Introduction to JavaScript

Video Transcript

Video 1 – Introduction to Variables

So, in this exercise, we're going to create our first JavaScript variables. In JavaScript, variables are dynamically typed. Now, what we mean by that is that in JavaScript, there are eight different types. So, for example, we have string, we have number, we have object, and a subcategory of object would be an array, and we have booleans. And there are several other types as well, but those are the main ones that we're going to use in this course. So, x can refer to any of those. But depending on what it's referring to, the behavior in JavaScript may change.

So, for example, especially when we're passing variables into a function. If the variable is a string or a number, then when we pass it into the function, we actually get a copy of it in the function. We don't get the original. The original stays where it is. But if it happens to be an object that's being passed in, then we're passing it what we call by reference. So, we get a copy of the reference to the object, and we're holding that, but that's just like a handle on the object. So, now we have two handles on the object, one outside the function and one inside the function. But both of them allow us to make changes.

So, when we change the object, for example, in a function, we don't have to return it. It's already changed because outside, there's also a handle on it. So, we're dealing with just the one object. So, you need to understand quite carefully the difference between passing a variable by value or passing it by reference. And we're going to go into that and give you a mental model of exactly what's going on in JavaScript when we do that. So, pay attention to variables and pay particular attention to when they go in and out of scope.

And we'll talk about the scope of a variable. Some have what we call block scope. So, if you say, let x equals seven, that x will have just scope within the block. What we mean is that it will only exist within that block. It may disappear outside. Now, if it's a var or a const, it has function scope. So, if we define it, say outside a function, it exists inside that function as well. You can get access to it inside that function, and it exists after the function executes and has disappeared. So, you need to understand the scope of variables and their types because it's critical in JavaScript. So, this is a very important lesson.

Video 2 – First Variables in Console / Programming in Javascript

So, let's start programming JavaScript. I'm in my browser, and I pulled up this page. And now, I can go to an integrated development environment that is built into the Chrome browser. It's not built into other browsers in the same way. But in Chrome, we can get to it on the Mac by saying, Cmd Opt i or on a PC, Ctl Shift i. On both machines, we can use Fn F12 to go there. So, I'm going

to use F12. And it brings up this development environment. And we need the Console tab. The other tabs we'll get to later. But in this tab, we can type JavaScript.

So, for example, let me type 'a = 5;', 'b = 7;', 'c = a + b;'. So, c will be '12'. So, those variables now will stay in memory, and we can reuse them, or we can change their values. So, for example, we could redefine 'a'. Now, often we'll precise the name of the variable to give the compiler a clue about how it behaves. So, 'var a', we could redefine as '7;'. So now, if I type 'c', it doesn't change value. But if I type, 'c = a + b'. So, 'a' now has the value '7', and 'b' has the value '7'. So, 'c' should be '14', yes. So, let's now, I'm going to 'clear( )' this. It just clears the screen.

The variables is still there. If I type 'c', it's still '14'. Now, we can define a function. Let's call it 'add'. And we say, '= function'. And we need to pass in two variables to the function. These are called arguments. And now, the curly brace, we specify what we want to happen. And what we want to do is we want to '{ return' a value that, not a + b, 'x + y;}'. And this curly brace is basically is a block. It's called a block, and it defines something called a scope, which we'll come to. So now, we've got a function call 'add'. If I type 'add', it just tells me what the function is. But if I type, let's suppose I say 'result = add(a, b);'.

Then, let's see what's plus a, can you remember what 'a' was? Well, the 'result' is '14', and 'a' was '7', and 'b' was '7'. So, this shows you how to type in variables, and we've had our first function. Now, if we want to write out the value, we can use something called 'console.log', and if I type '(result)', it will write to the 'console' the value of 'result'. So, to make it a little bit prettier, we could do ' ('Result = ' '. This is a string, and we can add to the string ' + result); ' the variable. And it's smart enough now to be able to print out 'Result = 14', and we'll come back and talk in more detail. But for now, why don't you bring up the Console and experiment with it.

Video 3 – First Array

Let's look at our first array type. So, let suppose we have an array 'a = ' and I'll go with '[1, 2, 3]'. So, it's got three elements. And it actually tells us that. So, when I type 'a', it's going to actually type out '1, 2, 3'. Now, let's suppose I have 'b = a'. The question is, what does this mean? Is 'b' going to get a copy of 'a' or is 'b' going to be the same as 'a'? Let's take a look. So, let's change now 'a[0]', and make it '= -1;' Question is, so 'a' is definitely '[-1, 2, 3]'. 'b', notice it's also '[-1, 2, 3]'. So, let's take a look what's actually happening.

So, this is what's happening. We've got the name A now points to this array [1, 2, 3], and that is an object. And in memory, it gets mapped to three locations. And we have 1 in the first, 2 in the second, and 3 in the third. Now, when we say B = A, it means that B points to exactly the same object here as A. So when we changed A[0], now A[0], A[1], etc., point straight to here, to the memory. So, when we changed A[0] to be -1, this one changed. So, B also then changed. So, that's how arrays work. Let's just go back quickly to our Dev environment, the Console. Let's see if I say 'const c = [1, 2];'. Well, so we have two elements in 'c'.

Now, 'c' is a constant, so that 'c' must always point to the same object. However, 'c[0]', remember, points into that memory and is not constant. So, we can say, 'c[0] = -1', and 'c' will now be '[-1, 2]'. So, be careful when you're using constant that it doesn't mean the elements inside that object can't change. They can. So, if you remember the diagram, it means that we're pointing to this object. So, that's the way I think of it. So, it explains why we can change the elements of something that is supposed to be a constant. The actual elements here are not constant. It's just that this if it's a constant, it must point to this object all the time.

Video 4 – First Object

Okay, so let's take a look at our first object. I'm going to call this 'contacts'. And we'll start off with a simple object, and in it, we're going to have '{name:' so, ' 'fred' ', and an email address. So, the name value is the 'name' before the ':', and the value after. So, the value, say ' 'fred@mit.edu'} ' for email. Okay, so we will just give two elements to this object. So, now we have 'contacts', and we can access the name. So, that would be 'contacts.name', and 'email', and you see, the value is the string. We can also access it by using the [ ]. And for example, ' ['email'] ', and that is "fred@mit.edu". So, they're pointing to the same thing.

So, there's two different ways of accessing this. And in fact, there are two different ways of constructing. So, let's have, let's just call this 'contact' without the s. And this is a different object, but in this case, we're going to construct it a little bit differently. So, let's suppose I have a 'name = 'fred';' and ' 'fred@mit.edu';'. This will just be 'contact', something to, this is totally a different name to 'contacts'. Okay? And this time, we can just put '{name, email}'. So, just by feeding, it knows the value of 'name' is 'fred', and it knows the value of 'email', and it knows that this is an object. And so, if we look at 'contact' now, there we have it. It's exactly the same structure.

So, those are two different ways of constructing objects that lead to very similar state. Now, let's just change one of these elements. Now, we said this is constant, but just like we had with arrays, it doesn't mean that the elements inside the object need to be constant. So, we can say 'contact .name'. So, that's in this element here; I mean this object here, ' = 'anne' '. And if we look at 'contact' now, you'll see it's 'anne', we haven't changed 'email', but we can change 'email', 'contact.email = 'anne@mit.edu';'

Okay, so that's one way of looking at this. Let me clear the screen a moment. And we've still got 'contact'. That's that. How about now I say, 'p = contact'? What's happening now? What do you think the value of 'p' is going to be? So, 'p' now has got the same; it looks the same as 'contact'. The question is, is it the same? And the answer is yes. If I change now, 'p', let me say 'p.name'. And I change it to ' 'amy';', if we look at 'contact', it's also changed. So, let's take a look at our diagrams to see what's happening here.

So, just like with arrays, B if I have an object A, and I put B equal, say, what it means is B points to the same object as A. Now, if, for example, the name gets changed. If I change A.name, B.name will also change. If I change B.name, A.name will also change because they're pointing to the same object. So, just be aware that objects are very different to variables such as integers

or floats, that they behave totally differently, right? So, think of it in terms of these arrows pointing to an object. And the fact that the object is constant doesn't mean that the elements in the object are constant. They can be changed.

Video 5 – Block Scope of Let

So, let's take a look at block scope. So here, we've got a block here defined by these { }. And inside, we've got {let x = 2;}. Now, let is different from var, so above we've got var x; let x has scope within a block. So, what that means is this x will only exist inside these { }. So here, we've got var x = 3. So, we have outside the block x equals 3. Then we go into the block, {let x = 2}. So, inside the block, we have x equal to 2, and anything if we printed out x now console.log(x) inside the block, it would be 2. But then we leave the block.

And so, what happens is it goes out of existence. And console.log, now of (x), will have the value of 3 again because this is the only x in existence, that x has disappeared. Okay, so that's block scope and why we use, and how we use let. So, let's repeat this in the Console. So, 'x = 10;', say, and then we have the block, and we have ' {let x = 2;', And we say, 'console.log('x inside block = ' + x);}'. So, that writes that out, and we'll end the block. And so we see inside the block it's 2, and then we'll do a 'console.log(outside the block x = ' + x);'. And we'll see outside the block it's 10.

Video 6 – Function Call with Argument

Okay, let's step through what happens when a function is called. So here, I'm defining a function f1, that takes an argument (a) and then returns a + 1. And we have a const x = 3. And we're going to call that function f1 with (x). And the return value we're going to store in r. So, at the beginning, we've got this situation where f1 points to this function; x is a global variable, and its value is 3. And we've got r, and it doesn't yet have a value. Now, once we step into the function, we have a situation where x, then its value gets copied into the argument a, so that a is now 3. And we return a + 1.

Now, there's no storage for a + 1, so a temporary variable gets created with the value of a + 1. So, here's that with a value of 4, and that temporary variable now gets copied into r, and we exit the function, and everything inside now disappears. The scope of a is just inside that function, inside those { }, if you like. And now it's gone, and r has the value 4. So, that's what happens when we call a function. Notice that often variables are coming into existence and going out of existence if a variable doesn't have something pointing to it. So, that temporary variable, for example, didn't have anything that was referencing it, it disappears. And I find this very useful for understanding exactly what's going on. So, I hope you do as well.

Video 7 – Function Call with Object for Argument

So, we've seen how to pass a variable like an integer or float into a function. Let's take a look what happens when we pass an object into a function. So, here I've got a function f1, and it takes (a) as an argument, and it assumes that (a) is an object, and it accesses the x slot of a.x and sets it equal to 4. Now, it doesn't return anything, and let's see why it doesn't need to. So, here is the x that we're going to pass in, and it has an x slot and a y slot; x is 3, and y is 4. Now, we fire f1 with (x) as an argument.

So, this is the situation now that we get where a the argument points to the same object as x. And now, a.x reaches into that object, to the x slot, and sets it equal to 4. So, then we come out. We don't have to return anything. So, everything inside a disappears inside the function. But the object that exists outside has the state x equal to 4 now. So, it's updated. So, when we pass objects into a function, we can update them without having to return. Okay, so, why don't you practice passing different kinds of variables and objects and arrays into functions to make sure you understand how they behave.

Video 8 – Pass by Value-Pass by Reference

So, I want to show you the difference between pass by reference and pass by value. So, can you guess which one is pass by value? Let's suppose we pass in an object; is that pass by value? Or if I pass in, for example, an integer. Well, pass by value is an integer because when we pass by value, we take a copy of the contents. So, let's suppose we've got some variable x. We make a copy of whatever x is equal to; maybe it's equal to 7. And the argument here becomes a totally separate copy of cup.

So, this one filled cup here, we're filling a different structure. We've got a copy of it, and we're filling that. And now, if we return it or something, we'll have a full cup as well as an empty cup. Now, when we have an object passed, we're dealing with the same object. The argument here is just a reference to the cup that exists out here. And when we fill it, we're actually filling that cup that exists outside the function.

And this is why we don't need to pass it back by a return because we've already updated it. Now when we have a return, we might return a true or false to tell us whether we've been successful in our operation. But pass by value is for integers, floats, things where when you say, a = 7, b = a; that b has a copy, it's not dealing with the same object at all. Whereas pass by reference is for objects. And here, when you say b = a, b is just a reference to the same object as a. And so, you're operating on the same thing. You're filling the same cup. Okay, I hope that clears up pass by reference and pass by value with a graphic that will help you remember it.

Video 9 - Basic Looping And Looping On Array Exercise

Okay, so let's take a look at looping, how we loop through objects. Well, one way is with a for loop. And what I want to do, I've got this array called 'contacts'. It's got three elements, and each element is an object. I want to add an 'id' into that. And I'll just use, I want to add the 'id' '0' to the first element, '1' to the second, '2' to the third. So, let's write a for loop. So, 'for', and we need a looping variable, and that's 'i'; '(let i', and we need a starting value. And we need an ending value that 'i' must be '< 3;'.

We want to loop from 0, 1, 2, okay? And then we need to have an increment. And we can have 'i = i + 1', but a shorthand for that is just 'i++)'. It just increases the value of i. Now, we have what we want to do, and I'm going to hit Shift+Enter. So, I'm still completing this statement. And what I want to do is I want to get 'contacts', and I want to get '[i]' one. Okay. So, I've got the '[i]' one. Now, I want to add into it an 'id'. So, I can actually just type 'id' and '=' and give it the value.

So, let me give it, for example, the value of 'i;' and let's complete that. And now, let's run that loop. Okay? And now, let's look at 'contacts'. Let's see what we have here. Yup, it's had an 'id' of '0', '1', and '2'. So, it's done exactly what we wanted. So, that's how you can loop, and this 'let' is like var, except it says that 'i' will only exist within this statement. If I try to print out the value of 'i' now, it won't exist.

Video 10 –Loop Over Array And Add To Objects

So, in this exercise, we're going to use arrays, and we're going to have objects in those arrays. So, here we put an array called 'contacts', and it's got three objects in it. The first one starts with 'amy', the next 'fred', and 'anne'. What I want you to do is first change the course that 'anne' is taking. So, here I want this to be ' 'frontend dev' ' instead of ' 'web dev' '. And the second, I want you to add a new contact to the end of this. So, I want to make this array fall along. And you can do it by using ' contacts [3] = ' something. I want you to figure out what that should be and put your name in there and your email. And put your course, which is 'web dev'.

Video 11 – Introduction to Pacman Exercise

In the second exercise called Pacman, we're going to have you control an image of a little creature that used to be in a game environment that would chomp along eating things. And so, we're going to give you two images with the Pacman with his mouth open and with it closed. Then you're going to move it across the screen. Now, when it hits one of your page boundaries, we want you to reverse its direction. So, to do that, you need two more images of the Pacman facing the other way.

And you need to arrange that now you'll use those images cause the Pacman will be chomping away in the other direction. And what we want you to learn here is about how to control what we call the DOM. The browser has its own memory and holds in that memory the webpage that you're

producing. And you need to be able to get to the elements of that webpage. So, you're going to get it at the image tag, and that you're going to change its location. And then you are going to change whatever images that it's pointing to. So, this is a very basic webpage assignment that will give you command of some elements in the DOM.

Video 12 – Pacman Exercise

So, in this kind of fun exercise, we're going to program Pacman. So, Pacman is a video game creature that chumps its way across the screen. Now, what I want you to do is to control how Pacman moves. So, here, Pacman is moving automatically. But now, I want you to reverse his direction when he hits that boundary; when he hits the side of the page, you got to turn him around. So, we've got to figure out what's going on here. Well, what's happening is we're using two different images, one with the mouth open and then one with the mouth closed. So, as he moves and increment across the screen, we flip the images.

So, here he's starting with his mouth open and then mouth closed, and we can make him go faster if we want. So, let's take a look at how images are positioned on the screen. So, here we've got an image, and it's a certain width and height. So, you'll have to deal with that. Images are positioned by their top-left corner with respect to the screen. Now, screen coordinates are measured. The horizontal is as normal X is measured from left to right, but the Y-coordinate is measured from the top-down. So, this top point is 0,0, and then it increases positively, going down. So, this would be, say, Y would be 600 here.

Okay, so we're positioning that point. But now, the image has some width and height, and we need to check; we're going to check when it hits this right-hand boundary; we need to check this coordinate. So that coordinate, we need to add on the width of the image. So, we'll need to deal with that. Okay, so let's take a look at the code. So, here's the code, and we have an image, and the 'width' is ' '200' ' of the image. So, it's 200 by 200. So, we know that. Now, we see that there's a 'style' attribute and its' "position:absolute" ', and that means that we can position the image. So, let's see how we do that, and notice the 'id' of this image is ' "PacMan" '.

So, what we're going to do is we're going to have an array of images, and we'll see how, why we got a two-dimensional array here. Notice this is an array, and inside that array, we have two other arrays. So, we've got an array of arrays, and we need to know how to deal with that. But the first two images, a 'PacMan' going from left to right. So, this is mouth open, mouth shut. This one is going from right to left, mouth open, mouth shut. So, we need to be able to pick the right image. So, I want to show you quickly how two-dimensional arrays work. Here's an array 'A', and inside it, I've got subarrays, '[1, 2], [3, 4]'

And the way to think about this is that this is the first row, the 0th row, and this is the next row. Now, to get at rows, if I want to access a row, it's done by that first bracket. So, 'A[0] [0]' would be one, it would be that guy. 'A [0] [1]' would be the next one, okay? So, we want to get at 1 or 2; then it's, the first bracket will always be 0. Now, if I want to get at the second row, then I have to go to make it 'A[1]'. So, 'A[1] [0]', you can see is '3', and '4' would be 'A[1] [1]'. So, that's how you

get at the various elements of a two-dimensional array. And we'll see that where we positioning the image. Well, this is where we're getting the image source.

So, you'll notice we're getting a hold of the array, and depending on direction and focus, that will define which of these images we're choosing. So, direction is either left to right or right to left. It's either 0 or 1. And the 'focus' again will flip. Notice, here we're doing 'focus', and we're adding '1' to it each time. And then, we're taking the modulus. So, we're dividing by '2' and taking the remainder. So, when it's even when focus is even, then 'focus' will be '0'. When it's odd, 'focus' will be '1'. So, it's flipping between 0 and 1 all the time. This is where we're choosing the direction.

So, position is incremented by '-20' if 'direction' is true. So, 0, as far as boolean go, '0' is not true; it's false. And '+1' is true. So, any plus number is actually is true, but '+1'. So, we're flipping direction between 0 and 1. And if it's 0, this is not true, so it's adding, so it's going from left to right. Okay? And if direction is 1, it is true. And it will be going from right to left. Now, to animate it, you need to call this 'setTimeout'. So, you'll have to look up how setTimeout works. But 'setTimeout' needs to call 'Run( )'. I'll give you that clue, okay? And you can specify '200 millisecs'.

It's measured in a 1000th of a second, okay? So, 1 is a 1000th of a second, 200 is fifth of a second. Now, here we're choosing 'direction' by checking 'PageBounds'. So, this is where we check where the image is and whether the left-hand side is less than 0 or the right-hand side is bigger than your page width. So, you've got to put in here some if statements that will tell you which direction you're going. So, you're coming in with some 'direction', and the question is, do you need to flip it or not? Okay, so that's what you need to program. It should be fun, and at the end, you should have Pacman running across the screen enjoying himself. So, there he goes.