Lesson 1 Video 1 Transcript: Introduction to JavaScript and Hello, World!

Welcome to COMI 1241, Introduction to Object-Oriented Programming.

This course builds on COMI 1150, Programming Concepts, and should help to prepare you for any course that includes an object-oriented programming language. This includes Java, JavaScript (as taught in the Client-Side Scripting class at the time this video was made), Visual Basic, Python (as taught in the Game Programming class at the time this video was made) C#, and C++.

Of the languages just listed, Java is the most object-oriented. However, I am going to use JavaScript for the examples in this class. I have made this decision for two reasons. The first is that JavaScript is fairly easy to get started using. The second is that I want to prepare you for the Client-Side Scripting Class.

So to start, you will need a text editor and a browser. This is because JavaScript runs, among other places, in a web browser. That is how we'll be using it in this class.

You can use any text editor, but I recommend NotePad++. It is free, it is a language-sensitive editor, which means it will highlight your code for you, and it allows you to launch your program, once you've written it, directly in a browser. In these videos you will see me using NotePad++.

What we're going to do in this video is write the canonical first program in a new programming language, which is Hello, world! And I'm not superstitious, but you should always write Hello, world! first.

```html
<html>
<head>
  <title>COMI 1241: Introduction to Object-Oriented Programming</title>
</head>

<body>
  <h3>Assignment here</h3>
  <h3>Your name and date here</h3>
  <p id="output"></p>
  <script language="Javascript" type="text/javascript">
    var outputPara=document.getElementById("output");
    outputPara.innerHTML = "your output here";
  </script>
</body>
</html>
```

What you see here is a template for programs in this class. HTML is not a prerequisite for this class, so don't worry about the HTML portion of this file, which is most of it. You will always be writing code here, between the `<script>` and `</script>` tags. And you will usually be putting your output here, after `outputPara.innerHTML =`, between the quotes, where it says "your output here." You will see that in my
examples. If you need to do something different, I'll tell you what to do. I will also explain this at the end of this video, but don't worry if it doesn't make sense.

So first, let's just run this as is and you can see what the template does in a browser.

So you can see the words "Assignment here," and "Your name and date here," and you can see where they are in our template. What you should do is replace the words "Assignment here" with the assignment, and replace "Your name and date here" with your name and date.

I'm going to replace the first with "Lesson 1: Hello world! Program," and the second with my name and the date.

Now we can also see where it says "your output here," and that's where you're going to put the fruits of your programming labor. In this case, with Hello, world!, we only want to display the words "Hello, world!" to the screen. What we're going to put here is a string. Remember that "string" is the programming word for text, and strings are enclosed in quotes. So we can put any text here and it's going to show up in our browser window where it previously said "your output here."

So, changing that... and then let's run it in a browser.

```html
<html>
<head>
  <title>COMI 1241: Introduction to Object-Oriented Programming</title>
</head>
```
And there it is! Hello world!, in JavaScript.

In subsequent videos in this lesson, I'm going to be reviewing some of the imperative, or non object-oriented, programming concepts from your Programming Concepts class, and showing you how to program those structures in JavaScript. This will include data types and variables, selection statements, repetition statements, and functions.

But now I promised I'd go over the template. Once again, if you don't completely understand this, don't be concerned.
First, let's define HTML and talk about what a tag is. HTML stands for HyperText Mark-up Language. It is not a programming language, as you're not creating something dynamic. A straight HTML page doesn't execute; it doesn't do anything. It's static. HTML is a mark-up language, which means it allows you to mark up a straight text document, identifying the structure of the document by putting tags around the different structural pieces. We can then add some programming with JavaScript.

The tags are enclosed in angle brackets. We always start with the html tag, and note that tags nest, so we always end with the slash html tag. Nested between those tags is the content of your HTML document. An HTML document has two parts -- a head part, nested between the head and slash head tags, and a body part, nested between the body and slash body tags. The head part is information about the document and doesn't appear in the browser window. The body part is the part that appears in the browser window.

So looking at the head part, note the title. That doesn't show up in the browser window. Let's go back and see if we can figure out what the browser does with that. Pause the video and take a look, or program this example yourself, run it, and see if you can find the title. Do you see it up here, in the title bar or grab bar? It's also in the tab, and if I make a bookmark, it's the suggested name of the bookmark.

Let's look at the body part of the document. There are two h3 tags, which stand for "header level 3." Browsers usually render, or draw, header level 1 in such a large size font that I decided to use a smaller header here. And the text between the h3 and slash h3 tags is the text that will be rendered as a header. The original intent of HTML was a document structured as a hierarchy, so the top-level heading of the document, you know, which might be a title, would be an h1, the sub-headings below that would be h2, then sub-headings below that h3, etc.

Within each heading you have paragraphs, and that's what the p tag is for. Notice this paragraph tag has an attribute called id. Attributes allow us to specialize tags. Each tag has its own allowed attributes. And before you take the Client-Side Scripting course, if you take that, you must take an HTML course and be sure to learn your attributes. As far as I know, any tag can have an id, and the id allows me to find the tag using JavaScript programming and then do something with it. So what I'm doing in the JavaScript, between the script tags, is finding that paragraph by using its id, and then using its innerHTML property which is what's between the paragraph tags (this would be the innerHTML of the H3 tags). So what I'm doing is using the innerHTML property to put some text in there. That text becomes the actual paragraph you see rendered by the browser.

Now JavaScripts are programs and they actually execute. When they're embedded in the body of the document like this, they execute as the browser reaches them. They can also be in the head section or in an external file. For now we'll embed in the body of the document. So here I define the paragraph, and then after I've done that I can write a JavaScript that puts some content in the paragraph. This example is silly because we could simply have typed Hello world! directly into our HTML document. But as I show you some programming constructs in subsequent videos, you'll see some more interesting things we can do with JavaScript and the examples will get slightly less silly.
Now that you've written Hello, World!, let's talk about the types of data that you can represent in JavaScript. In your Programming Concepts class, you learned about numeric data that fell into two categories -- continuous, or Real data, which has a decimal part, and discrete, or Integer data, that has no decimal part. You also learned about Boolean data, which has only two states: True or False. And you learned about string data, which is text.

There are also a few other types of data, such as arrays, that we'll talk about later in this course.

In JavaScript there is only one number type, and that is a Real type (usually this is referred to as floating-point in programming). You can also store Integer data in a number type.

Unusual to scripting languages in general is that you don't have to give your variables a type when you declare them. Don't get used to this. Most programming languages are strongly typed and require that, when you declare a variable, you give it both a type and a name. In JavaScript when you declare a variable you will be required only to give it a name.

So let's just look at the three major types of data -- number, string, and Boolean, and we'll look at them in assignment statements, as literals, as variables, and in expressions.

```javascript
<html>
<head>
  <title>COMI 1241: Introduction to Object-Oriented Programming</title>
</head>

<body>
<h3>JavaScript Types Example</h3>
<h3>Maggie Burke, January 1, 2012</h3>
<p id="output"></p>
<script language="Javascript" type="text/javascript">
  //This is a JavaScript comment
  //I'm going to define a list of types using one var statement.
  //The list is separated by commas and terminates with a semicolon.
  //I initialize my variables (give them initial values) as I declare them.
  var num=0, a=5, str="Hello, there!", goodDay = true;

  //And here I'll do some additional manipulations to them:
  num=a*7;
  str+=" Are you having a good day?";
  goodDay=false;

  //Let's print something out to the browser window now:
  var outputPara = document.getElementById("output");
</script>
</body>
</html>
```
So first I'm going to declare a couple of number types, and initialize them with values in the declaration. Get in the habit of declaring your variables this way in JavaScript -- at the top of the program, and initialized with values. Then I declare a string, and finally a Boolean, to which I assign the value true.

Now I do a little processing. These are assignment statements in JavaScript. They are the same as assignment statements in the pseudocode from Programming Concepts, but without the Set keyword. So the variable on the left is taking the value on the right. On the right, in some cases, we have expressions. Note that here I have an expression which is the multiplication of the value in the variable a, which is 5, and the literal number seven. Here I have a special kind of assignment statement. This is a shortcut for \texttt{str = str + \text quotation mark "Are you having a good day?"}, and here the plus sign is the concatenation operator -- it concatenates, or appends, or sticks together, the two strings. Finally I have a straight assignment of a literal value to the Boolean variable, which now contains false.

Let's look at how I print these out now. I want to have some more output, and I want to have some space between the output lines. In HTML, we achieve that space with a line break tag, which is the less than \texttt{br slash greater than} here. So I'm putting that in everywhere I want some vertical space. With my non-string variables, I need to convert them to strings (or cast them to strings) before I print them out. I do this with this dot notation -- the variable name, dot, then toString, open paren close paren. You hopefully recognize this as calling a function. In JavaScript, because it's object oriented, the function is part of these types, the number and boolean types, and so we use the dot notation to call the function. You will learn about that in the second lesson. For now, know that you need to use it in order to convert numbers and Booleans to strings.

Note the comments, also. The commenting syntax is the same as it was for Programming Concepts. Finally, note that JavaScript requires a semicolon after each executable line. It will probably take a little bit of getting used to knowing where to put a semicolon. I'm going to show you a tool that can help, but first, let's go ahead and execute this and see what we get.
Now you are going to need to practice simply writing a program and getting the syntax correct, if you
don't have a lot of programming experience. In Programming Concepts you were not required to write
executable programs. It takes a little bit of practice to get your syntax correct, and the disadvantage to
JavaScript is it's sometimes hard to figure out what your syntax errors are.

I strongly recommend you type in all of the practice examples yourself and get used to the syntax of
JavaScript and get used to correcting syntax errors. You will have to learn to debug, which is a critical
programming skill and a difficult one. You must think very scientifically and record all changes you make
and what you have tested while debugging, or you will send yourself around in circles and you won't
learn why your code doesn't work.

Now I'm going to show you a tool called JS Lint that will help you check your
programs before you run
them. The problem with JS Lint is that it can be pickier than it needs to be. Some errors it will identify
are actual errors, and some are more matters of style.

So I'm going to copy the JavaScript code, the code between the script tags, and visit jslint.com, paste the
JavaScript in, click a few settings, and click the JSLint button.
Now, if I don’t change these two settings, I’ll get some errors that I shouldn’t see -- complaints about indentation and white space, and complaints about using browser objects like document. Here are the settings I’m going to change, "Assume a browser," and "Tolerate messy white space."

And here is the result:
It tells me I have a number of globally defined variables. Remember from Programming Concepts, this means the variables aren't declared within a function or module. JavaScript has only functions. Global data is bad in JavaScript for a very important reason -- there is only one global space for the entire browser session. So if you have more than one JavaScript running, or perhaps some add-ons that are written in JavaScript, then you might have two scripts that use the same global variable name, and this can cause serious problems. So in the future we will be avoiding global variables as much as possible. For now, we won't worry about that.

So I'm going to make a real error for JSLint to find, and I can do it right here in JSLint. I'm going to remove the assignment operator altogether from this statement:

```javascript
str " Are you having a good day?"
```

Now when I press the button I get some errors, and in case I didn't know they were errors, I'm alerted by the pink color of the box and the word "Error:"

```
Error:
    Problem at line 9 character 9: Expected an assignment or function call and instead saw an expression.
    str " Are you having a good day?"
    Problem at line 9 character 12: Expected ';' and instead saw ' Are you having a good day?'.
    str " Are you having a good day?"
    Problem at line 9 character 14: Expected an assignment or function call and instead saw an expression.
    str " Are you having a good day?"
    Problem at line 9 character 43: Expected ';' and instead saw 'goodDay'.
    str " Are you having a good day?"
```

I get "Expected an assignment or function call and instead saw an expression," and "Expected ';', and instead saw ' Are you having a good day?'." So pretty good error messages, if you think about how a parser is going through this and trying to make sense of it. It expects only so many things following a variable name -- either a semicolon or an assignment statement. Pretty much anything else is illegal syntax, so it suggests to you that you meant one or the other of the legal constructs.
You know that programming isn't very interesting when it's sequential flow of control only -- essentially a straight execution path from top to bottom. When it gets interesting is when we add some Boolean logic and some selection and repetition statements. Getting some input from the user doesn't hurt, either. So I'm going to show you how to do all of those things in JavaScript now.

First, let's get some input from the user. We'll do this with the browser's built in dialog box capability. There are three kinds of dialog boxes, but to get input we'll use the prompt dialog box, which will allow the user to type something in.

So I'm going to create a new file from my old file, renaming it and changing the title text, and I'll just add a prompt dialog box in there and print out what the user types in to the screen. Please note that anything the user types in is going to be a string, and if we need it converted to another format, such as number format, we will have to perform that conversion ourselves.

First let me declare a variable to hold what the user types in, and I will put that at the top with the other variable declarations:

```javascript
var userStr="";
```

Now I'm going to do this at the very end, just to distinguish it from the other code:

```javascript
//Now let me get a string from the user:
userStr=window.prompt("What is your name?");

//And print that out to the screen:
outputPara.innerHTML += "<br /> Hello, "+userStr+"!";
```

And let's execute that. Notice that it executes the rest of the script before the prompt comes up. I didn't specify some default text, which I could have specified in the argument list after the prompt, so the default text comes up as undefined. That's pretty ugly and I should probably change that to an empty string.
I type in my name and I click OK.

Let's just change that line of code so we don't get "undefined" in our dialog box:
//Now let me get a string from the user:
userStr=window.prompt("What is your name?","");

And that works much more nicely.

So now that we can get input from the user, we can do some more interesting things. Let's add some control structures to this.

If/else statements are pretty straightforward. As you know, we'll need a Boolean expression or variable, and we can execute different lines of code depending on the value of that Boolean expression or variable.

We happen to have a Boolean variable, so let's just use that.

```javascript
if (goodDay==true) {
  outputPara.innerHTML += "<br />"+"It is a good day. :)";
} else {
  outputPara.innerHTML += "<br />"+"It is not a good day. :(");
}
```

Now, some notes about this syntax. Note that I’ve put the expression in parentheses. You need to do that in JavaScript.

Remember the test for equality is two equals signs. JSLint is going to suggest that we use three equals signs here, which is called a test for strict equality. I'm going to suggest that you use it as well. The equivalent not equals operator is !==. Because JavaScript is not strongly typed, we can compare two expressions that aren't of the same type, and sometimes we will get surprising results. For example, false==0 is true, and ""==0 is true. We probably don't want or expect that result. What the test for strict equality does is will only return true if the two expressions are of the same type and contain the same value. So we will use tests for strict equality in this class, which will make our code more precise and it will make JSLint happy. It's generally considered a JavaScript good practice.

Now, the next thing is we use these curly braces. I don't strictly need them here, because I have only one statement after the if and test and only one statement after the else. You need curly braces, you must use them, if you have more than one statement that should execute after the if or after the else. Enclosing them in curly braces tells JavaScript they're all still conditionally executed. But we are going to
use them even with only one statement, because then if we happen to add a statement, we won’t worry about forgetting the curly braces. That is also a good practice. They don’t hurt and they can help.

So this line of code executes when goodDay===true evaluates to true:

   outputPara.innerHTML += "<br "/>"+"It is a good day. ;)";

And this line of code executes when the complement of that is true, when goodDay!==true is true, or when goodDay===true is false:

   outputPara.innerHTML =+ "<br "/>"+"It is not a good day. :("

And that’s about it for if/else statements. You can add an else if in if you like. Why don’t I do one of those so you can see what it looks like. And I’ll add that here and I’ll pause and come back with that added in.

   //Now let me get a string from the user:
   userStr=window.prompt("What is your name?","");

   //And print that out to the screen:
   if (userStr=="Maggie") {
       outputPara.innerHTML += "<br "/>Hello to my programmer!";}
   else if (userStr=="Brendan Eich") {
       outputPara.innerHTML += "<br "/>Hello to my creator!!!";}
   else {
       outputPara.innerHTML += "<br "/>Oh, hello, "+userStr+". *yawn*";
   }

And if I execute that, you can see that there are three possible paths. Typing in Maggie to the prompt will give me the additional line "Hello to my programmer." Typing in Brendan Eich, the creator of JavaScript, will add the line "Hello to my creator," and any other name (or word) will add the line, "Oh, hello, <name>," followed by a yawn. It’s a pandering program. Let’s look at that executing.

Let's look at repetition structures now. And I'm going to show you two -- one counting loop structure and one conditional loop structure.

The counting loop structure is a for loop structure, and this syntax is the same syntax you're going to see in most programming languages that are based on C. You'll recall that with a counting loop, you have all of the loop control information in the first line. The best use for a counting loop is when you know in advance in your program how many times the loop will execute.

So, for example, I can ask the user how friendly she wants the program to be, and then print "Hello" that number of times. This counts as knowing in advance because the information will be stored in a variable in the program in advance of the loop executing.
So this will also give us an opportunity to get a number from the user. It will be a string initially, so we'll have to convert it to a number. This is a tool that you'll be able to use in your programs.

I think what I'll do is return to our Hello, World! program and modify that, so you don't have to look at a lot of irrelevant code and can focus on the loop.

```javascript
<script language="Javascript" type="text/javascript">
    var howFriendly, howFriendlyStr,i;
    var outputPara = document.getElementById("output");
    howFriendlyStr=window.prompt("How many times would you like me to greet you?","5");
    howFriendly=parseInt(howFriendlyStr);
    for (i=0; i<howFriendly; i++) {
        outputPara.innerHTML += "\nHello world!";
    }
</script>
```

So first, let's think about our variables. We're going to ask the user how many times she wants us to greet her, so we'll need a string to store her response. Then we're going to have to convert that to a number, so we'll need a number to store the converted string. Let's write that code. I'm going to put a default value of 5 in so the user hopefully gets the idea that I want a number here.

Now I'm going to convert that, and since we loop a discrete number of times (there are no fractional loops), I'm going to convert it to an integer. If I want to convert to a floating-point type, I use parseFloat. But here I'll use parseInt.

So in a counting loop, remember that we have the initializer, the updater, and the test all on that one line. This syntax will look a little unfamiliar, but if you get used to it, then you'll be prepared for counting loops in other C-like languages.

The keyword for, and then in parentheses, we start with initializing the loop control variable. And that reminds me I haven't declared one, so I'm going to go back up to the top and put an i in there. i is a typical loop control variable and you can think of it as standing for iterator. Now I can start i at 0, like this, and then a semicolon, and then I put the test. I'll test for i less than the value the user put in. Since it's starting at 0, that means my loop will execute exactly the number of times the user entered. Finally I need an updater, and I'm going to increment i by one each time. We write that as i++. That is a shortcut for i=i+1.

Now as with the if/else statement, I use curly braces to delimit the loop body code. What I'm going to do is just print out Hello world! the number of times indicated, so once each time through the loop. I'd better use += so I don't lose the output from previous iterations of the loop, and I'll put a line break in there so they're on separate lines.

If I run it and stick with the default of five:
So that's a counting loop. Let's explode that into a conditional, or while loop.

With a conditional loop, we're in charge of writing the initializer and the updater in appropriate places. The initializer I can take care of when I declare the variable. The updater has to be within the loop, or else we'll have an infinite loop. And if I change the keyword for to while, I'm left with the test.

Let's execute that. This time we'll use a number other than the default. I'll enter four.
So this syntax is not very different from what you've seen in Programming Concepts. The for loop has
the initializer, test, and updater on the same line. The while loop has only the test, so you must give the
loop control variable an initial value before the loop, and you must update it in the loop body. The loop
body, in both cases, is marked by curly braces.

Lesson 1 Video 4 Transcript: Functions

Recall from before that JSLint noted my global variables. We generally do not want global data in a
JavaScript program, because we are sharing that global name space with other scripts (written by other
programmers) that might be running in the browser. If I have a variable called i, it's possible that
another program will also have a variable called i, and this could cause one or both of our scripts to
break.

The way to avoid global variables is to define your variables, as much as possible, within functions and
using the var keyword as we have done here.

So returning to our Hello, world! program, I'm going to put the code in some functions.

```javascript
//Greet the user with "Hello world!" in the outputPara paragraph, numTimes times
//Input: numTimes (number)
//Output: (no return value)
function greet(numTimes) {
  var i=0;
  var outputPara = document.getElementById("output");
  while (i<numTimes) {
    outputPara.innerHTML += "<br />Hello world!";
    i++;
  }
}

//Get input from the user and convert to a number
//Input: (no input parameters)
//Output: the value entered by the user, converted to a number
function getNumTimes() {
  var howFriendly, howFriendlyStr;
  howFriendlyStr=window.prompt("How many times would you like me to greet you?","5");
  howFriendly=parseInt(howFriendlyStr);
  return howFriendly;
}

//Greet the user by getting the number of times s/he wants to be greeted,
//then printing a greeting to the outputPara paragraph
//Input: (no input parameters)
//Output: (no return value)
```
function greetTheUser() {
    var howManyTimes=getNumTimes();
    greet(howManyTimes);
}

greetTheUser();

Recall from your Programming Concepts class that modules can take both pass-by-value and pass-by-reference parameters, and they don't evaluate to a result. Functions can take only pass-by-value parameters, and they evaluate to a result.

In JavaScript, there are no modules. And your functions don't have to evaluate to, or return, a result. But they can.

So I'm going to break this up, using functional decomposition, into three pieces. The first will be the main piece, which I'm going to call greetTheUser. And this function will not take any inputs and it won't return any outputs. So it's actually very much like the main module that is in many examples in the book. In this greetTheUser function I'm going to invoke getNumTimes, which will be this part -- the part that gets the number of times the user wants to be greeted and converts that string to a number type. And getNumTimes will return, or evaluate to, that number. So I'll store that in a variable in greetTheUser. Then I'll pass that number to another function that I will call greet, and greet will display to the output paragraph the number of times passed in. So it doesn't evaluate to anything, but how many times it prints "Hello world!" will depend on the value being passed in as a pass-by-value parameter.

So you can see that the syntax of a function is the keyword function. There is no return type, as you learned in your pseudocode and as you'll see in other languages such as Java, because JavaScript isn't strongly typed. In another language we would put the keyword "void" as a return type for functions that don't return anything, and we would put a numeric type for getNumTimes. But we don't do that here. So the keyword function, and then the name of the function, and then a parameter list. If there are no parameters, then we put an empty parameter list.

And this will come as no surprise, I'm sure -- we use curly brackets around the body of the function.

Now notice with my code now, there are absolutely no global variables. All of the variables are defined within functions, and I simply invoke my function greetTheUser, the function that is the equivalent of a main function, right here. This is good programming practice. When I run this, it runs exactly as before. (A little too exactly.) But there are no global variables, although technically the functions are, in JavaScript, global variables. But we won't worry about that!

It is very important that you practice with everything I've shown you in these videos, as object-oriented programming is layered on top of these concepts. Object-oriented programming is a way to organize your program, very similar to functional decomposition. With object-oriented programming, the data...
and functions that work on the data are organized into structures called objects. And we will begin learning about that in the next lesson.

Lesson 1 Video 4 Transcript: Debugging

In the process of creating the video for the selection and repetition structures for lesson 1, I had a bug. And so I wanted to show you a little bit about the process of debugging and also to show you what caused this bug that I had. So what I was writing was this code at the bottom which is an if/else if/else structure, down here.

Okay, so let me run that for you with the bug, so you can see what it does. All right, and already you can see on the screen it hasn't printed out what it was supposed to print. Okay, "Not a number, it is not a good day" is the output I'm getting, and that's not the output that I should be getting. Let's go back and look at the program.

And you can see up here where I'm putting the output out, num contains the value 35, that should've printed out, and then the content of string which is "Hello there, are you having a good day" and then because goodDay is not equal to true, it should print out "It is not a good day." Each of these on a separate line. But we weren't seeing that.

So something was causing that error. Then of course it prints out "Hello to my programmer" because I typed in "Maggie." Okay, and that's my new structure at the bottom. So what's going on? So, one thing that you can do is run it in JSLint, but unfortunately, JSLint did not see the problem for me. It did not give me any errors.

So the next thing that you can do is look at your variables and one way you can look at the values in your variables is by using alerts. So what I thought was maybe there's a problem with num, since I'm getting Not a Number, so I decided to put alerts in my program, and an alert is just going to print out an alert box which you just then dismiss. And generally what you want to do is to put a string in, which explains where you are in the program. So right here was after the initialization of num, and then I'm going to put the value that's in num and because that's a number type of course I have to convert it to a string before I print it out. So I'll let you see what the program looks like with that alert in it. Okay. So after the initialization of num and it contains the value zero, as we would expect.

Okay, but still, my program is not executing correctly. Why is it giving me that output? So what I did is I put alerts in all through the code, at all different stages, to examine that value of num, and not only did I look at the value of num but I also looked at the value of output para's innerHTML, several times, both of those variables, to see what was going on in my program and it was always what I expected. It was always 35, and including the innerHTML, had 35 in it, and now when I ran this in Firefox, I could actually see 35 being printed to the screen and then replaced with Not a Number, I didn't see that in Internet Explorer. So it was very puzzling.

So the next thing I did was I started editing the program from a point where I knew it was working. Deleted all of the new stuff. And that's when I found this equals plus. Instead of plus equals. When I added that line back in. And probably just I pressed shift at the wrong time because equals and plus are both on the same key and I've done a lot of incorrect combinations of plus equals. As far as I know that's not JavaScript syntax, so I'm not sure why JSLint didn't pick up on it.
But you can watch it executing properly now that I've fixed that to plus equals. See I already have the correct output there, and one thing to note is that I was not seeing that correct output even though it happened before the wrong code. I never got to see it. So that made it very difficult for me to debug.

It's important when you're following these steps, when you're starting from correct code, that you add absolutely only one thing at a time so that you know exactly what code it is that isn't working. It's very important that you do this scientifically and methodically. It's very tempting to just go in and make a lot of changes, and then once you've done that, make guesses and make changes, once you do that a few times, you don't know what code is doing what in your program.

So, very methodically, start from a working program, add in the lines of code, or add alerts in so that you can examine the values in the variables at different points in execution, and be sure to write a little string in there that tells you where you are in the execution when you're looking at that value. And both of these methods should help you to debug any bugs that you have while you're writing JavaScript.