Callback Functions

Video Transcript

Video 1 – Callbacks Introduction

Callbacks are a powerful construct within the language. It's at the heart of the browser, is at the heart of the JavaScript server. It's at the heart of the JavaScript DataStore. And because of it, this is something that you want to develop comfort with, something that you want to understand at its fundamental components. It is something that is very powerful, as mentioned. This ability to be able to designate a function that will carry out a task to be specified later, for example, when it comes to sorting, it might seem odd that the language itself cannot sort a group of numbers, but it requires you to write a function to be able to sort those numbers.

What might seem at first like a shortcoming, in fact, is a very powerful mechanism, which means that you can sort anything you want, from colors to numbers, to cars, any data structure you may conceive, as long as you provide that function. At first, it might seem a bit odd. The syntax might look a little bit dense. However, as I mentioned, start off small, write small pieces, tests your understanding, and develop that comfort that will be important and that will pay off greatly as you move forward in your learning journey.

Video 2 – Callback

In this lesson, we're going to be talking about callbacks. These turn out to be tremendously useful. They give us a lot of flexibility as well as extensibility. Javascript, as you might know, allows you to define functions as variables. In fact, they are no different than any other objects. The only difference is that you can invoke them. And so, we can take advantage of this feature of the language to pass them around and allow us to gain some flexibility. When it comes to callbacks formally, we think of a callback as a function that is passed as an argument to other code.

Now, the code that receives this argument is expected to callback or execute the argument, you know, that function once it's needed. So, let's go ahead and write some code to illustrate some of these concepts. Let's go ahead and move to the editor. And as you can see here, I have a list of numbers defined and in array. And so, what I want to do is I want to be able to write a filter, as you can see here on line 11, that will take a callback. Now, the reason you might want to do this is that you cannot foresee ahead of time all of the different ways in which someone might want to filter data.

They might not even have an array, they might have a collection of objects, or they might have some other complicated data structure. And so, how they are filtering, what criteria they're using,

you cannot know ahead of time. So, if you provide the flexibility of allowing the user of your API to be able to provide a function, a callback function, then you can accommodate this capability without having to write all of the functionality. So, as you can see here, we have a couple of functions.

One is the callback, which we will write in a minute. The other one is the filter, and then, we call the filter, as you can see, with two parameters. The very first parameter is the data that we're acting on, which is our 'list'. And the second parameter is the 'callback' function. So, let's go ahead and take a look at our callback function. This is going to be a pretty simple callback. We're going to filter this list that we have here. So, we'll create a new variable for ourselves. We'll call it 'newList'. This will be an empty array. And then we're going to create a 'for' loop to simply move through the data.

So, we'll use a JavaScript for loop, and we will run through the entire length of the list. So, we'll add a condition there to stop the loop once we've reached that length. Now, inside of the loop, we're going to have a condition where we're going to check to see simply the value of that number. And in this case, just, you know, to use, to take a certain example, we're going to filter for those that have a value of '<5'. So, we'll enter that condition here. And if that turns out to be true, we will 'push' that item into the 'newList', as you can see there, I have done just that. And then, once we're done with this, we will 'return' the 'newList' to the calling function.

Now, all we have to do here at this point is to be able to invoke that callback with the parameter. So, we will call the function, and we will pass the parameter, and we will return all of this to the line where it is invoked on line 22. So, this is a pretty flexible mechanism. And as you will see, there's a lot of structures within the language of JavaScript that allow you to provide a callback function. And we will see that throughout the number of lessons that we will do on this topic. So, let's go ahead and save this file and move over to the browser simply to see what this looks like once we execute.

So, let's go ahead and reload this page. And let's go ahead and open up the Console, and go ahead and load it. Oh, I am missing a message to the Console, so I can take a look at what that value is. I'm simply going to write the value of '(filtered);'. So, if we reload this page, you can see there that we have the values that are '<5'. If we were to change this, we would get the values that are '>5'.

And so, you can see then that we could change this function really to do anything else that we may want. You could have it concatenate the values. You could have it divide by 4; you could have it simply select the ones that were multiples of 2. And so, I'll leave this for you to do as an exercise to sort of push on this concept of callbacks and to play around with the flexibility that it allows you. And so, with that, I'll wish you happy coding and goodbye.

Video 3 – For Each Element in an Array

In this lesson, we'll cover forEach, which is a function which is part of arrays, and it takes a callback. It is a very useful function as often we have to loop through collections of information, and in this case, we can do it without using counters, and it also makes code a lot more compact. So, let's go ahead and take a look at some code. As you can see here, we have an 'employees' array, which is a collection of objects. Each of them are value pairs. They have a 'name:', and they also have a 'salary:'.

Notice well that each of those salaries are strings and not numbers. As you can see down here on line 15, we are using the 'forEach' function, which is part of arrays, and we're passing in a callback. That callback is '(add)', and we're going to write that next. So, let's go ahead and write our 'function' stub. Now, one of the things to note is that whenever you have a callback on an array, automatically behind the scenes, three things are pass in. The first is the item that you're working on, and it's, I wrote 'item'. Yes. The second one is the 'counter', and the last one is the 'array' itself.

So, each of those items in the array are passed one by one. The position that you're at is passed in as well, as well as the entire array on each iteration. So, let's go ahead and inspect what's taking place within the browser. And we'll write to the 'console' in order to do that or simply write the item. Let's go ahead and save this file, and let's go ahead and move to the browser. Let's go ahead and reload the page. You can see there that all three of those items were written to the Console. But let's go ahead and put in a breakpoint so we can see how this is called on every iteration of this collection.

So, let's go ahead and reload the page. You can see there that the first time it stops, it's passing in the first item of that array, which is 'name:' and ' "peter" ', which is this one right here. It also passes in the 'counter' position. As you can see there, the first item in that array is the [inaudible] 'item', and then it passes in the complete 'array'. So, if we look at, if we expand this, we can see there's ' "hanna" ', there's ' "bruce" ', and the first item is ' "peter" '. So, if we let the code run until the next breakpoint, we've again stopped. And this time, the 'counter' is '1'. The 'item' we're looking at is ' "bruce" ', and the 'array' will always be the same.

Let's go ahead and run it once more. And as you can see there, we're now on the last 'item' of the 'array', the counter is '2', and the 'array' once again, it's the same. Go ahead and run it all the way to the end. Let's remove the breakpoint; let's go back. And one of the things that we'd want to do, in this case, is add all of those salaries. So, let's go ahead and create a global, a global variable of 'salaries', and let's start it at '0;'. Now, each of those items has salary information, so we can use 'salaries' and access the item, that salary information.

Now because it's a string, we need to convert it to a number, and we can use the JavaScript 'Number' function to do that. And at the very end, we'll write to the 'console'. And we'll write out the 'salaries'. This is simply, so it reads better. And now, if we go back to the browser and we reload the page. And we move to the Console; we can see that we have salaries of 475000, if

you add up the values here, you'll get that number. So, as you can see, this is a pretty clean way of iterating through all of the items in your array and assigning a function to act on each of those items.

Video 4 – For Each Exercise

As an exercise, I'm going to ask you to do the exact same thing, to add all of the payments, in this case, of a dataset from the city of Chicago, and you can see that here. These are all of the payments that were made to the city of Chicago or that were made by the city of Chicago to vendors between 2005 and 2010. Now, it's a lot of data. If we go all the way to the end, you can see there we have over 30,000 data rows. And I'm going to ask you to walk through all of this and do very much the same thing we did with that array of three items. But now, do it on an array of 30,000 items. And as you'll see, it's very much the same exercise.

So, let's go ahead and take a look at your starter code. This, I'll provide you this file. And as you can see here, the very first thing in the file here is referencing that file, so that gets loaded into memory. So, you can see here the very first line is assigning all of this JSON data onto this variable of 'chicago'. So, once we get past this line, we have 'chicago' in memory, and we can start to make use of it. So you can see here, 'chicago.data.forEach' will let us act upon that information. So, let's go ahead and reload and load this file onto the browser. So, this is addPayments.html.

And now, if we entered 'chicago' at the console, you can see there that are the top of the hierarchy. We have two value pairs. One of them called 'meta', the other one called 'data'. And even if you knew nothing about this dataset, you'd get a strong hint that 'data' has the meat of the information which it does. So, let's go ahead and access 'data' and access a specific element, in this case, the element, the 0th element. And you can see there that you're already getting access to the information. There's many of these, so we can say access the 20th, 20 south in the element number 20,000. And you can see there that you're getting back information.

Now, I'll leave it up to you as a data science exercise to dig through the data and figure out which is the payment. There's a number of values here that could be potential payments. This is, I can tell you that this is the year. But if you look within the file, either at the data or comparing between rows, you'll be able to figure out which is the column that has the payment information. Now, if you come back to your file, you can write your 'add' function to add all of this information up. Now, make sure you are within the right column. And as you start to write to the Console you results, you'll be able to tell whether what you're doing is correct or not. So, with that, I'll wish you happy coding and goodbye.

Video 5 – Filter Callback

In this lesson, we're going to be looking at filters. This is something that you'll use frequently. You'll often have large collections of informations and arrays that you'll want to filter based on a specific criteria that you're going to specify. The language provides you this capability using a callback. So, let's go ahead and take a look at some code. As you can see here, I have an array of 'employees', and within it, we have four items. Each of those items is an object, and we have a couple of value pairs. We have the 'name:' and we have a 'salary:'. Now, in this case, we're going to write a filter that is going to select a number of these items based on the 'salary:'.

And we'll write that criteria in a moment. Down here on line 18, you can see that we're making a call to the filter function. You can see here 'employees.filter', and then we're specifying the callback. Now, callbacks for filters need to meet a specific criteria. They need to return a true or false value. If the item meets the condition of the callback, you keep that item in that new collection that has been filtered. If it does not meet the criteria, you return false. Now, you have the flexibility to write any criteria you want; you're writing the callback yourself. However, you must return true or false in order for the filter to do its job.

So, let's go ahead and write here a criteria for 'isBigEnough'. We're going to be specifying what our criteria is for 'isBigEnough'. In this case, I'm simply going to make a comparison, and anything that is over half a million; I will keep anything that's not, I will disregard. And keeping here means returning true. Disregard means returning false. So, let me go ahead and write a variable to hold the comparison value. And to this, I will assign the result of this comparison. So, I'm going to make a 'comparison' between 'salary >= 500000;', as I mentioned before. Let's see if we have three zeros. Yes, we do. And after that, we're simply going to return the 'comparison'.

Now, after we apply this to each of these items, as you remember, whenever we're acting upon an array, the callback is applied to each of the items within that collection. Once we are applying that callback function to each of the items, we will have a filtered array. And you can see here that I've assigned that to a variable call 'filtered'. And then I'm simply going to write it to the Console for inspection. So, let's go ahead and take a look at this within the browser. So, as you can see here, I have the page already loaded, filter.html, and if I reload the page, you can see there that I am filtering, and I'm only getting the values that are over half a million.

Now, let's go ahead and take a look at this within the debugger. And let's go ahead and reload this page simply to emphasize the point that it is applying this condition to every single item in the array. So, as you can see here, this is the very first item that it's being applied to. You can see it's the value of ' "peter" '. And Peter has a salary of 100,000. So, the comparison will be false, as you can see here, and that's what will be returned. So, this item will not be included. So, I'm going to go on to the next element. And in this case, this salary, this person has a salary. Let's go ahead and look at the element a 425. That should also be false. And as you can see there, it is false. Let's go to the next one. In this case, the element.

The person has a salary of 750000. And this one is true. And so, that one will be kept. And so, if we go to the next one, this is the last one. This one will also be kept because it's over half a million. So, if we continue, then we look at the Console, you can see there that those are the values that met the criteria. Now, simply to demonstrate what would happen if we change this value here, we should get the ones that are below. So, if I go ahead and reload and let me go ahead and remove the breakpoint, now, if we look at the Console, you can see that those are the salaries under half a million. So, that's the bare-bones example of filtering a group of data, an array of data.

Video 6 – Filter Exercise: Find the Graffiti

I want you to imagine that you're a fan of American graffiti and you're going to be visiting the city of Chicago, a city that has a lot of graffiti. And I've downloaded from the city of Chicago a public dataset that has reports of all the graffiti that has appeared within the city. Now, you might also know that American cities are organized into ZIP Codes. You can see here a lot of the ZIP Codes within the city of Chicago. And if I click on one of these, I can get more information, the actual number, code, and so on. And so, there's a lot of these in the city of Chicago. Now, let me go ahead and go back to the code.

And this file here has all of the reports that have been made about graffiti and the city for the month of July in 2005. I will also give you one for the full year, but that's quite a big file. So, get started first with the small one so that you can debug it more easily. Now, this file can be a little bit overwhelming as it has a lot of information, and as you can see here, if we go all the way to the end, it has over 12000 rows of information. Now, all of this information here is graffiti reports, and each of those graffiti reports is reporting on the ZIP Code.

So, your job is going to be to move through all of this graffiti information, identify how many reports are being made for each of those ZIP Codes, and you can do that using a filter. So, let me go ahead and show you here this starter code that I've given you. I'm referencing Google charts because, at the end of this exercise, we will display this information in a chart and then giving you a collection of the ZIP Codes, as you can see here. These are all the ZIP Codes in the city of Chicago. Now, I've also given you the file that I mentioned. This is the graffiti reports for the month of July in 2015.

And finally, I have some code there to graph all of this information. This part you don't need to concern yourself with. Now, at line 8, we simply make some space, we define it '< div' that is going to hold that chart, and then, let's get into the starter code here. We're defining a graffiti array to hold all of the information that we will collect. We're also defining another variable to hold the matches. And then finally, one that is an object, and I will explain more about this in a minute. And you can see here, let me go ahead and go back to the file. And let me simply illustrate that all of that information, as you can see here from this file, is being assigned to a variable called 'chicago'.

So, if we enter 'chicago', this is that file that I'm showing you. If we enter 'chicago' at the Console, we'll immediately get back to the top of the hierarchy of the information. And so, you can see there you have '{meta:', and you also have 'data:'. So, even if you knew nothing about this dataset, which most of you will not, you can tell that most of the data will be inside of the data property. So, if I enter that there you can see that I get those 12000 rows. And if I go further in and select a specific item, you can see there that I'm getting a specific graffiti report.

Now, I'll leave it up to you to figure out what the ZIP Code is. This is part of the fun when you do a data science exercise, is to be able to identify the information. All of that information is within that file. I'll give you another sample here. Let's go ahead and enter the 12th row. And you can start to compare and get a sense of what the information here is. So, let's go ahead and go back to the file and go look at the starter code. As you can see here, we're dealing with two collections; two arrays. The first one is the number of ZIP Codes that we know exists for the city. And so, we're simply going to do a 'forEach' and then assign a callback.

So, for each of those ZIP Codes, we're going to assign a callback, the callback as graffiti reports. And that simply gives is a ZIP Code that we're going to consider. We then use that ZIP Code to filter through the information that we have within the Chicago array. So, if you write your filter correctly here, you will be able to get back. This is very similar to the example we did before where we were simply matching, in this case, for a salary. In this case, you will match for a specific ZIP. In this case, you know the ZIP that you're passing in.

And you will get out of those 12000s, only the rows that match that 'zip'. Now, once you get that filtered array, and so you can see here, the array will be called 'matching'. Then further down, you can get a count. And I'm going to ask you to create a new object that we will then push into the 'graffiti' array that we've predefined on line 12. As you can see there, we create a new object. Then we assign the 'zip', 'zip' that were being passed and the one that we're considering in this loop. And then finally, onto reports, we are simply going to assign the 'length;'. This is the number of rows that we found matching that ZIP Code.

So, with this bit of code, you can get a count of all of the graffiti reports for every ZIP Code in the city of Chicago. Now, the last two lines here at the bottom are simply to generate a dataset, and these codes that are being written behind the scenes you don't need to understand how this works. But simply to know that if you construct your array here within this loop, then these last two lines will graph it onto the screen. And if you do everything correctly, this is what it looks like. You can see here a number of the reports. You can see that the report of graffiti vary tremendously with that in the city. So, if you are trying to find out which ZIP Codes had the most graffiti, and you enjoyed watching graffiti, you could make a good choice based on this information.

Video 7 – Sort Callback

In this lesson, we're going to be looking at how to sort. This is something that you'll do often when you have large collections of information. In this case, we're going to be considering first a simple array and later a large collection of JSON data. As with the rest of the lessons in this section, you will be doing this using a callback. So, let's go ahead and take a look at the editor and write some code. There's a few things to call out. The very first one is that we have an array. This is simply a list of numbers from '1' to '9' out of order. We're also making a call to the 'sort' function, which is part of arrays in JavaScript.

And we're passing in a callback. That callback is in lines 13 through 17. And we will be writing that code in a minute. Then at the end here, we simply write out to the 'console', simply to inspect our results. So, as you can see here on lines 4 through 9, these are the rules that JavaScript imposes on you whenever you're writing a comparison function. This is the callback function that will make a comparison between two items of an array. This is how JavaScript implements 'sort'. You must write a 'compare' function that returns one of three results whenever you have two items.

Let's call them 'a' and 'b'; you have three possible results. The first one is that 'a < b,', in which case you 'return -1;'. You have a case in where 'a < b,', in which case you 'return 1;'. And then the third case in were, in where 'a === b,', in which case you 'return 0;'. Now, this is a convention of the language. It chose to require those that are writing comparison functions to return a '-1', '1' or a '0' based on those three cases. This is simply convention; there's nothing special about it. So, we're simply going to follow these rules, and we're going to write a comparison function for ourselves.

Let's do the first condition, 'if (a < b)', then we're going to 'return -1;' . 'if (a < b)', let me go ahead and line up my code here. Then we're going to 'return 1', positive '1'. Else, 'if (a === b)', and this is the JavaScript comparison that compares both value, and type. We're going to 'return 0;' . So, we're taking care of those three conditions, JavaScript behind the scenes when we called 'sort' function, and we give it, the callback is going to pass in a couple of items at a time. And behind the scenes, choose an algorithm in order to sort this list. So, let's go ahead and save it. Let's go ahead and go to the browser and run that code.

If all went well, we should get the array in order, as you can see there. Let's go ahead and take a look at the Source and inspect that compare function to take a look at a couple of comparisons. We'll put a breakpoint in there, and we'll reload the page. And you can see there that the first two items have been passed in. Now, in this case, 'a' is greater than '2', than b. So, let's go ahead and go over, and you can see that it has advanced to the second condition. However, you will not go beyond that. At this point, it will return. Now, let's go ahead and go to the next two; in this case, it's passing in '4' and '7'.

And as you can see there, it starts to move through the array. The order in which it does it and how it does it, it's opaque to you writing the comparison function. But simply know that you need

to, all you need to consider is two values from that array being passed in, and take care of those three conditions. And the rest of it will be taken care of by the runtime. So, let's go ahead and remove this and simply run to the end and move on to the second exercise. So, this is a pretty straightforward bare-bones example on how to do or how to order an array using the sort function of arrays, giving it a callback function.

Video 8 – Sort and Chart Salaries

Now, what I'm going to ask you to do is I'm going to ask you to sort the salaries for the city of Chicago. The city of Chicago publishes the salary information for all of the public employees in a large dataset that we have within this file. All of this will be provided for you; as you can tell here, there's a great deal of information, and it might be a bit overwhelming, but as we'll see soon, it has a pretty good hierarchy, so it's easy to navigate. You can get here, and if we go all the way to the end, you can see that there's over 30000 lines of data or rows of data. So, let's go ahead and take a look at what this looks like.

You can see here that I've given you some starter code. And there's three things to call out when we get into the script. The very first one is the Chicago data is being assigned to a variable called 'salaries'. And as you can see there that all of that JSON data is assigned into this 'chicago' variable here. When we move over to the browser, we'll inspect this a little bit closer. Now, in addition to that, there are three I mentioned there are three things to call out. There are three functions. The very first one is 'generatePoints', the second one is 'compare', and the third one is 'run()', which will simply execute the order of functions within the page.

So, the very first one is 'generatePoints();'. If we go here, we can see that there's an array of points being created. And after that, there is a 'forEach'. So, it loops over all of the salaries within the data from the city of Chicago. Now, it creates, it pushes points into the points array, and it simply pushes in value pairs. That's the job, and that's also the salary. I've gone ahead beforehand and inspected this dataset, and I know that these two positions within the data give us back both the job title and also the salary. Now, what you have to do is you have to uncomment line 23. But first, write the compare function so that all of those salaries are ordered within the array.

And I'm going to go ahead and show you what that looks like before you sort it. And if we look at the Console, we can inspect the Chicago data. As I mentioned, you have here; there was a lot of data within that JSON structure. However, as you can see here at the very topmost, we have only a couple of value pairs. We have '{meta:', and we have 'data:'. We have over 30000 rows. So, if we go to city of Chicago with that data, we get directly access to all of those rows of data. And if we enter a specific row, you can see there that we're starting to get back the information.

Now, as you can see here, there's salary information; there's also job title. And what I've done in that first array, let me go back to the code. In this first loop, here, through the array, is to simply extract the job and the salary information. Now, the next step is for you to make the comparisons within, within that new array of points, as you can see here; it's 'point.sort(compare);'. And you will have to do the same thing that we did before, that we did with this array. But now, you will have to do it with the point, the objects that are inside of the points, each of which has a job and a salary property.

So, if you do that right, you will see here that you will create a nice ordered graph as opposed to this one that has a storm of information where it's hard to tell what's taking place, how does it grow? What does the distribution look like? On this one, you have a nice order set of information that goes from lowest to highest, from top to bottom. So, I'll leave that up to you now to inspect, play with. There's many other things that you could do. You can imagine being able to filter by types of job titles. You know, that the, what does the police make versus what do people in schools make versus what does the fire department make. And I'll leave this up to you now for you to explore and have some fun with. So, with that, I wish you happy coding and goodbye.

Video 9 – Map Callback

Now, what I'm going to ask you to do is I'm going to ask you to sort the salaries for the city of Chicago. The city of Chicago publishes the salary information for all of the public employees in a large dataset that we have within this file. All of this will be provided for you; as you can tell here, there's a great deal of information, and it might be a bit overwhelming, but as we'll see soon, it has a pretty good hierarchy, so it's easy to navigate. You can get here, and if we go all the way to the end, you can see that there's over 30000 lines of data or rows of data. So, let's go ahead and take a look at what this looks like.

You can see here that I've given you some starter code. And there's three things to call out when we get into the script. The very first one is the Chicago data is being assigned to a variable called 'salaries'. And as you can see there that all of that JSON data is assigned into this 'chicago' variable here. When we move over to the browser, we'll inspect this a little bit closer. Now, in addition to that, there are three I mentioned there are three things to call out. There are three functions.

The very first one is 'generatePoints', the second one is 'compare', and the third one is 'run()', which will simply execute the order of functions within the page. So, the very first one is 'generatePoints();'. If we go here, we can see that there's an array of points being created. And after that, there is a 'forEach'. So, it loops over all of the salaries within the data from the city of Chicago. Now, it creates, it pushes points into the points array, and it simply pushes in value pairs. That's the job, and that's also the salary.

I've gone ahead beforehand and inspected this dataset, and I know that these two positions within the data give us back both the job title and also the salary. Now, what you have to do is you have to uncomment line 23. But first, write the compare function so that all of those salaries are ordered within the array. And I'm going to go ahead and show you what that looks like before you sort it. And if we look at the Console, we can inspect the Chicago data. As I mentioned, you have here; there was a lot of data within that JSON structure. However, as you can see here at the very topmost, we have only a couple of value pairs. We have '{meta:', and we have 'data:'.

We have over 30000 rows. So, if we go to city of Chicago with that data, we get directly access to all of those rows of data. And if we enter a specific row, you can see there that we're starting to get back the information. Now, as you can see here, there's salary information; there's also job

title. And what I've done in that first array, let me go back to the code. In this first loop here, through the array is to simply extract the job and the salary information. Now, the next step is for you to make the comparisons within that new array of points, as you can see here; it's 'point.sort(compare);'.

And you will have to do the same thing that we did before, that we did with this array. But now, you will have to do it with the point, the objects that are inside of the points, each of which has a job and a salary property. So, if you do that right, you will see here that you will create a nice ordered graph as opposed to this one that has a storm of information where it's hard to tell what's taking place, how does it grow?

What does the distribution look like? On this one, you have a nice order set of information that goes from lowest to highest, from top to bottom. So, I'll leave that up to you now to inspect, play with. There's many other things that you could do. You can imagine being able to filter by types of job titles. You know, what does the police make versus what do people in schools make versus what does the fire department make. And I'll leave this up to you now for you to explore and have some fun with.

Video 10 – Reduce

In this lesson, we're going to be looking at the reduced function of arrays. As the name implies, it reduces the array to a single value. It applies a callback function to an accumulator into each value of the array from left to right. So, let's go ahead and take a look at an example. As you can see here on line 5, I have an array of 'numbers', and then on line 8, I am calling the 'reduce' function. Now, note the parameters of the callback function. We are, we have a 'previous' parameter and a 'current' one.

Most methods that take a callback simply take the current position of that array. In this case, we have 'previous', we have 'current', and we also have an optional additional value, which in this case I have set to '0'. Now, in this case, what I want to do is I want to simply add up the numbers of this array. So I'm going to take the current position and the previous one and simply return them to the next time this call is being made. So, let me go ahead and write this. I'm going to simply add the 'previous' to the 'current' and 'return' that.

Now, on the next iteration or in the next call of this callback function, the 'previous' value will have that summation. And we will then use that to add it to the current array position, which we will then return, and then we will add to the next position in the array. So, as you can see there, we're slowly moving from left to right, carrying the summation forward, and at the very end, we're saying we assign that to the variable of 'total'. We then write that to the 'console'. So, let's go ahead and take a look at this within the browser. So, let's go ahead and move over.

And I'm going to go ahead and reload that page, and you can see there that I got the value of '20', and that's simply adding up all of the numbers in that array. Now, let's go ahead and take a look

at the Source of this and step into that function to try to understand a little bit, in a little bit more detail, what's taking place. So, the very first time this function is called, 'previous' should have the value of '0' because that's the initial value, and then the 'current' should be '1', which is the first position in the array. Let's go ahead and reload the page, Aathe first call. And as you can see there, 'previous' is '0' and 'current' is '1'.

Now, I'm going to go ahead and jump to the next time it's called, and now, 'previous', which is the summation of '0' and '1', has the value of '1', and then the next value, it's '1' as well. So, you can see these, these values here, right? So, when I go to the next time, the function is called, now 'previous' is '2', right? Because it added the previous two numbers. And now, 'current' has a value of '2'. It's at this position in the array. And I go to the next function call. Now, we have a value of '4', and 'current' has the value of '3'. So, you can see there how it's making its way from left to right and carrying forward that summation.

So, I'm simply going to remove the breakpoint and move all the way to the end, and we can see there that we get '20', again. So, let's go ahead and look at an example that it's a little bit more involved. In this case, we're going to be adding the number of times a word appears within a list. As you can see here, now we have a couple of words that appear more than once, and we are assigning that to a list. That's simply an array of words. And then we're going to 'reduce'. We're going to get a count. Now, in this case, we have the 'previous' and the 'current' just as before.

But note that my initial value is an object, and what I'm going to do within this object is that I'm going to add each of those words as properties. And then every single time I see that word again, I will increase the value of that property by 1. So, let me go ahead and start to write that code, and I'll go ahead and remove this. And the very first thing that we want to do is we want to know if 'current', the current word is in the previous, the 'previous' value, and the previous value here is simply that object. It's 'current' in the object. Now, if it is, I'm simply going to increase the counter.

So, I'm going to use the property name of 'current', which is the word, and then I'm simply going to increase that by '1'. 'else', that means the word doesn't exist yet within this object, the property doesn't exist. 'else{' I'm going to create that property. And I will create it with the value of '1', that means we've seen it once. I'll then 'return' the object so that it can be used in the next call of the function. So, let's go ahead and save this file and go ahead and take a look at this within the browser. It is this next exercise here. And I'm going to go ahead and reload the page, and as you can see there, well, let me go ahead and use 'JSON.stringify', so that we can see what's the value is.

Let's go ahead and reload the page, and as you can see there we get a count, ' "hi" ' appears '1', ' "boo" ' appears '2', ' "hello" ' appears, '1', ' "ada" ' appears '3' times, and that matches the occurrence of each of those passwords. Let's go ahead and take a look within the Source here, each time that function gets called, and try to understand what's taking place in a little bit more detail. I'm going to go ahead and reload the page, and as you can see there, the very first time it stops, the 'if' statement, that object is empty, and the word that it has is 'hi', which is the very first word on that list.

So, since it hasn't seen it before, it should jump to the 'else' statement, and yes, it does. Now, if we move over to 'previous', you can see there that now the object has one property with a value of '1'. I'm going to go ahead and run till the next call, and I'm going to simply step through all the way to 'previous', and you can see now that we have two properties within, within the object. Going to go ahead and run again, step through this. Now, we have an increase. We've increased 'boo' by '2' because we've seen it twice. And so, you get an idea of what's taking place, going to go ahead and remove the breakpoint and run all the way to the end. You can see there the values.

Video 11 – Introduction to Word Count Exercise

In this exercise, we're going to count words in a paragraph. Now, we could go much larger than a paragraph. We could go to, for example, the whole dictionary of English words. But our goal here is for you to count the number of times each word appears in a paragraph. Now, to do that, we're going to take advantage of a special facility in objects. So, in objects, we use key value pairs. All of computation is about having something with a name and a value. So, x equals five, say. Now, in an object, we have a property, and we can give the key to that property, and we can use the word.

So, for example, we can pick out any word in the paragraph, and we can create a property with that word as the key. Now, we will have the value as the number of times that it appears in the paragraph. So, you need to loop over all the words in the paragraph. Much of your work is going to be cleaning up the paragraph. We need to get rid of spaces. We need to get rid of full stops. We need to get rid of punctuation in general. So, you clean up the paragraph, get it into a form that you can manipulate, and then run through it and put every word into what we might call a dictionary. It's like a HashMap. Other languages call it HashMaps, but in JavaScript, it's a basic object.

Video 12 – Word Count Exercise

So, in this exercise, we're given a long string of words, and then we need to count the most common words. So, here's the low string. It goes on quite a way. And what we need to do is to figure out, first, we need to 'remove the periods and the commas'. Otherwise, they're going to be attached to the words. I'm going to use 'wordArray = text.split(" "); So, this should pick out each word separately. Now, we also need to remove capital letters. Otherwise, words with capital letters will be different to the same word if it starts with a small letter.

So, that's the first thing that I need you to do is to clean up this string. And then finally split it on the spaces into an array. So, now we've got a long array of words. So, here's that array. So, it's a '(134)' long, and we've removed the commas; we've removed the periods, et cetera. So, once you've done that, now what we need to do is we're going to use an object to store the words. So,

we're going to use key and value pairs. So, the key for each item is the actual word itself. So, here we are. We're going to create 'wordCount'.

It's an object, and the key is the word. And then, we're going to give a value to that word. And if it hasn't appeared before, if it doesn't appear as an item, i.e., it's 'null', we give it a value '1'. If it does appear already, we just increment its value by '1'. Then finally, what I want you to do is to produce an ordered list of those words with the most common word at the top. To do that, you need to use an array. It's best to turn that object 'wordCount' into an array. And this will do that, 'object.entries' will give you an array of arrays.

And in the arrays, there's going to be two things. Let me show you. So, here, for example, is that array. You will see that these are the most common. And we got 93 out of these arrays of two. And so, this array. So, now I want you to sort it's not going to be in the right order. I've already sorted this, but I want you to do that as well. So, good luck with this exercise. And this has got many uses. Basically, the whole of Google's page ordering was based on most common pages that they had to order, just like this.