INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

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 - Identifiers, Expressions, and Variables

(Video Time: 00:14)

Hello everyone welcome to modern application development part 2. Now the next thing that you need to know about Javascript is of course how do you add comments and you can see that just putting like two or more slashes is enough to start a so-called simple comment a single line comment. On the other hand you could also have a multi-line comment which starts with this slash star and ends with star slash.

The extra stars over here or the fact that you know it is split across lines in this way are irrelevant all that matters is the slash star at the beginning and the star slash at the end. These are the two things that really matter when you want to have a multi-line comment. So, unlike in python where you just use the hash symbol in order to comment out a given line and there is no concept of a multi-line comment.

Comments in Javascript are closer to what you would see in something like C++ or Java you could have single line comments like this or multi-line comments that look like this. Let us move further down. We already talked about identifiers and an example of that would be this statement let x = 0. So, x is an identifier over here what is let? let is a reserved word what exactly does let mean, what does const mean and so on.

We will look at that in a little bit more detail later but for the moment it is a reserved word it is not something you can use as a variable identifier. What about const AnotherVariable something else that we declare as var and where it gets really interesting is when we start looking at what is called Unicode. Now you are already familiar with the idea of Unicode all that it says is rather than just restricting ourselves to single byte values in order to represent letters which are sufficient for the ASCII the basic Latin alphabet.

We can also have like more complex characters from different languages by allowing multiple bytes in order to represent different languages. Now most Javascript engines use Unicode but with a particular format. So, Unicode just talks about code points what are the numbers associated with a given character there is also something called the format for representation and that is either utf-8 which is a Unicode transformation format 8-bit which is probably the most popular or utf-16 which uses 16 bit values for the majority of characters or there are a couple of other different ways as well of doing it.

But for whatever reason most Javascript engines actually use utf-16 as the way of representing characters inside them. And the interesting thing about this is that you can see that I have actually got a variable out here right which is in the devanagari script and basically has the word sheher. And as far as the solution for that is concerned or rather the solution the value of that is concerned I have put in the value Chennai in the Tamil script .

So, I am mixing multiple languages here my let keyword remains in English or at least in the Latin alphabet the variable name is in Devanagari the variable value is in the Tamil script what happens when I actually run something like this. Now all these let const, var by themselves do not have do not show up on the console nothing will come only this final console.log is going to generate some output along with of course this first line out here this console.log ("Hello console!"). I am going to leave that up there and it will be permanently on.

Now if I run this what is going to happen, the same the index.html output remains the same hello console from the first line remains the same and now you can see that console.log (Sheher) resulted in Chennai being output. So, there are two parts to this one is it means that you know using different languages in Javascript is looks like it is fairly easy you can assign like different languages to variables strings in different language or different scripts to variables without really having to struggle very much.

You could if you wanted also use a different language to declare your variables in the first place. Now this is something I would sort of strongly recommend against because you need to keep in mind that even though you might be familiar with the language and reading it at the end of the day it is likely to be seen by someone else who may not be able to work with that particular language.

Now you could argue that you know about what about English there may be people who do not understand English, yes but the point is Javascript by its language by its basics requires us to at least be able to recognize these kinds of keywords. You cannot get around that. So, if that is the case then it is probably better to use that same minimal set in order to define your identifiers as far as possible.

This is not sort of set in stone if you decide that you want to have something where you are going to declare your variables in your own language that is not forbidden. This is just more of a convention or a suggestion that you know it may be better to sort of try and use a neutral set of alphabets in order to represent identifiers just to maximize the usability of the code that you are writing.

So, I am going to now comment this part out again. Now the next thing that we need to understand when we are talking about variables is that there is something called scope. And the scope of a variable basically says where is this particular variable visible. In other words if I declare a variable somewhere then is that visible throughout my entire program is it visible to other scripts that are being invoked from the same html file.

So, that scope becomes very important because one of the biggest problems that we can see as far as scope is concerned is if I am not careful with scoping I might end up declaring a variable which overlaps with a variable that has been declared by someone else in some other library. And if I do not have a way of limiting scope this actually makes programming quite hard.

Because you know whoever is creating the library in the first place has to be able to use some variable names and there is no way that they can guarantee that you know no one else will ever use that variable name. What is usually done in such cases is to use an idea of so-called name spaces. So, Javascript does not at this point you know, there are concepts of name spaces you can have sort of modules and so on that you could use in order to restrict what you know how different names are seen into different parts of the code.

But still even within your own code you might find that you want to restrict certain scopes. So, here is one example and what I am showing over here is I have declared one variable x1 = 10 using what is called a var keyword and then I have this just open brackets close brackets it is not a function this is what is called a block. And inside this block I am going to log the value of x1 I want to basically see whether x1 is visible inside the block.

And then I am also going to declare x2 where x2 = 30 and outside the block I will sort of output its value to the console. Let us see whether this you know what happens when I do this. And when I run it I find that it actually prints both. So, the console.log which was there inside the block came out as 10 and the console.log which is outside the block came out as 30 and this is a bit worrying because this var x2 the x2 was defined or declared only inside this block.

What this line 24 is telling me is that even though I declared it inside the block it is now visible outside which means that x2 is now globally present everyone can see it possibly even interfering with other libraries and so on. So, this was problematic and for a long time even this idea of how do we sort of restrict scope over here was quite difficult to handle. Now where I mean I have to sort of qualify that statement a little bit it is not that it can be directly seen by other libraries.

What happens is everything which is inside the same function scope can see this which means that anything at the top level script the entire top level script can see that var x2 even though it is declared inside a block. Now on the other hand what Javascript later the newer versions essentially brought in this thing called let which allows you to do almost the same behaviour you can just use let instead of var.

And by doing that now what happens when we run this is you find that there is a reference error x2 is not defined ok at script.js line 24. So, essentially basically what it is telling you is that this line 24 ran into a problem because x2 is not defined. Why is it not defined because this let means that x2 is defined only within the scope of these brackets. So, in other words the primary difference between let and var is let allows you to have more restrictive scoping.

It easily allows you to create for example something inside a for loop that will not then be seen by everything else outside the for loop. There is another thing which is also called const and the idea of const is very similar in this case const will also work exactly the same way as let did. It will give you exactly the same results and the reason is that you know once again as far as const is concerned it has block level scope.

And basically the block level scope that it has over here basically means that the line 24 console.log of x2 is not going to see this variable x2 that has been declared on line 22. There are other differences between let and cons that we will get to in a moment but this idea of scope is something to keep in mind. General rule of thumb avoid using var unless you have a very specific reason to do so.

let or const should do the job for you even better than that const should be used in most places because basically what it means is that once you declare something using const that variable then cannot have its value changed. So, even if you do accidentally end up sort of reassigning it or something the const will flag it and tell you look there is a problem here. You said it was supposed to be a constant but now you are changing its value.

But if you know for sure that you want to actually have a variable that can change value use let. So, let us comment that out and get on to strings. So, I am going basically going to have this line over here where I just declare let s = hello and then console.log (s) what do I expect it is just going to output this world hello out here. But a string has some additional properties associated with it one of them is called the length of the string I can also pick out a individual element in the string let us say s [0].

and if I run this what I will find is that the length of the string is printed out as 5 and s [0] because just like in python we start with 0 indexing of strings and arrays which we will get to later. The 0th character is h if I change this to 1 I will find that it should print e and there we have e as s [y]. What happens if I instead try to print out a character a sub string. So, a substring let us look at this notation here it basically says s is a string and s.substring.

So, for those who are familiar with python you would immediately recognize that this is sort of a method call and what I am saying is that string has a method in fact this you know the notation that comes up in replit automatically tells you a fair amount of detail about you know what that substring is. And in this case what is going to say is it will start from character number 2 and go on till character number 4 but not including character number 4.

And let us see what that will print out, it basically prints out ll right. So, character number 2 to 4 essentially is this ll part of hello. It does not include character number 4 which is the o itself. Now on top of that what we could do is you know once again I could do something like declaring just going back to the previous example that I had where I said sheher = Chennai and if I look at sheher.length I am going to get the length of this string indicating Chennai.

And that is actually interesting how do you count the length of such a string because if you look at it and count it in Tamil this this is chei and that should be sort of seen as one character this is the in and you then have the nai part of it which should ideally be seen as three characters. But clearly not we can see that you know this part of it is this I well it is actually being shown as a single character you can only select it as a single character.

But the problem is when you finally print out the length you find that it gives the value 6. What this means is that the length of a string is not that clearly defined or it is not that easy to use when we are working with different languages. It is clearly defined I mean you can get it from Unicode but why exactly does it end up being 6, that depends on how exactly it was represented internally. So, let us move forward and look at this concept of something called templates.

So, what is a template as you can see this is basically using the so called backtick operator, the backward code. And any string that starts and ends with a backward code is considered a template string and inside that template string if I have $ bracket and anything which is basically Javascript and close bracket then it gets logged onto the console it will do the interpolation for you.

So, I actually need to have this also uncommented. So, that when I run this what I will find is s also needs to be defined and then when I run this I will find that this first thing out here got interpolated as hello world. And the second string came out as length of well ${sheher} which means that it actually puts in Chennai over here. And I can also compute ${sheher}. length and have that printed out earlier.

So, this is sort of a very simple way by which Javascript allows you to create sort of formatted strings similar to what would be the python formatted strings. Not a full-blown templating language but useful for creating certain simple outputs. Now let us move forward and look at another concept that is useful in Javascript called operators an example of an operator is just something like 3 + 4. So, the plus is the operator out here.

What happens when I run something like this it just basically executes a 3 + 4 and prints out 7. In fact let me get rid of this hello console log out here. Now the next question might happen what happens when you do 3 + 4 where the 3 and the 4 are put within quotes as you might imagine it is sort of going to treat these as strings and what does plus mean in the context of strings, concatenation.

And it is going to take 3 4 and put them together and make a string. Now what if only one of them was a string the 3 was made a string but the 4 is actually given as a number it does something called coercion which means that it converts both into the same type and in this case it converts both into a string. And when I run this I will get again 34 because what it has done is it has taken this 4 converted it into a string and used the string form of plus to combine the two smaller strings together looks good until we see what happens when we try doing star which is multiplication.

Now star is not defined for strings. So, what does Javascript do it does not just complain and say you know I do not know how to do star on strings. It does the opposite conversion and converts both into numbers and you get 12 as the output. Now to me this is just a recipe for confusion I mean there is like no particular reason for converting some of them to strings and some back to numbers when you need to but that is a choice that the makers of Javascript made please avoid using any of this sort in code that actually goes into production because it is just going to confuse somebody and not really be helpful.

Now after this of course comes the question of how do we define equality because some kinds of comparisons are always going to be needed. So, what happens if I just had something like console.log (3 == 4). I should get something that says false perfect and what happens about 3 == 3 very good that gives us true. What about string three equal to number three once again Javascript starts doing coercion and it says it is true once again this is potentially a source of problems.

Please be careful about this because you do not automatically want to convert strings to numbers or vice versa you want that to be in your control you want to know exactly when such a conversion is taking place in which case it is probably better to resort to what's called strict equality which is the triple equals sign. What happens when I run that it says 3 equal to 3 no it is not ok you can go further and you know you have this thing where there are two parameters undefined and null both of which essentially have a so called none type of characteristic.

But if you do a triple equals on them it is going to say they are not equal to each other for the most part unless you have a very specific need triple equals is probably the safer one to use for comparisons. You should be able to use it in most cases unless there is a very specific instance where only the double equals has to be used and then please be aware of what possible pitfalls happen because of that.

(Video End: 19:56)