Functions: Array Manipulation and Scope

Video Transcript

Video 1 – Introduction to Array and String Manipulation

So, in this section, we're going to talk about array and string manipulation. Arrays and strings are fairly similar. They hold different things. A string holds characters, and an array can hold actually any types; it can hold characters, it can how numbers. It can hold objects or even subarrays. And we'll see this with two-dimensional arrays, which we'll deal with later. But there are a number of functions that are provided by JavaScript for you to manipulate strings and arrays. The first thing you usually want to know is- how long is the array?

So, there's adopt length, and it's not a function; it's a property. Now, other things that are functions that you call on arrays and strings. So, for example, there's slice and splice. So, slice and splice allow us to pick out parts of arrays. Now with strings, slice allows us to pick out some strings, and we'll see that that's really important. Now, there's a nice function called split that allows us to take a string and say split it up where there are spaces.

So, this would give us, potentially give us an array of words. So, arrays and strings are very fundamental to the data structures that we deal with in JavaScript. So, you need to know and actually be familiar with how they're manipulated. We're going to give you some of the functions. But often, you'll need to look up other functions because there are 20 or 30 of them available. And we're going to deal with maybe just five or six of them in this exercise.

Video 2 – Array Manipulation Functions

So, let's take a look at how arrays can be manipulated. So, let's suppose we have an array here, '[1, 2, 3, 4]'. And let's suppose we want to add another element to the end of the array, to the right-hand side of the array; push will do that for us. So, if I say 'A.push(5), then the array 'A' now will be '[1, 2, 3, 4, 5]'. And the opposite is to pop. Now, we don't specify anything in pop; it will take the last element of the array and give it to us. So, let's suppose I want to say, 'B = A.pop( )'. Then 'B' will be '5'. It's popped it; we pushed it on the end.

Remember, now we popped it off, and 'A' will be '[1, 2, 3, 4]'. So, push, and pop go together and think of them like a stack of plates that we're going to push something onto the right-hand end. Then when we pop it, we take it off. So, it's like the plates stacked up in the diner that you take the top one, and the rest pop up by one. So now, let's look at splice and slice. So, let's take a look at 'A', it's '[1, 2, 3, 4]'. If I use slice, slice will always leave A in the same state. So, suppose I say, 'B = A.slice' Now, let's take '(0,' the start, and it shows us the end. Now, what it means is this is the 0th element, and let's take off as far as the second element.

So, '(0, 2)', 2 is actually the address of the third element. So, what is this going to give us? This is going to give us '[1, 2]'. So, B is 1 and 2. Slice leaves A the same. So, let's suppose I want to get 2 and 3. So, 'B = A.slice'. Now, if I want to get to 2 and 3, two is the '(1, ' and then I want to say '3)'. Yeah! So that gives me '[2, 3]'. So, that's slice. Let's take a look at splice. Now, splice changes the array. 'A' still hasn't changed. But now, if I do 'B = A.splice', and let me say '(0,' and now let me say '1'. Now, what this is, these arguments are slightly different.

The first one is the start of what we're going to delete, actually. And then the deleteCount, '1' is the deleteCount. So, we're going to delete one of them. So, we're going to take off the first element, which will be the '1'. So, 'B' is '[1]'; A now is '[2, 3, 4]'. Suppose I want to take the next '2' off. We do 'splice'. And now the 0th element is 2, has got the value 2. And let suppose I take two of them off, then I'm just going to be left with a being '[4]'. These powerful ways; splice is often used as to pull that first left-hand element off. '(0, 1)' will always just take that first element off that array.

Video 3 – String Manipulation Functions

So, let's take a look at some of the most common functions that we have for string manipulation. So, we have a string here, 'A', and we can find out its length, 'A.length'. And that's not a function; that's a property. So, there's no brackets after it. But most of them are functions. So, for example, 'A.split', let me say 'B = A.split'. And we need to say where we want to split, split, split on the spaces between the words. So, 'A.split (' ');' gives us an array 'B' with four entries. In 'B[0]'. So, it's this; 'B[1]' ' "is" '. So, split is very useful indeed to give us immediately an array.

Let's take a look at substrings. So, 'A.substring'. Now, this uses the start and end locations. So, the string starts with 0, so the t would be a 0. Let's pullout from '(1, 4)'. What will that be? It's not going to include the fourth. So, we have from 1, 2, 3 are included, 4 is not included. So, we get ' "his" ' back. So, the other one indexOf. It'll give us the first index of, for example, if I did say, '('string')', it would tell us where string starts. So, it starts at '10'.

So, that would be location in the long string. This, in 'A', this would be the location of the 's'. And of course, if I just do that, it would give us the location '3' because that's part of t, h, i, s; this. Okay, so those are some of the most common. There are many others. And you can see a lot of them are given by, if I do 'A.', you can see there's a lot here that we can use; 'split', 'substring', 'UpperCase( )'. ' "THIS IS A STRING" '.

Video 4 – String Template Literals

So, let's take a look at template literals. These are new in ES6 and are really useful. So, let's suppose we have an expression 'a' that's '10', and 'b' that's '3'. Suppose I want to set a 's = `The alien' Now, here, this is a backtick( `), and it's at the top left of your keyboard. And if we evaluate that, It's going to say '${a + b}', '10 + 3', '13'. So, everything in the '$' and the '{ }' is evaluated. So,

it'll evaluate that expression. So, let suppose we have a function, and I've got one that I just wrote called 'hey', and it returns ' 'hello world'; '.

So, we could say now, 's = and we could do the ' `The alien'. And now, ' `The alien says ${hey( )}` ', gets evaluated, ' hello world" '. So, this is very, very useful, and it allows us to do multiline, what we call literals. So, 's=', and in this case, we have, ' `the day is good but tomorrow it will be better` '. And this then evaluates to backtick, and it's perfect. So, these are really powerful, and we'll be using them in the course.

Video 5 – Anonymous Functions and Fat Arrows

So, let's talk about the various ways we can write functions in JavaScript. So, this is one way where we say, 'function', and we give it a name, and then some arguments, and then the body of the function, whatever it does, and this can be multiple lines. Here, I'm just showing returning 'a + b;'. But we can also write it this way where we say, for example, 'let add =', and then we can put 'function'. And then if I'm pretty it up, it looks like this. Now, here what we're saying is, I've got a handle, now, on this 'function'. But over here, this function doesn't have a name.

The handle is 'add', but this is function, and there's no name to it, and so, we sometimes call this an anonymous function, this part, because we can use an anonymous function. For example, let suppose I am using 'setTimeout'. And 'setTimeout', we have to put a function in here, and then we say a '1000' milliseconds. It'll call 'func' every '1000' milliseconds. So, I could put, I could put 'add' in here, except this is not allowed to have any arguments. But we could certainly put an anonymous function there.

So, for example, we could put that, we could define the function. So, 'function( )' and maybe '{console.log}', and we'd say something like, '('Hey')'. So, here we're putting in, and it's an anonymous function, and that's kind of neat. Now, we've got no handle on this, so we can't call this function again, and that could be a problem. But that's anonymous functions. Let me show you another way of writing a function. This one is called fat arrow(=>). We have an equal sign and a greater sign. So, now let's rewrite add as a fat arrow. So, we can say, 'let add =', and in this case, we just specify the arguments, and then we use a '=>', and then we say what we want to do in the body of the function.

Okay, '{return a + b;}', and pretty that up. That's what it looks like. 'let add =' that. Now, we can use fat arrows up here. Suppose I've got this 'function'. I can just say this 'function' is this, and a '=>', and there it is. That's the function. It has no name but here, and as I said, 'setTimeout' doesn't take any arguments, but so we specify no arguments, but here's the body. So, take a look. I love using fat arrows; they're very concise. So, all of these ways are ways of writing a function. We can use it anonymously. Okay, so, take a look, practice writing functions in different ways.

Video 6 – Score of Variables

So, let's take a look at scope. We see that var has a function scope, whereas let and const have block scope. Now, let's see what that means. Let's actually do an example. So, here I'm in the Console. And let's suppose I say, 'const x = 99;'. Now, this has, it's not inside a function. It has a kind of global scope. So, it certainly is available if I now define a function. So, let me, I'm going to copy some code here, and we're going to analyze what happens. Here, I've got a 'function f1( )', and I 'let y = 11;'.

Now, remember that's called block scope. So, what block is that in? Well, it's in this, and this ends way down here. Okay, so the scope of this is actually the whole of this. It exists in that. Now, there's another function defined inside of 'f1', and that has its own 'y' So, its scope, that 'y' shadows this one inside the function and has a value '33;'. So, where we print out '(y)' here, it will have a value of '33;'. Now, let's take a look at where things are printed out in what order. This is executed first, then this 'console.log(y);', and then 'f2' is called.

So, I obviously have to call 'f1' first, but 'f2' will then be called, and '(x)' is passed into 'f2'. So, what x is that? Well, it's an argument, so it gets the value of '99'. But it here, so that '(x)' here is not exactly the same. It's a copy of that one, but it's certainly not a const. It exists within this function, and therefore, we can modify it here, and add '1' to it. If it was exactly that one, we wouldn't be able to do it because it's supposed to be a constant. But here, it's okay. So, what we're going to have is this '(x)' will be referred to that constant one up there; '99'.

So, that's printing out the constant 99. y will be this one '11'. Now, f2 is called with the value 99. But this (x) is not exactly the same as that, it's the same value, but it's a different variable. And now, I can add 1 to it, so I've got 100. And here I'm going to print out a 100 and then 33. Let's execute it and see if we're right. So, 'f1()' doesn't take any arguments. I think I was right. '99', '11', '1' was added to the x, remember, and '33'.

So, make sure you understand what's happening here. That variables defined outside, say a function can exist within, they may be shadowed. So, here, for example, this 'y' shadows that 'y', that one would exist inside the function if that 'y' didn't override it here. So, be clear on what scope means, which variables are in existence at what time.

Video 7 – Passing Functions by Reference

Let's take a look at functions, especially when we pass functions as arguments to other functions. So, let's take a look at a definition of a function here. So, in this case, we've got a variable 'f1' that is a reference to this function. And here we're using the backticks, and we're just printing out the value of x there, ('f1;' whatever the value of x is that we pass in. So, for example, if we do 'f1('hey')', then it prints out 'f1: hey'. 'f1:', and then the value, whatever the value of x was, and inside here it was, 'hey', so it printed that out. Okay, so that makes sense. And if we did 'f1(37), then, yep, prints out 'f1: 37'.

Okay, so let's suppose we have another function, 'f2'. So, now 'f1' and 'f2' are references to functions. We can pass them into a function. So, suppose I have a 'function f3' that takes functions as arguments. So, let me call it '(fa, fb)', that I'm passing in. But one of the things is could fire the function. So, for example, I could have '{fa', and maybe pass it to something like '('hello');}'. I'd need to know what it does, but certainly, that is a possibility. And 'fb', I just don't do anything with it. But that should be okay.

So, now we've got a 'function f3', and if we pass in 'f3', and we pass in say '(f1, f2)'; then what's it going to do? It's going to execute 'f1' with 'hello'. So, let's see how that works. Yup, it prints out 'f1: hello'. So, we can pass functions around into other functions. We can have those functions fire them. 'f4' may take two functions, '(fa, fb)', and for example, might pass them back as an object. So, 'fa', depending on what we put in there and 'fb'. And now, it's going to; this would be equivalent to 'fa: fa'. So, 'fa' would be the key. And let's do it actually, longhand 'fa'.

So, the first fa is a key. The second fa is this function we're passing in, assuming that we have passed in a function. So, let's pass it back like that. Okay, and that looks good. Remind ourselves 'f4' is this function. And we can pass in, for example, f1, and f2. So, I'd say something like result because I'm going to get an object back 'f4' and now, I'll pass in '(f1, f2)'. So, the result, notice is we've got a function that 'f' is short for a function. But this is function one, this is f2.

How would I fire f1? Wow, I'd say 'res', cause that's now what it's called; '.fa', that's that function, and I could pass in '('hello world')'. Let's see. Yeah, it's that f is f1 and it's going to print out 'hello world'. And we could pass back 'res.fb', and if we pass in there '('goodbye earthling')', 'fb' is whatever this function is, and we passed in f2. So, now we're firing f2. Yep, 'f2: goodbye earthling'. So, we can pass functions around very easily.

We can put functions into objects. So, we can have objects having variables and functions in them quite easily. So, this is one of the nice things about JavaScript is it's very easy to pass functions around. So, we could have a number of functions, for example passed into a function that might choose which one to fire. So, it might fire left, or it might fire right, depending on some other state in that function.

Video 8 – ES6 Module Pattern

So, as our code grows, we want to import libraries, and in JavaScript, we usually call those modules. And so, you'll see statements like this, 'import {', and then two names, 'from', and this obviously is a JavaScript file ' "./module.js"; ' here. And in module.js, we'll see that there's an 'export'. And so, we can export things. And typically we want, when we're writing a module like this, we want to hide some things and export others. What we mean by export is, we expose them to the world. And we saw when we were talking about writing our own kind of modules, how to do that? And here we're saying, "Okay, I'm going to wrap my functions up into an object."

So, here I got 'foo:', and 'bar:'. So, those are functions, and I'm going to expose those, I'm going to 'export' them. And also, I got a string here, 'hello'. And I say, I'm going to export that too. So,

we use, you'll see 'export', or sometimes you'll see 'export default'. And now, in main.js, I'm going to pick those up with 'import', and I can decide how many of the things I want to use. I may just want to use 'hello' or just use 'funcs'. But here, I'm using them. These two statements here execute 'foo( );' and 'bar( );'. And if you look back, they are alerts.

So, they're going to raise a window with '("foo")' in it and a window with '("bar")' in it. And here I'm getting hold of a 'div' by its 'id(app)', and writing it into its 'innerHTML = hello;'. And if you remember what 'hello' was, it was just an '< H1 >' tag with 'hello' in it. And so, the rest here's our main code. If you'd like our HTML, index.html. We're picking up main.js with a module.js imported. And so, we can pick it up as a '< script >' here in a '< script >' tag. And here's our 'div' that we're writing into. Now, to access this, you can't just drop the file onto the browser.

Let's, so here, I've got a browser. Here, I've got my index.html file, and if I drop it on there, nothing happens. Now, this is because we need to access it through a web server. It will have problems with what they call cause. So, here I'm in the directory where I want to. So, let me just show you. This is where I've got my index.html file, my module.js, and the main.js. I want to run a webserver here. And we can do that with 'np' not 'npm', but 'npx' and 'http-server'. This will run a web server and will serve up index.html on port '8080'. So, now let me go here to localhost:8080.

And now, we'll see if I go there, foo raises an alert, bar, that alert, and now it writes hello. So, that's quite slick. So, just one last example here where I've added two separate functions, 'f1', 'f2', and I've added them to the exports. And here in main.js, we can just add them to the imports. So, this will feed '("Goodnight"); ' into 'f1', which is an alert, and 'f2('Good morning'); '. So, let's take a look at this running. So, here we are. So, we've got foo; we've got bar; we've got Goodnight, and we've got Good morning, and then we get hello. So, you've got great flexibility in how you export and import.

Video 9 – Module Pattern

Let's take a look at an anonymous function pattern that's used in module creation. So, you may have seen this construct before. Let's take a look. So, suppose we have an anonymous function like this, but we enclose it in brackets, and then we cause it to be executed. This '( )' at the end causes this function to execute. Let's see. Yep, there you are, it printed out 'hey'. So, this 'console.log('hey);' gets executed. And we can use this construct to partition our code. That functions defined inside here.

For example, variables inside here only exist within this function and are inaccessible from outside unless they're possibly returned, and that's what we're going to see. In the module pattern, we're going to modify this so that something is returned. Let's take a look how this works. So, we're going to come in here, and we'll have, we'll declare a global 'Module ='. Now, we need to return something so that this gets a value. So, inside here though, we can imagine that we could return something. As you see, nothing is defined there, but let's go and enlarge on that.

So, let's suppose I do this. So, I'm returning an object, but in that object, I've got that 'publicMethod' points to a function, and that function just as 'console.log('Hey')'. Suppose I execute this. What do I, what happens? Well, nothing happens, but now I've got a variable called 'Module' that has, as you see, it has an object. So, if I want to access that, I can do it with ' . ' notation 'publicMethod'. And we know it's a function so we can fire it. And we get, 'Hey', we execute this.

So, this is a beautiful construct that's quite widely used in JavaScript. Okay, so let's expand on this method. So here, let's define a private variable that won't be accessible outside this function, also, a private method. So, this function, but we're going to return this 'publicMethod'. So, in an object, we return the 'publicMethod', which is this function. Let us see how this works. So, we know we've got 'Module', and we've got a '.publicMethod( )' that I should be able to fire. Let's see.

Yup, we can get it, the 'Hey'. So, this kind of modular structure is actually quite important in the sense that we can control our variables so that other codes can't access them. And actually, we won't pollute other codes. So, we have a very clean interface if you like, we're defining an application program interface to our module. So, this is quite a powerful pattern that you will see in JavaScript. So, you see the key to this is that we only reveal the interface that we want to reveal. Everything else remains hidden.

Video 10 – Introduction to Walk Boston Exercise

This exercise is about walking the Boston data. So, we've acquired some data from Boston of the salaries of all its employees. There are over 22,000. So, it's quite large data. We want to give you the experience of handling large data. And what we want you to do is to walk through that data, picking out the things that we're interested in. And so, we're going to tell you to pick out certain data. You need to do that, put it into a form that you can manipulate, and then we want you to render it on your webpage. So, this will give you a lot of experience with handling fairly large data.

Video 11 –Boston City Data

So, in this exercise, we're going to look at Boston city-data. This is employee salary data for the city of Boston. And we're going to ask you to pick out certain things from this data. So, first, let's go and get the data. So, here I am in a directory that I've set up to catch the data and where we're going to be operating. I've already got a few files in here. So, I've got an 'index.html', which I'll show you. This is the one we used to read the posts and inject them into a 'div'. And what we're going to do is we're going to do a 'curl', and we're going to get data from here.

Now, I want to save this data. So, I'm going to do not just a plane curl which will list it out in the terminal. I want to put it into a file. So, I'm going to call this file 'boston.', and I'm going to call it '.js' because I want it to be eventually JavaScript. Now, I give the name of the file, and this is at 'amazon'. It's an 's3' bucket there. And we got 'bostonEmployeeSalaries.json' So, we're going to

go get that. And you see it downloading, and now, if I look at it, we should have, yeah, we've got a file, and it's got almost 6 megabytes, '5.8' megabytes of data.

So, it's quite a big file. Let's go and have a look at it now. So, here is your starting code. This is the code we used to pick up a file called 'posts.js'. And if you remember, we injected that into a 'div'. So, here we've got a 'container', that's the 'div', and we inject it into the 'HTML'. So, we're going to use this file, but we're going to pick up instead of 'posts', we're going to pick up boston.js. The file we've got here. Okay, first let's take a look at the boston.js file. So, it's one; it's quite a big file. And you see it's got lots of data. It's over 20,000 rows of data.

So, we can try and look through here to make sense of this. But it's easier if we pick it up and look at it in JavaScript. So, let's do that. We'll go to index.html, and I'll give myself some space here. Now, I'm going to edit this. So, instead of 'posts', I want to get, now, I may as well do all of these. Okay, we'll call this 'boston.js'. Okay, I've replaced 'posts' everywhere with 'boston'. Now, we're picking that up. So, we should be able to at least pick up this file. Now, the problem is we downloaded what kind of file?

Well, we call it boston.js. But if we look at it, it's not. This is a JSON file. So, what I need to do here is I need to put a 'var', and I'm going to call it the 'boston'. This will be the data, and I'm going to save it. Now, this is a JavaScript file. And you see it; we can edit quite big files like this quite easily. Okay, so now, we're, now we're good. Now, let's see here, I am going to go into the browser. Then I'm going to put this file, this HTML file, I'm going to put it into the browser and then, I'm going to go into the development environment and take a look at what boston looks like.

So, let's just do that. And you see, so now I've put the file in there. You'll see here, index.html. I put a breakpoint here. So, what will happen for you is first, it will fail. So, go to the Sources and put a breakpoint here at 'renderPosts'. Now, notice this is 'renderPosts'. There's often a problem that the browser will cache data, and it will replay the cache, and that's what's happening here. It should have picked up, remember I changed these to boston. It picked up an old cached file. So, I need to go in here. So, I'm going to move this across so you can see what's going on.

Under Tools, there's Clear Browsing Data. And you need to click on Clear Browsing Data. It'll take a while to clear. Let's just wait a moment. Now, it's cleared. Let's go back here. I'm going to reload the file, and it's still 'posts'. Let me get rid that, time to reload it once more. Go to F12. Okay, now we've got the right data here. We're looking at the right file. So, be careful of that. That may happen to you. Okay, so now I've broken here at '(boston)'. Now, notice '(boston)', if I hover over, it, says, we've got some data in there. We've got an Object, and inside the Object, notice these columns. This means this is an object, and it's got a property 'data'.

And that data looks like it's an 'Array', maybe of some other 'Arrays', and it's got a property 'meta'. So, if I wanted to access this, the whole object is called Boston. I'd have to put boston.data, and then I'd see these arrays. This is important because we're going to need to get hold of it. Let's experiment here. I'm going to the Console. And so, we're interacting now live with this program. And so, we said there's a variable called 'boston', and you can see it's got something called 'meta:',

and something called 'data:', and it's an object, the curly braces. Let's, I want to look at the data; how do I do that? Well, I do '.data'.

Now, that's all of this. Okay, just, let's put, I'm going to put 'var', I'm going to call it 'b = boston.data' So, 'b' now, is this 'Array'. And you see this '22,000'; it stops printing them. It's quite smart. But let me suppose instead of 'b' I put 'b[0]', that would give me that element there, that first 'Array(20)', and here it is. And if I click on this, it's going to expand that array. So, this is what the array looks like. In the first slot of the array, we've got '1', and then some numbers, looks like a code of some kind. Then we have the name of a person, where they're working, and we're looking for salary data.

So, this looks like a salary of some kind. This looks like another dollar amount or some amount, and this looks like something. We need to find out what each of these is. And to do that, we might need to look at the metadata. But we can guess what some of these things are. I'm going to print out. I'm going to print out '8' and '11', okay? Because I think that's the name, and that might be the salary or that might be, we need to find out. Okay, so let's go back now to program. So, in programming, I changed all the 'posts' to 'boston'. So, that's okay. I like to have a capital 'B' for the function name here.

Okay, that's nice. I can recognize it as the function now. Okay, so what I'm saying is 'Boston' now, what do I got? 'Boston' is the basic array, I need to get 'boston.data', remember? So, here let me change this. I want to say, let's say the 'people = boston.data' Okay, so that's got that big array, and I want to say 'var', the length of it is 'boston.data.length;' that's the length of that array, which we think is something like 22,000. So, this now needs to loop over; I'll call it 'len', that's fine. So, this is correct. Now, it's going to loop over whatever it is, 22,000.

Now, here, I don't want 'boston', this should be the 'people'. This would be 'people' it be, yeah, that be the '[i]'th person. Or let me call it 'person'. Naming is important actually, the names you choose. So, this will be the [i]'th 'person' in Boston. Okay, and we've got that. And now, we need to pick out, we've got an array. This is an array now. So, if I want to pick out the name, I said that's the 8th element. And here, let's pick out what we think might be the salary. We're not quite sure. So, we put '[ ]', and I think that was a '11' we said, Okay, that's. Okay, that's better. Save it.

Go back, reload. Okay, it ran. Okay, we were pretty good. So, here we've got now a list of all these people, and we can end this program. Let's make sure, just get rid of this. So, what we want you to do is to list out the top five salaries. And also, to tell us how many people earn over a 100,000. Okay, so in Boston, who are the top five earners, and what do they earn? And how many people earn over a 100,000? Okay. Good luck! It should give you a lot of experience with handling fairly large data. Okay, so that's your exercise. Bye for now.