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 - Function methods, Prototypes and Classes

(Video Start: 00:14)

Hello everyone welcome to modern application development part 2. Now in addition to object literals and the object methods that we have already looked at there are also certain sort of derived properties that we can think of and those are usually and the most common versions of those are what are called the get and set getter and setter patterns in some sense and Javascript provides some direct support for that.

Now let us look at the place where something like that would be useful So, we are designing we are basically defining a user object using the let user and saying that it has two literal properties the first having this value and last having another value. And what we would like is that I would like to get the full name of the person and I do not want to have to repeatedly you know keep on taking the first name put a space at the last name and create that.

One way that I could solve this would be to just create a function and then call that function full name. Now of course the problem with that would be that you know first and last would then be literal properties whereas full name would be a function a method not that that is a problem in itself except that Javascript provides something to sort of make life a little bit easier over there.

It essentially says you can define two new functions one is get full you can see that it is a function because it basically has the parenthesis over here. And similarly set full and in that case it actually takes one parameter. So, let us try and understand first what get is going to do. Intuitively you can look at the code and what it does is it takes this dot first adds a space and this dot last and finally creates.

So, we should expect to see Albert space Pinto and sure enough if we comment out the second portion and just run this we can see that user dot full prints out the value Albert Pinto all right. So, that is good now what is the set method right. So, this set full method you will notice that both of them have the same name right. So, in some sense this is sort of like method overloading but ultimately it is a for a very specific purpose.

And what we are saying over here is if I give you a full name how do you extract the first name and the last name and one way by which we can do that which is at least works if a person just has a single first name and last name would be to say split on any space and take the first part and call it first and whatever remains or rather the second part call it last. Now if the person has a first name middle name and last name obviously this will not work.

So, this is a sort of trivial example that we are looking at right. So, now what I am going to do is you will notice that I can straight away use user dot full equal to and give it an assignment. And in particular I do not need to sort of call user dot set full name with a parameter not that that is more difficult except that this is sort of more intuitive and the cleaner syntax. What does it do we expect that it is going to split and it should give us the two parts user dot first and user dot class should be separated out.

And sure enough if I run it I find that it prints out the first name and the last name separately right. So, these get and set properties are examples of what we can call derived properties. In this case the get and the set are specific keywords that permit this to be implemented in this way. Now what we can do in addition to this is if I define an object xx I can also have a function right.

So, for example I can define add directly as a function it is basically an anonymous function that I am creating here taking two parameters x y and in this case what I am saying is it returns x + y + this dot a right. So, it not only just adds two values which are given externally it can also add it to something which is part of the object itself and what happens when I run this is that with 3 + 4 + this dot a has the value 5.

It adds all of them together and gives us the value 12 so far so good. Now what we are going to do is something a bit more a different way of sort of handling functions. First I

will take the xx dot add and assign it to a separate function separate variable let z equal to xx dot add this is just to make it a bit easier to refer to that function. And what I want to do is to show that there is a function by itself has another method defined on it and in this case the one that we are looking at is what's called z dot call.

So, it is the call function called method of a function which means that in principle when I do z dot call I should be invoking the function z which in turn was a method of this object xx. The thing is z dot call in addition to the parameters three and four as before that we had also takes one more parameter the one in front which is basically the equivalent of the this parameter.

In other words whatever I give over here as the first parameter to the z dot call has to tell me in what context I am calling z and basically what happens over here is that I have this. So, this call method which is a method applied to the function. So, in other words over here z is xx dot add z refers to a function z dot call is therefore a method of that function in other words a function applied to the function.

And when I call it, if I was to directly invoke z I would have probably just done z of 3, 4 but in this case I need to call z dot call and when I do that I need to also give it the context within which I am calling it. In other words what should be used as the this parameter the object for which I am calling this particular method which means that now I have just given it an empty string.

I have not given it an object I have basically given it sort of undefined the equivalent let us see what happens when I try running this. I get not a number why is that? Because internally if I look at what add has it does x + y + this dot a . I have not defined what this is in this case this happens to be an empty string. And therefore this dot a is undefined and when I try doing 3 + 4 + undefined the result is nan or not a number.

How do I fix this I need to make sure that I give it the correct this parameter which is basically the object itself. So, z dot call xx, 3, 4 should hopefully give us a correct result and yes it does. Now why is this useful well the way that it is shown this is clearly a contrived example it is not particularly useful in itself. But it can be made useful if we

for example have multiple different functions and we want to be able to dynamically at runtime decide which one of them to call and with which context.

So, most of the time this is probably something you all you do not really want to do you should prefer to either stick to a pure object oriented approach where you just call a method on an object. But this does give you the ability to call a method of an object in the context possibly of another object all right. So, let us go further z dot call was one of the properties z dot apply can also be used.

And basically what it does is I can give it a sequence of arguments a list or an array and what it does is it basically spreads this whatever is given as the extra argument out here which means when I think about it what is going to happen is z dot apply will effectively translate into z dot call of xx, 1, 2. And it will have an extra couple of parameters three and four. And one of the things about Javascript which goes back to the nature of the language it was written in the context of something which needed to be highly forgiving of syntax errors and so on.

The choice was made that if there are extra arguments to a function they will be ignored and if a function does not have sufficient arguments the extra arguments will basically get an undefined value automatically it is not going to complain. And in this case what will happen is when I do z dot apply xx with 1 2 3 4 we can try running it and we basically get the result a which is 1 + 2 + this dot a which is 5.

And these two last values 3 and 4 are basically ignored as being extra parameters. Now there is one more thing that can be done over here which is property using z dot bind. And this is something that is more commonly seen in the context of functional programming again it is an interesting idea it is not absolutely essential that you fully understand it at this point but it is good to know.

Effectively what we are doing is we can say that z was a function z took two parameters x and y let me bind one of those parameters to the value 2. In other words given that z originally had this definition x + y + this dot a effectively what I am going to get is that I now have 2 + y + this dot a where the value 2 has come from the bind this dot a has

come because it was bound to that particular instance of the object for which the this dot a value was 5.

So, as a result when I actually run this and you will notice that I am not using z2 dot call I am just directly invoking z2 as a function what do I get I now have 2 + the parameter 3 which is bound to y now + the this dot a which has the value of 5. So, 2 + 3 + 5 gives me the value 10. Now what has happened over here we took a function z which took two input parameters and converted it into another function z2 which takes only a single input parameter only the y is taken as input.

Now this can be quite a powerful way of creating new functions this is something called a closure and what is happening over here is it has effectively taken this value the bound value 2 and associated it with this instance of the function. So, z2 is now a new function where the two have been bound into the definition of the function itself. So, instead of z2 having a definition x + y + this dot a it now has the definition 2 + y + this dot a.

So, this is once again potentially it can be useful in some contexts. Now we have seen all this about how an object can be created in Javascript. Now comes the question of how do we do inheritance how do we get one object that is derived from another object or that inherits certain properties of another object. And let's start by first creating a simple object by itself right.

So, I define const x is equal to a colon one and inc is a function that does this dot a ++ in other words it will increment the a value out here and what happens is if I just you know print out the value out there all that I will see is it basically has a colon 1 and inc is a function it does not tell you the definition of the function it just tells you that it is a function. Now I am going to do something new right this is where I am defining a new object because after all its defined using these curly brackets.

So, it is an object right. So, y is an object simply because it is defined using curly brackets but the first literal of that is given for y is this thing called underscore underscore proto underscore underscore right. So, this pattern of writing is sometimes called a dunder double underscore. So, you can call it dunder proto right. So, this double under proto or dunder proto whichever we want to call it is defined as x.

And in addition to that there is also one standard object literal b which has the value two. So, now let us see what this means I mean what does y look like ok and we find that y actually seems to have only one parameter b we do not see anything about the proto at all I defined something but it is not showing about here. But now comes the interesting part what this proto does is basically telling us that y is an object.

But the prototype or the sort of template from which y is derived is x which means that any function or any literal parameter that x has y automatically also gets which means that we should find that y has a y dot a parameter. And if we run this we find that yes it does and it has the value 1 which is same as what it was defined for x. What happens if I call y dot inc which is after all a function that was not defined in y but was defined in x which is the prototype for y.

When I run this I find that y dot inc does work because now when I log y dot a I find that the value has increased to 2. So, this is the basis of how Javascript implements classes. It essentially uses this concept of a prototype you just have objects and how do you make one object basically be inheriting from another you just set the prototype to that other object from which you want to inherit.

So, if you want to define something like a base class you first create that as an object and then you create your other objects and just set their prototype to be this base object right. So, there is no there is no separate concept of a class as such now having said all that clearly this approach while reasonably simple to understand and reasonably simple to use can also be a little confusing especially for beginners which is why the newer versions of ECMAScript have added in new syntax which basically allows you to directly define classes.

So, all of this Javascript classes in other words right so a Javascript class could for example be defined in this way it uses the keyword class it then gives the name of the class and within these curly brackets start and end of curly brackets it has some functionality defined inside it. And the thing to keep in mind at some level is that despite the fact that this looks as though it is a whole new syntax all that it is doing under the hood over here is actually implementing the same prototype based inheritance.

So, it makes certain things certain kinds of you know the way that the code is written a bit more clean and easy to understand which is why it is probably recommended that you use this because it definitely makes it easier for others to understand your code when you use this kind of structure. Rather than sort of trying to do prototypes and manage them directly use the class syntax it makes it much easier to understand code.

How it is implemented is using prototypes which mean that it has some of the restrictions of prototypes. For example multiple inheritances possible with some difficulty it is not straight forward but otherwise it pretty much has all the features that you need. So, now that I have defined a class animal over here one of the things that is needed is something called a constructor

So, one of those this class animal even though I have not mentioned it anywhere is itself sort of extending upon a base class called object that is sort of considered the ultimate base class in Javascript everything else extends from object. So, I do not mention that anywhere I have to give something called a constructor and it takes one parameter in this case I have given it a parameter and it takes this dot name and basically sets it to name right.

So, in other words I am creating a class which would have one object variable this dot name and it also has one method called describe which would return dollar this dot name makes a sound dollar does not sound. Now I have not defined this dot sound anywhere So, you might wonder what exactly is this dot sound let us get to that later right. So, I define a new object of this class and that is where the keyword new comes into the picture.

Let x equal new animal and I give it a name because after all the constructor was expecting a name and if I tell it to describe this variable x what happens is that I get Jerry which is the name after all makes a sound this dot sound is undefined therefore I end up getting the value undefined printed over there. Now how do we fix this? Of course you could directly define this dot sound inside the constructor right.

But instead what I am going to do is create a new class and I will call this class dog and say that dog extends animal and it has the same constructor and once again it has a constructor which takes an input name. But what it does with that is it just calls super. So, super is a keyword which refers to the parent class. So, in this case if you basically call the animal constructor with this name.

But now what it does is it also sets this dot sound equal to oh right. So, now if I create a new dog using exactly the same method new dog and give it a name and then describe it what I will see is jerry makes a sound undefined but spike makes a sound go and we can take that further and have yet another class called cat which once again has the same kind of constructor it.

Once again takes this except that it sets the sound to meow in this case now there is one more function out here the static method which I will get to later for the time being the first thing to notice is that if I create a new cat and run this I will find that tom makes a sound meow all right. So, in other words this is how the basics of classes are implemented. Why would you want to implement classes well for the same reasons that you can already sort of see over here.

You could define something which is a basic animal you could define a class called cat and then you could have multiple instances of cat in a more practical scenario maybe you have a class called employee and there are multiple employees of different kinds that could be somebody in sales. There could be a manager there could be an engineer right and each of those is the sort of sub class of employee.

Within the engineering you might have further things which is you know the software related or hardware related or some other parts right. So, you could sort of have things where there is a whole hierarchy of classes built up this is all standard object oriented programming information which we do not really have the capacity to get into in much detail it. Of course it is pretty much exactly the same as what it would be in python or any other object-oriented language that you might have encountered.

(Video End: 20:29)