IIT Madras

ONLINE DEGREE

Modern Application Development II
Online Degree Programme
B. Sc in Programming and Data Science
Diploma Level
Prof. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology- Madras

Javascript Collections - Iterations and Destructuring

(Video Start: 00:14)

Hello everyone welcome to modern application development part two. And look at this new concept called iteration. Now what I am going to do over here is I once again declared now x in this case is this mixed array and I am saying x dot length equal to 5. So, pretty much what we had earlier I am just calling it x instead of y but now I am writing a for loop and let us look at the for loop it is precisely how a for loop in C.

For example would look except that I am you know just directly using a let statement here in order to initialize the index variable i. Now this could this have been a const no because I actually need to modify the value of i as part of the loop which is why for a for loop where I am actually incrementing the index variable like this i need it to be left. But there are other for loops where I can actually use const.

If I am for example just using a off or in iteration we will look at again examples of those later. The bottom line is what I am doing over here is fairly straightforward I am just going from 0 to x dot length i + + and I am printing out that you know this was x of i and I am printing the type of x of i what does it look like when I run this the first value x of 0 essentially is the value is 1 and what is the type number see this was generated by the Javascript interpreter when it ran.

As far as b is concerned it thinks it is of type string as far as a to a + 1 is concerned it thinks it is of type function and the last two values which were created simply because I made x dot length equal to 5 artificially are undefined. The value is undefined the type is undefined. So, both of these I mean essentially it comes out you know saying this is the value. The value is undefined the type is undefined whereas for everything else it is either a number or a string or a function it could also have been an object.

Meaning it could have been an array itself. Now there are a couple of other ways of running the same iteration one of them is using the keyword in. Now notice over here I can run const i in x. So, what does i in x mean? This basically means that let me comment out this other for loop because otherwise we have too much text on the screen yeah. So, when we do this const i in x I am doing exactly the same thing and I run it you will notice a couple of things one is even though x dot length is equal to five it basically did not look at elements three and four.

So, it went only for the 0, 1 and 2 and it I have to in order to get the value I actually need to do x of i in other words this i is actually the index into the array that is being iterated over. So, what happened over here the difference between a for with an explicit index computation and for with i in x is the fact that when you do i in x it goes over the index values but it skips and defines and I can actually declare this as a const because the incrementing of I is being done.

It is not really doing it as an increment of its actually going through and finding the next value it is basically calling the iterator function internally that is what is happening which means that every time I go inside these curly brackets i has got a new block scope which is why I am able to declare it as a const I never have to change its value inside the scope. And finally there is one more way of iterating where I do const i off.

And what is the difference over here look at the code a little carefully I had const i in x and I was printing out x of i whereas when I do const i of x I just print out i and I look at type of i through x in this case it is not skipping the undefined. So I actually need to yeah what happens with const i of x is that the i of x means that it is taking the values of x not the i in x which means it is taking the index values which are present in x.

So, const i of x actually takes the direct value itself and therefore I can use I itself as the value in whatever I am printing or actually computing and I can then print it out it then says 1, b, a each are of number string and function type and then it continues to go over the remaining values in x and says that they are undefined. So, all of these are the basic these are the most basic forms of iteration.

But there is of course you know a little bit more to it in particular we can start looking at something like this which is we could have something where I define x as an object notice the curly brackets. So, because of the curly brackets and the fact that I am giving a colon 1 b colon alpha c colon 3 2 1 and so on x is basically an object it is not directly an array and what are the index values in x it is these keys a b and c it is not 0, 1 and 2.

Now when I do for const i in x I can actually do console dot log x of i and once again it basically goes through and basically prints out I alpha and this entire array 3, 2, 1. Now I could also just print out I in this case and what I would see then is I should see the index values and what are the index values as expected that a b and c. Now instead if I use the off iterator I can do that as well

.

And what do I expect to see it should take. So, what happens in this case is that the const iterator actually fails. Now why is the const iterator failing look at the error message that has come out over here it says that x is not iterable. So, x was defined as an object when I use the const i in x it was able to iterate over it. So, it looks as though x is iterable why is it saying x is not iterable.

Now what is actually happening over here is when I use the for i in x it is not actually iterating over x directly it is of iterating over another sort of parameter of x which is basically the x dot keys. So, there is a function which will extract the key values in x that is iterable and therefore I can run const i in x and it will iterate over the keys in x. Const i of x on the other hand is supposed to iterate directly over the values of x.

And for that it says it is not a trouble because it does not have a concept of 0 1 2 indexing it does not know directly how to find out the values corresponding to those slightly subtle point over here but you know bottom line is it is probably better to avoid these kind of scenarios in any case. Whenever you are writing a for loop the more explicit you can make it what exactly are you iterating over what are the keys what are the values it is easier to understand and maintain in the long run.

Now you can use off if you go with an other way of writing the code where we call explicitly a function called object dot entries. So, this object over here is actually a class

it is a the sort of primitive data type present inside Javascript from which all other data types are derived and that has a function associated with it a method called entries which will basically generate an iterable list of all the keys and values that are present inside the object.

So, what you are doing is we are taking this k, v two values. So, effectively everything that is there in this object.entries it is like a tuple in python it has two values the key and the value and that key, value is assigned first to the first entry in object.entries of x then to the next one then to the next one and here I am only going to log the v that is the value itself.

What happens when I run it as expected it gives me back the values I could add the key as well and then it would print both. So, it would basically tell me that a 1 b alpha and c 3 2 1. So, in other words it is printing both the key as well as the value that comes out of it. So, what is the takeaway over here are objects iterable not directly there are certain methods that can be called such as object dot entries which is sort of effectively implicitly called when I do const i in x.

So, it is safe to use that but if I try doing const i of x I basically hit the error saying that an object by itself is not iterable. So, let us move forward. Now here is something else that we can do we can also use a new syntax over here where we basically say x is new array of five. So, this is new in a couple of different ways we are using a new keyword right literally the new keyword which is trying to create a new object.

And what is this object notice over here this array is something that I had not really used before I mean previously we just use square brackets in order to put the data into the array. Now when I declare array of five like this I am effectively creating a new object with five elements in it and assigning that to x. Now what this has done is that it has created a new object but there is no value in those objects.

So, in other words everything inside the object is undefined but I can go around assigning values to it I can say x of 1 is equal to 10 x of 3 is equal to hello and then I can go through and print out the values I can print the index the value and the type of the value what happens when I run this I find that at index 0 the value is undefined at index

1 which is where I put in x of 1 is equal to 10 I find that the value is 10 and it is of type number.

Automatically the moment I assign the value 10 it updates the type also and says. Now it is a number index 2 still has nothing it is undefined index 3. Now has hello which is of type string and index four is once again undefined. So, effectively what we have done over here is we just created an empty array. So, it was full of holes and just inserted values at some random places at some in a couple of different places inside it leaving the others empty.

So, they continue to remain as holes and remember what we said about the in operator effectively you know how this works? Let us look at the result let me comment out this other part for clarity and when we run this we find that the in operator actually iterates only over the defined values the sort of the values that are where the the indices inside the array where the value is not undefined.

So, in other words only the ones where there is a clearly defined object in place it will take only 1 and 3 and print those values at all. Now there is one thing over here which is relatively simple to describe we are just declaring two arrays x is 1 2 3 and y is declared as 0 and this notation out here with the triple dots dot dot dot x what does dot dot dot x do it is basically called a spreading operator and what a spreading operator does is it takes whatever x is and spreads it out internally.

So, that you get the complete effectively it becomes a larger array with you know all the values directly spread out inside it. So, if I had not got the dot dot dot and if I had it this way what would the result have been I would see that inside I actually retain the square brackets. So, that the second element of y becomes an array and at this point the length of y would be actually three because it would have the first element as 0 the second element as the array 1 2 3 and the third element as 4.

But by doing the spreading just putting in the dot dot dot over here and running it what I find is that. Now y has length 5 you can print that out and check it but y has length of 5 and it has element 0 1 2 3 and 4. This becomes useful like I said in a couple of other

places it is just an interesting piece of syntax. Now there are some interesting like I said you know the functions that operate on functions.

I am just starting by declaring a value an array x and then I am going to declare another array y and say x dot find and what is this inside the parenthesis over here it is a function. This is basically a function that takes this entire x value whatever it is and identifies where x is less than zero. Now the interesting thing is probably I have confused things a little bit by using the same x over here and over here they are unrelated.

So, I could actually change this and make it let us say t and make it t less than 0 because all I am doing is basically declaring a function what do you think is likely to come out of this when I log y I should find those elements in x where the value is less than zero and in fact well it is not exactly doing that it is finding the first element in x where the value is less than zero.

So, find in other words scans through the array from left to right and the moment it finds some value which satisfies this function which has been given to it, it prints that value or it returns that value in this case it got printed because I printed y that is all. Now this is useful if I want to find out for example whether a particular string contains the letter e. So, I can do x dot find of e and yes it should find out whether or not the string is there.

Of course in strings there are probably some slightly more optimized functions that can be used but something similar in the context of an array. Now what if I change this around a little bit let me take out y and instead use another function called filter. Now notice that you know like I said ok I use the t over here but even if I use x over here this x is within the function scope which means that this x that I have inside the parenthesis versus this x to which I am applying the filter are different values are different parameters.

So, please keep that in mind when you are reading code ideally it is probably better to avoid writing code like this where you have the same name both inside the parameter and outside because it can lead to confusion are you using the same x. The interpreter itself is not going to be confused it has very clear scoping rules which tell it that this x is

different from this x. So, it should not have a problem but yeah you know maybe this should have been renamed to something else ok all anyway what happens when we run this filter is sort of like find except that it applies across the whole array.

So, what would it do it would find out all the negative values. So, -2 -4 and -7 get logged as a result of applying the filter. Now there are other more interesting sort of function of functions that we have map is another one what map does is it takes in this function that we have declared applies it to each one of them. So, filter what did it do it applied the function and it gave you only the ones for which that function gave the result true.

So, in other words it helped you to pull out a few of the values that were present inside the function map is something a bit more interesting over here in some sense because what it does is it actually allows you to modify the return values. In this case what I am doing is I am checking the value is it greater than 0 if. So, I return the string + else I return the string - what happens when I run it I get + - + - + + - + depending on whether the values of those corresponding positions were positive or negative.

So, the map has taken each of these values one and converted it to + -2 and converted it to -3 and converted it to + and so on. And continuing on this we now come to something called the reduce function. What does reduce do reduce is something even more advanced than this it actually takes a callback function as well as an initial value. And what it does is in this initial value is first taken as the value of a over here the accumulator value it is called an accumulation.

So, reduce is basically going to take this callback function that I have provided over here and repeatedly apply it over x because I mean after all I am doing x dot reduce. So, this function a, i giving a + i basically it takes two parameters a and i and returns a + i what are those 2 parameters a is coming from within reduce itself it is initialized as 0 and then it keeps on coming from application of reduce.

Whereas i is coming from the array on which the reduce is being implemented. So, what is this going to do it will first take the value 0 it will take the first value from x which is 1 and it will do 0 + 1 and the value that will be returned will be 1. Now the value of a is going to be equal to 1 on the next round through this. So, it will take a is equal to 1 i is

equal to in this case -2 and do 1 - 2 and the result will be -1. So, now a becomes equal to -1. Then -1 + 3 will become + 2 a becomes equal to + 2, 2 -4 will become -2 a becomes -2, -2 + 5 becomes equal to + 3. So, a becomes equal to + 3, 3 + 6 is equal to 9.

So, a becomes 9 9 -7 is equal to 2 a becomes equal to 2 and finally 2 + 8 is equal to 10 we have reached the end of the array that is the final value that should be returned by the reduce and let us see sure enough what happens is that it prints out the value 10. So, once again how was this obtained it takes the 0 as the starting value the accumulator initial value and after that repeatedly applies this function a, i giving a + i to each of the values coming from x.

And after applying it the result is now stored in a you could have done the same thing using a for loop of course. But the point is that writing it in this manner is a very compact and efficient way of expressing your code it is not just about you know using less lines of code. The fact that you are able to give it as a reduce has implications for how a compiler can then interpret it and optimize it for implementation on parallel hardware in some cases that is not done in Javascript.

But in general that is the reason why functions like reduce are used the interesting thing is I could also change this I do not need to have this reduce with just plus I could have another function in this case I am multiplying and then of course if you know if I start multiplying with 0 as the initial value of course the final value is going to be 0. Instead I start with 1 and when I run it I find that you know the product is 1 into 2 into 3 into 4 8 and because there are 3 minus signs over here the final result is negative.

Now there are also a couple of other functions x.sort what does x.sort do this is probably not what you expect right. The first thing that we might expect is of course the negative number should come first but something went wrong I should have seen -7 as the first entry. And the reason why that is not happening is because this is basically doing what is called a lexical sort it is doing a string based sorting.

So, the minus sign of course comes first but after that the order becomes 2 4 7. what happens if I actually wanted to sort them as numbers I have to explicitly tell it to sort it as numbers I basically say x.sort and I pass in the function to use as the comparator over

here. Take two values a and b take the value a - b and if that is less than zero it means a is less than b if it is greater than zero it means a is greater than b and if it is equal to zero it means a is equal to b.

In other words that is what the sort is assuming as the comparison function. The good thing is if you can declare some other function which would do that for a string or an object or anything else you could use the same x.sort on those strings or objects what happens now when we run it this way. Now it knows to treat them as numbers and we get the proper sorted order as we expect.

Now one last part over here which is for the destructuring. Let us look at what we have over here we have declared an array x is equal to 1 2 3 and then I say a, b is equal to x and what do I expect a and b to contain I mean I am now sort of saying you know x had three elements the new array into which I am destructuring x has only two elements let me log a and b separately.

And what happens is a gets the value 1 b gets the value 2 what happens to the 3 that was present in x it is basically ignored. So, destructuring is sort of able to pull out a partial subset of the values. Now you can do this further the same destructuring but now apply it to objects and this is where things really start to get interesting ok. So, I am declaring an object directly declaring it as a person.

And saying that yeah you know this is it has one parameter called first name another one called last name another one called age and another one called the city all of which are basically separated by commas and are present over here. Now let me just print this out directly and see what happens I find that it basically just dumps the entire thing pretty much as I typed it in.

But what I am doing here is something else I am basically saying cons first name colon fn city colon c is equal to person. So, what exactly is this it starts and ends with curly brackets. So, it is like I am creating an object and in this object it has I am basically saying take the first name from person and call it the variable fn and take the city from person and make it variable c which means that at this point if I now log fn and c

remember that I have created an object without any name of its own the object is not named.

I am using it directly to give names to these two variables which means that from that point onwards I can declare I can use fn and I can use c, does that work? Yes sure enough it takes the first name from the person object and c becomes the city what else can I do? I can also directly say const last name colon, h I do not need to rename it and then what will happen is yeah I need to log those values of course. So, that they are visible.

And when I log that it basically says it prints out the last name and it prints out the age. So, in other words this first name colon fn was just if I wanted to rename it if I did not I could directly have used last name itself as a variable from this point onwards. Now you can do something further which is that you can basically say const first name and everything else gets sort of spread into rem.

So, let us look at first what first name looks like and that is pretty much what you know as expected it turns out to be the first name that was present inside the object what's more interesting is rem the remainder is now an object and it has all the remaining keys. So, it has basically taken the first name alone out and it has destructured everything else such that the first name goes out of the object all the others get created into a new object with the remaining keys.

So essentially what we have done over here is this becomes very powerful because you can use this kind of destructuring when you are passing parameters into a function. You could declare the parameter going into a function as an object and inside the function directly sort of say you know I will pick out only the values that I want and if you have any default values you could declare them as well inside the function.

There are techniques for doing that you can look up how this is done in practice for. Now I just want to sort of mention how this can be used.

(Video End: 28:20)