created. So whatever that is, could be paragraph, whatever. We have added some new text at the end of that item.

So here's our example. Let's run it.

So I'm gonna click on the page, and then it will run the code. And it has added the child to this element. If I run it again. It again, appends adds a child at the end of this element. This element could be a paragraph, a heading, a div. It doesn't really matter what it is. The code would be the same. Okay? Each time I click, it's adding a child at the end. Let's continue with our presentation.

So let's think about what we just saw from the DOM point of view. Before we added anything, we had this webpage. We're actually using an h1 element. It wasn't a paragraph. It wasn't a div. It was actually h1. So before we added anything, this is what we had. We had an h1, and it already had some text. That text was, please click on this web page. That was an h1. Then, we ran some JavaScript and it added an element. It created and added another text node within that h1. Because it doesn't have to have just one text node. It could have two, three, four, as many as you like. And they all get displayed by the browser. So my code added this. If I run the code another time. It will add another node and another node and another node. And I would keep adding more nodes over to the right. So here's the whole code. web page, body. When I click on that, it runs my function, JavaScript function. It creates a text node. And then, it adds it at the appropriate place, which is at the end of something called my text. In this example, my text was an h1. And so, inside there, it was adding some text right at the end of that h1.

So, that's the end of our discussion about creating and adding nodes in the DOM structure.

9. Deleting Nodes

So now let's look at deleting nodes from the DOM structure.

So after this presentation, that's what you'll understand, how to delete nodes from the DOM.

And that is all related to this command, removeChild.

So, let's look at an example.

First we find a particular element that we want to delete, a particular node in the dome structure. So here we're searching for something with the name myPara. It doesn't matter what the name is. Search for something, and then we have to ask the parent of that child to delete it.

So, you have to say from the thing that you just found, go up to the parent. Ask the parent to remove the child. Now, that thing may have lots of children. Right? It may have five or eight different nodes under it. So, you have to say remove which child. So it's quite kind of annoying, a bit troublesome to remove a node from the DOM structure.

So, again, you find the thing that you want to remove and then you go up from that thing to the parent, and you say, please parent, remove that child. And you have to say exactly which child. So it's quite annoying, quite troublesome.

So here's three example pieces of code, which are also on the next slide, for deleting one element in this particular DOM structure.

There's a body element and it's got a paragraph. That's the first thing in the web page. That paragraph has some text, and then there's some more text and so on. The complete DOM is not shown. Our target is to delete this paragraph, just as an example. So what we have is three sets of code, which will delete that paragraph.

So then we zoom in on those code. So one way to do it is, find the name of that node. And then, as we just illustrated, ask the parent to remove that child, remove that particular child. That works, it'll disappear. Another way is find the thing that you want to make disappear using the other methods we've talked about like tag name. I'm trying to throw away the first paragraph. Okay so find all the paragraphs and give me the very first paragraph. Zero means the first one. So, that gives me the very first paragraph in the webpage and then I do the same thing. From that item go to the parent, ask the parent to remove the child, this particular child.

That would also delete that first paragraph. Here's another example, this one a little bit different. This one we search for the body. So that's going straight to the parent.

Search for the body, that's the parent to the top and then we can ask the parent straightaway, we can say okay parent, please remove your child. Which child? Well it's this child and you have to be very precise and you have to point to the exact child you want.

So that is another way to delete the first paragraph in a web page. So different ways to delete things and run our first example, hello has disappeared. That was our first paragraph. Let's reload the page. And then when our second one, hello has disappeared. That was the first paragraph using slightly different code. And then here's our third one. And again, hello has disappeared. And that was using our code which goes straight to the parent, okay. And again, to reset, I just reload the webpage. Back to our presentation.

So here's the code that we were just running in our example. That was a body and that was our webpage, and we just had three buttons. Those buttons were just running different functions. First function, second function, third function. This is the first part of the web page. And here's the part which has the actual JavaScript. And those are the three functions, which we were executing. First one, search for the name, and then ask the parents to remove the child. Second one, search for the first paragraph, ask the parent to remove the child. Third one, search for the parent straight away. And then from the parent, remove the child, and then you have to refer to the child a bit more precisely.

So, let's look at another possibility. Sometimes you want to delete all children, not just one child, but all children. For example, perhaps you want to throw away everything in the webpage. Well, the webpage is stored in the body node and so you could go to the body node and then dump everything underneath the body node.

So that's what we're doing here in this example. Let's run the example first.

In this example we just run some JavaScript and it's all the children in the body node.

As you can see, everything's gone, including the button, because the button was also a child of the body node. Let's carry on with our presentation. All right, so let's clarify what we just did, with that example. This was our web page before we did anything clever with JavaScript. We had a body, we had a paragraph, had a bit of text, had a break, had another paragraph with a bit of text. Doesn't really matter what it had. And then we ran some JavaScript code and it deleted all the children of the body. Now, when you delete the trojan, the things underneath those things, the nodes underneath the nodes which get deleted, they also disappear, because they can't be connected, so they've also disappeared. So we have, in this example, six nodes that get destroyed and thrown away.

The result is this, just the body element by itself.

So that's what we just saw and this is the key code. We searched for something called the body and that was the name of theBody. And then we had a loop. Okay. A little JavaScript loop so that loop it keeps repeating. Repeats again and again. And it says while the body has a child, while it has a first child, remove the first child. So it will remove the first child. And then it will check. Is there still a first child? Yes there is. Then it will remove the first child. Check again. Is there a child? Yes, remove. And it keeps repeating in a loop and removes all of those children of the body.

What does this say here? Another way of typing this would have been while theNode.firstChild exists. That's the, kind of English version of what we're looking at. While there is a first child. While the first child is not nothing. That's other English ways of thinking about what's happening here. So we have a loop, keeps repeating while there is a child. So there's the whole code, we had a webpage, lots of things inside it, and then we clicked it, and then we ran some JavaScript code, searched for the body, and then has a loop to go through and destroy every child.

So, that's the end of our examples talking about deleting nodes.

10. Cloning Nodes

So now let's look at Cloning Nodes.
0:05
So, after this presentation you'll be able to copy, clone, same thing, a node, you'll also be able to copy a whole set of nodes in a branch. These are the commands we're gonna look at.
0:21
So, basic idea is copy, paste. That's really the idea when we say the word clone.
0:29
So just remind ourselves of one of the words that we've used and that is the word branch, particularly. You've seen this slide before and that's basically meaning a whole area, a whole branch of nodes all together. We'll use that word a bit later.
0:45
So, let's do some cloning, some copying. Basically, you just say the node that you wanna copy .cloneNode(). That's actually the same as the node you want to copy .cloneNode(false), all right? So you have nothing, false, same thing. That's how you would copy a node.
1:07
Let's do an example. Let's copy a list item in a list. And where should we copy it? Well, let's copy it to the end of the list.
1:18
So let's run this example.
1:22
Okay, so let's run the example, perhaps over here, on this machine here. So we're gonna do a cloneNode(), with no message in there, which is the same as cloneNode(false). So let's press, Copy it! And it has run the JavaScript, and that has copied the list item. The list item has been copied. However, you can't see it. So you think something wrong has happened, and you're right. Something incorrect has happened. So press Copy again. And again, the same thing, we don't actually see a list item content that we're trying to copy. And there's a reason for that, the reason for that is because we said cloneNode(), and we didn't have the right parameter in here. The right parameter was true, we did not use that, we used false or nothing. So, I'm running some JavaScript and it is copying the list item, but actually it's not copying the text elements that goes with it.

2:24
Let's continue with our presentation.
2:28
So this is the code that we just executed.
2:31
We have a webpage, we have a list item,
2:37
lots of list items inside a list. There's our first list item. There's our second list item. All of these are inside an unordered list. And then we had a button. Click on the button, it runs some JavaScript code. JavaScript code, it finds the list. myList, there it is. getElementById, myList, find it. And then find the lastChild, there's the lastChild of the list. It's the list item with the word, Hello, in it. So, we find the lastChild and then that particular node, we clone it, we copy it.
3:16
And then we wanna paste it somewhere. How do we paste it? Well, we know how to paste things. We do find a particular node and then appendChild(). That is one way to paste the copy, so we do copy paste. Okay, cloneNode(), appendChild(), and so that has added the copy at the end of the list. However, it didn't look very good. It did not copy the text, Hello. It did copy the list item, but it did not copy the text within the list item.
3:48
So, that's not very good, and that's what's happened here. There's our first list item, there's our second list item. I said clone it and then add it at the end. So it has added the list item but it has not added the text node. S, that's because we didn't say cloneNode(true), okay? It kind of did not add this text node. It did not copy the entire branch. Really what we wanna do, is copy the entire branch and then paste the entire branch. So that's the failure here.
4:23
So, cloneNode(true) much better. Copy the entire branch. Much more common. So let's do the same code basically, but this time we copy the branch, we do cloneNode(true), so it copies the list item together with the text node underneath the list item. And then we add it at the end of the list. That's what we want to do. So let's run the example.
4:50
So this is the correct example. There's our list together with the two items, which are already in the list, let's run our code to trigger the cloneNode() and now, yes it has added. It has cloneNode() this one and then it has appended it at the end of the list. I could run it again. It does a cloneNode() and append. A cloneNode appended. This is successful because the cloneNode() is copying the branch now instead of just one single node.
5:20
Let's go back with our presentation.
5:24
So this is the code of the example we just saw, exactly the same as the code that we saw before, the first example. The only difference is, we now use clonenode(true), so it copied everything underneath the list item including the text node. So, now, everything works well.
5:43
So, just to illustrate it with our DOM. That's what we started with, two things inside a list. And then, we did cloneNode(true). It copies the entire branch and then we appended it at the end of that list. Success. So, that's the end of our discussion about cloneNode().

11. Mouse Events

Let's now have a look at handling mouse events in JavaScript. So, it's all focused on mouse events in

JavaScript. These are the events that we're gonna look at.
0:12
So, you're handling mouse events in JavaScript. Let's look at the most common ones. So, when you click on something, anything in a web page, that will trigger an onclick event, and that's probably the most common mouse event. There are others as well, so here's a few. So one is, onmousedown, and that gets triggered when you put your mouse over something and then you put your finger down on the mouse button but you have not yet let go of the button. So finger down on the button, that is a mousedown, all right? The other side of that is mouseup. So in other words, first one, mousedown, finger down on the mouse. Second one, mouseup, is when you let go of the mouse. So mouseup follows mousedown, really. In fact, you can think of the onclick idea as this one followed by this one, right? So onclick is basically onmousedown followed by onmouseup. Now most people don't really care too much about these, they just focus on onclick, it's simple.
1:18
So let's have a look at a simple example of using click. All right, onclick, so we have some text here, click on the best social network, and then it shows three pictures. One, two, three. One if for Facebook, one is for Google Plus, one is for Twitter, it's just some fun. And then if you click on this picture, it runs this second function, bad choice which shows a window, I don't agree. Click on the second picture, it runs the same function, I don't agree. Click on the third picture, it runs a different function, good choice. I do agree, good choice. Very simple, let's run this. And just to illustrate the onclick event handler, so let's run the first picture. I click on the first picture, I don't agree. Simple message. Click on the second picture, triggers the event, which triggers the code. I don't agree. Click on the third picture, and it triggers the event, which then triggers the code. Good choice. Okay, very simple, but a nice illustration of onclick.
2:21
So, some more mouse events. onmouseover, when you put your mouse over an object in the web page. onmouseout, when you move your mouse away from that object in the web page.
2:36
Two events there. Let's illustrated that with quite a clever little example here. We load this into the browser. What do we see? All we see is a div, right? Remember, a div is basically a rectangular area. So what do we have in our div? We have some text. Move your mouse over this and then move it out. Very simple. All right, and then we've got some behavio,r which is all doing something clever. Let's run the example to illustrate it.
3:09
So, here is the example.
3:12
As you can see, a message, move your mouse over this then move it out. So, let's move our mouse over it. Boom. Triggers a change.
3:21
Change a background color, and then move it out. That triggers another event, which does something different. So again, mouse over event, it does something. Mouse out event, does something else, okay? Mouse over, mouse out. Mouse over, mouse out. That's a nice simple way to get some kind of basic interaction in a webpage, right? So here's the code. Let's look at it in a bit more detail. So, we have our div, and then we've given it a name. Doesn't matter what the name is. We've got some text inside the div, which we saw in the web page. And then we've got some styling, and we've got two events. So, let's look at the event handling cuz that's what we're focused on. So you move your mouse over, over the div, then it will run some code and it will say change color function up here. And it passes a parameter to it, the parameter is red. That comes up here, red. So, move your mouse over the div, then it runs the function and it gives it red. If you take your mouse away from the div, then it runs the same function, but it gives it a different parameter, yellow. That comes up here and then we use it down here.

And what do we do with it? We use getElementById, and we find that div in memory. And then we change the style, what part of the style, the background color, so that it is the parameter that is given to it, all right? So we give it red here, okay, red comes down here. Give it yellow, all right, yellow comes down here, and we set the background color of that div according to what you do with the div. Move your mouse over. Move your mouse out.
5:12
Now what about this style thing here? Basically, it's just setting the width, the height, the position using top and left and the size of the text. Basic style properties. Typical things. And also, don't forget we need position:absolute so that the browser doesn't set the position of the div. We set the position of the div, okay? But still, it's a nice little example of onmouseover and onmouseout. All right, so that's the end of our explanation of mouse events in JavaScript.

12. Timer Events

Let's have a look at Timer Events in JavaScript. That's what we're gonna focus on.
0:07
So these are the commands we're gonna look at.
0:11
In general, the idea is simple, it's doing something later, possibly many times. That's the general idea. So we call that timers. Very, very useful for dynamic web page behavior.
0:26
So how do we construct a timer in JavaScript? We make a variable and then we use the variable to point to a timer. All right. We just set up a timer, and then that timer is going to run some JavaScript code, such as a function, a little bit later.
0:47
So this part here is setting up something to happen later. What's gonna happen? This function is gonna be executed. When is it gonna happen? It's gonna happen 1,000 later. Now that is a value in a milliseconds, so it is actually 1 second, 1,000 is 1 second. So one second later this function will be executed.
1:12
Now when it does this, it doesn't wait. You can do other things with your JavaScript code, many, many different things. And then suddenly, one second later, that code will be executed.
1:24
So that's the idea, doing something a little bit later in time.
1:30
So here's an example, let's run this example. Here it is, how long do you want to sleep? All right, so we can type in a value such as perhaps 8,000. Something like that, and then press OK. And remember, we're gonna use that value to set the delay for a timer. So we do OK, nothing happens. Don't see anything happening. But we have set up a timer, and eight seconds later, boom, up comes a message. Wake up, wake up, wake up, wake up. It's a silly example, but it just shows the idea of a delay before that JavaScript function is executed.
2:08
Okay, let's go back to our presentation. So that was simple to let's have a look at that. There's our web page, when we loaded our webpage we ran some code, what does the code do? It sets things up, all right. Functions can have any name this is just a kinda silly name. And so what does that do? Well it shows the question, how long do you want to sleep? And then we put whatever that person types. We're using prompt, so they can type something. Whatever they type gets stored in a variable. And then we use that variable for the duration of a timer. In other words, we say after that many milliseconds, we're gonna run this function here.

2:50
Where is this function? This function is down here. Very simple, it just makes a little window, "WAKE UP! WAKE UP! WAKE UP!!", very simple little example. There's the key part where we do a delay, and we run that function after the number which is entered by the user.
3:07
Right, let's carry on.
3:09
Let's do a kind of more advanced example, if you like. Now, let's show this one first. So, here's the example. As you can see, there is a picture. That picture is moving over to the right.
3:21
How do we control the speed? Well, that's controlled with a timer.
3:26
So, let's have a look at the code, which makes this work. So, when we load that web page, what do we see? Well, there's our web page. And we see basically a picture, an image. That is the image of the stones. And we apply some style rules so that our JavaScript code can move the position. Used by adding position absolute and reset the left position. That's the one that we change with out JavaScript code. So that sets things up. When the web page is loaded, start the timer and it runs this function here. What does it do? It finds in memory, it finds that picture. Both the stones using get element by ID.
4:10
And then it updates a variable which is storing the X position of the image. There's the variable here. Start with the value zero and then every time this function is executed we increase it by one and what do we do with that? Well, we use that variable, x position, we use it and we put it in the left property, right? The left attribute, in other words, the x position of the image. We find the image and then we change the style dot left, the x position of the image, so that will update it at the new position, which is moved a little bit over to the right.
4:49
Now, what do we finally, down here, the last line of code? Well, we set up a timer, and that timer is for the next time. Because we want to continually move that image. We set up a timer, after a very small piece of time, one 20th of a second I think. After one 20th of a second, it will run the same function again. Right, set timer. This is inside set timer. It is running the same function again, a little bit later.
5:23
All right, so a little bit later, run this function again, and then it will increase the x. It will update the x_position of the image, and it will run itself again, increase x, update position, again, again, and we have a special kind of timer loop.
5:40
So, that's what we saw when we just run the example.
5:44
Okay. So let's say you wanna stop that movement because that movement goes on forever unless you stop it. Maybe you wanna stop that movement. Stop a timer. Well if you've started a timer in the way that we showed. You start a timeout and you just give it a name to point to that timeout. If you do that, then all you have to do to stop that movement or whatever you're doing you just say clear timeout and then you use the name that you used when you made the timer in the first place, okay? So we used the word timer, so we use the word timer here and that will immediately stop that loop process, it will stop the timer Here's an example where we do that. We add a button to stop that. So, let's have a look. Okay. So, here is this example. And you can see it's the same as before, picture moving to the right hand side. But I just added a button here and that button, when I click on it, it triggers an event and we run some code. So, let's do it. Let's press stop. And now you can see, yeah, the picture is not moving anymore. I have in effect stopped the time of process, I've stopped the timer.
6:54

Let's carry on with the presentation. So here's our code and, as I said, as we saw just now, we've added a button, simple button. Click on the button, then it runs some code. What does the code do? It clears the timer. And where do we get this ting from, the_timer? Well, that was the one that we used, the variable name that we used when we made the_timer. So, as long as we use the same name, then we can clearTimeout, clear this particular timer. Because you can have as many timers as you'd like at the same time.
7:29
So that's adding a clear of the timer.
7:32
Let's carry on.
7:34
What else could we do? Well, that works fine. It's totally okay. There is another alternative. setInterval, and it's very, very similar. How do we set it up? Well, we just do setInterval. That's different to setTimeout, setInterval. We run this code, and we've got a number. So far, it looks identical to set time out, but the difference is after you've done this, every 2 seconds, it will run this code. Every 2 seconds. Not just once. Again and again and again and again, okay? So the other question is, how do we stop it? Something's happening, again and again and again. How do we stop it? Well we would use clear again, clearInterval this time because it's interval. And again you clear that variable, that variable that you used to start it.
8:26
Let's do a quick example, let's have a look. So here is our example it is exactly the same as before, the only difference it looks the same as before but the difference is in the coding we have now used setinterval to trigger the repeating pattern instead of using settimeout every single time.
8:44
Let's have a look at the code.
8:47
So basically, we rewrote a previous example, so that it used set time out. So, we load the webpage, and let's look at the code which is a bit different. When the webpage has loaded, we set up an interval. We set up an interval. Every 50 milliseconds we will run the code and the code is the same as before. It's responsible for changing the x value + 1 and then moving the image by that new x value.
9:18
Now, because we use setInterval that automatically runs that function every 50 milliseconds. So up here in this function that we built, I've removed the loop function right here. I said here, before, set timeout so it runs it again, set timeout, set timeout, set timeout. We don't need to do that anymore. Because we've set up an interval, it automatically does that repetition process every 50 milliseconds. Then the only thing that I changes also was the button, kick on the button, what does it do? Clear interval instead of clear time out. Okay, clear interval. All right, apart from that it's the same visual result, it's just using interval and clear interval instead of set time out. You really have a choice of which ones to use. So that's the end of our discussion about timers in JavaScript.

13. Adding Events Using JavaScript

Let's talk about adding events using JavaScript. So that's our focus, and these are the commands we're gonna look at. So we've seen this before, typical use of event handling. There's our web page, our body. And we want it to do something when it has loaded. Okay, so body unload do this JavaScript function, and then up here we have our java script function which is just showing a simple window. All right, we've seen that before. Everything works fine, no problem. Sometimes, you want to add the event handling by JavaScript. You don't want to do it by typing in the body or whatever. You wanna do it yourself. You wanna turn it on, you wanna turn it off under the control of JavaScript. So, this is one

way to do it. This is window.onload. So it's not body.onload. It's window.onload. So there's some small changes here, if you like. But once you know how, you can set it up like this. So that says from now onwards, when the web page has loaded, run the function do_something. We have set up an event handler function. All right, it works fine. Here's another way to do things. window.addEventListener and then which event caused this? There's lots of events as we know, click, load, that kind of thing. For the load event, this is the function that will handle it. So this is a second piece of JavaScript. It has the same result as the last piece of JavaScript that we just saw. It sets up the handler when the Web page has loaded.
1:44
Okay, now,
1:47
it is possible to have more than one event handler function for one event.
1:53
And if that is true, they get stored secretly in the browser, in a long list, in an array.
2:00
So what am I talking about? Well, for example, if you took this line of code and then you copied it and pasted it two more times. And you ran this line of code three times, it would make sure that when the web page has loaded, it would run that function not once, but three times. It would set up three event handlers, which may or may not be the same. And that would apply to that particular thing, a window being loaded. Okay, so the point is that you can actually add multiple event handlers to one item.
2:37
Right. So, let's say the event actually occurs, like a load. Then, all of those handlers that you've set up, could be three or four, they're executed one by one. Okay?
2:50
All right, now what about if you want to move an event handler? You've added too many. You've been making mistakes, something like that. Okay, I want to remove an event handler. So you find the item in memory using getElementById, which we've seen before. Find the item, and then tell the browser, remove the event listener. Which event, you have to say it, and whatever the code was that was executed, and that makes sure that, that particular item is not executed any more.
3:24
So you can add event listeners whenever you want, and you can remove event listeners when you want, any event.
3:33
So here's a little example. So let's run this example. So let's do Click Me! and up comes a message. Hello. All right, and a second message clicked.
3:48
All right, so I did one event, but it triggered two things happening.
3:53
All right, let's just try it again. Click OK. One piece of JavaScript and then another piece of JavaScript.
4:00
Okay, now let me remove the event listener and that's what I've done. And then I run my code again. One event list that comes up, but the second one does not come up.
4:13
Okay, back to our presentation, to look at the code. Okay, here's our code. So we can see, there's our first button, when we click on it, then up comes the message, we saw that.
4:25
But actually, when we clicked on it, we saw there was a second message as well. And that is because we had added an event listener. Okay, so we found our btn0, there's our btn0, the top button, and then we said, add an event listener. When someone clicks on it, run the code, do something, and that will

show a different message. Now, when we ran this example, we clicked on the first button, we saw this event being triggered first and then we saw this event being triggered second. Okay? Then what do we do?
5:05
We ran our second button, and our second button triggered some code down here. So we said, okay, when you click on the second button, let's run this function here. And our function said, okay, go and find the first button and remove event listener. What event listener? The click event listener. Okay, so when I click on the second button, it removed the click event listener. However, remember when we found that example, we clicked on the second button, it ran this code, it removed the click event listener. But we went back to the first button and tried it again, and it did show that first message. It didn't show the second message, but it did show the first message. So you can remove an event listener, but you can only remove an event listener that has been created using JavaScript. If you're using JavaScript to do removal, you can only remove an event which is created using JavaScript. Okay, the one which is created like this right here, so if I type it without JavaScript, it cannot be removed by JavaScript.
6:16
Okay, that's the end of our quick discussion of using JavaScript to add event listeners.

14. More on Functions

Let's look at some advanced uses of functions in JavaScript.
0:05
So, these are the different things we're gonna look at. How to assign a function to a variable.
0:12
How to pass a function to a function. And also how to return a function from a function. Fairly advanced features. And these are the different pieces of code we're gonna look at.
0:25
So what we know so far is, this is a typical way to make a function. No problem at all, everything works fine. Function, name of the function, have some code inside it. No problem, that's very typical.
0:39
You could also do one of these. You can actually say, make a function but you assign it to a variable name, okay? Make a function as usual, assign it to a variable. If you like, you could actually have the name of the function as well, but you don't have to, this one hasn't even written the name of a function. But you could write the name if you like. That function is assigned to this variable and you write all your code as usual inside the braces.
1:10
So, you could do that. We're giving a function to a variable and that function is declared when the browser gets to this point in the code. It may not get to this point at the start, when the webpage is loaded. It may get to this point, like for example, it may run a function with this inside it, later, possibly five minutes after the webpage each has loaded. That a difference to this one here. This one is created when the webpage is loaded. These are ways to make a function which are not straight away when the webpage is loaded but it could be later. So there is some subtle differences. Okay, what else could we do? We could pass a function to a function. In JavaScript, a function is just an object and you can pass that to a function. Let's see an example.
2:06
So the idea of this is, to check a operation. So here's one line of code.
2:15
Result is myDivide and then we are passing a function check, okay? That is passed to the division. It's just an example of passing a function here to a function.

2:31
Now the general idea of this is, if you divide a number by 0zero, you'll get an error, okay? You cannot divide a number by 0. Any number. So the simple idea here is, we write a function but we also include a checking function. We pass the checking function to that function. And then it uses that checking function to check if we can do a division. And we can do a division if the second number is not 0.
3:03
All right, so that function is actually here, check. All it does is, it simply checks if the second number is not 0 then return true. We can do division, otherwise return false. So there's a function. We are passing that function, check, goes to this function and we've given it a different name, fn. And so now we can use that function, give it two numbers which is the same two numbers. Can we do 44 divide by 1, let's try that out, 44 divide by 1 and that is given to the checking function, which returns true. So we get a message, it's okay, we can divide 44 by 1 no problem at all.
3:46
And so we get the answer coming back from my division, okay? Otherwise, that checking function has said, not okay, we have a problem, the second number is 0. Okay, now it's a bit of a kind of unusual example, a bit weird, but it's just illustrating a real example passing a function to another function and then using that function.
4:11
All right, what else can we do? We could return a function from a function.
4:16
All right, so let's look at an example of that. So here's our main code and we've said there's our function called counter. And we could go and we could run that function here and it does something and it returns, it doesn't return a number, it doesn't return text, a string, it returns a function, right there. That function comes back and is stored in the variable count, okay? So we ran a function up here, what did it do? Nothing very clever, but it returned a function in the variable.
4:54
All right, now we could use the name of that variable and we could run that function once, twice, three times, and each time we just add a number, and it will show a 1, 2, 3, okay?
5:09
So, that is an example of a function being returned from a function and then being used. Let's run this example. There we are. We get 1 and then 2, that's the second time we ran the function, which is stored in the variable. And then 3, that's the third time we ran the function stored in the variable. And then we finished. That's the end of our quick discussion of some advanced things you could do with functions in JavaScript.