

JavaScript III:

Java Hard with a Vengance

Lesson Plan

Insert lesson description here. Ex: A student with no prior knowledge of technology learns how to use a mouse and keyboard to open a web browser and practice.

Lesson Objectives

At the end of the class, the student will:

* Understand what a conditional is and be able to write an if/else if/else conditional
* Understand and use the basic comparison and logical operators in JavaScript
* Be able to write a while and a for loop
* Be able to use break and continue in a loop
* Understand the concept of an array and be able to iterate through an array using a for loop

Lesson Prep Work

(30 min, at a minimum, prior to student arrival)

* get in early to test for technology failure, because it will happen :-)
* pre-sign into accounts
* pre-load videos or web demos
* pre-save example documents in J: drive, etc.
* print handouts

Lesson Prerequisites

● It is recommended that students have some knowledge of HTML – it makes later lessons easier.

Lesson Outline

The lesson is completed in one 120 minute class session.

*(x)* Introduction

* Introduce instructor, students.
  + Ask students at introduction: Have you used JavaScript before? Why do you want to learn JS?
* Let students know it’s okay to take phone calls, but ask them to put their phone on vibrate and answer calls outside the classroom.
* Inform students that they can sit back and watch if the class is too advanced.
* Inform students they can go to the bathroom, they don’t need permission.
* Show order in which class will happen. Explain scope of class.

*(x)* Review

* Go over terms from class two
  + *Activity:* look at slide 2 of slideshow. Briefly review
    - Variables
    - Arrays
    - Conditionals
    - Loops
    - Functions

(x) Objects

* What are they?
  + *Explanation*
    - (slide 3) Much of what we’ve been working with thus far have been primitive values. One piece of data.
    - To make full programs, we need to be able to describe more complicated entities - objects. These allow us to describe things that have both data and behavior, and allow us to group data together in meaningful ways.
    - For example – say you’re designing a sales system. You may have an object representing the customer, one representing the product, etc.
      * Each of these objects will have things they do again and again – put something in a shopping cart, or change the inventory amount.
      * Objects let you define all the things that these objects can do, and all the data that you need to know about them, and then each time you need to update that data or invoke a certain behavior, you can refer back to the object rather than rewriting it each time.
    - (slide 4) Objects are declared with our var keyword, and then, on the other side of the equal sign, we have curly brackets.
      * We then have the name of the property or method, a colon, and the value assigned to that property or method. Each of these is separated by a comma.
        + Watch out for dangling commas, which can break older versions of IE.
      * Objects can hold properties and methods
        + A property is a piece of data about an object
        + A method is just a function (we know what those are!) that lives inside an object.
  + *Activity: Have class make an object*
    - Step 1 – Have class open jsbin.com
      * Clicking on the bar at the top of the window, close everything but the JS and Console windows
      * *Explanation:*
        + on the left side is where we write our instructions, and on the right hand side is our console, which shows us the results.
    - Step 2 – Ask class to make an object with their name. Add properties for “gender,” “age,” and a method called “sayHi” which says hi. For example:  
      var nate = {  
       gender: male,  
       age: 37,  
       sayHi: function() {  
       console.log(‘Hey there!’);  
       }  
      };
* Using objects
  + *Explanation*
    - (slide 5) To pull out a particular property or method from an object, we can use dot notation: the name of the object, a dot, and then the name of the property.
    - Note that we still need the argument parentheses at the end of our method name, just like we would use when calling a function.
  + *Activity*
    - Have class log the value of the age property  
      console.log(nate.age);
    - Have class run the sayHi method.  
      nate.sayHi();
  + *Explanation*
    - (Slide 6) We can also use dot notation to change the value of a property, add new properties, or delete them.
  + *Activity*
    - Have the class add a new property, favColor, to their object.  
      nate.favColor = ‘blue’;
    - Log the value of favColor to the console.
    - Change the value of favColor and log it again.  
      nate.favColor = ‘orange’;
    - Delete the favColor property, if you want.
  + *Explanation*
    - Objects can do everything we’ve done with primitive data types. (slide 7) We can make arrays filled with objects, and (slide 8) pass objects as arguments into functions.
    - (slide 9) Now that we’ve learned about dot notation with objects, let’s take a look at something we’ve been using since day one – console.log.
      * The console is an object created by the browser – the JS debugging console. Like any other object, it has its own methods and properties.
      * Log is one of those methods – it writes something to the console.
    - There are lots of other built-in objects. Some of them are available through web APIs (<https://developer.mozilla.org/en-US/docs/Web/API/Reference>), like the console, and some are standard objects built into JS (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects>). We won’t get into many of the API ones, but the standard objects are super useful – they define what an object or an array or a string is, and provide methods like length or parseInt (which converts a value into an integer)
  + *Activity*
    - (slide 11) Ask the class to create an object to which holds their favorite recipe. It should have properties for a title, number of servings, and an array which holds all the ingredients. It should also contain a method named printRecipe, which logs the entire recipe to the console. The answer will look something like this:  
      var recipe = {

title: "French Toast",

servings: 3,

ingredients: ["bread", "eggs", "milk", "cinnamon"],

printRecipe: function() {

console.log('title: ' + recipe.title);

console.log('servings: ' + recipe.servings);

console.log('ingredients:');

for (var i=0; i < recipe.ingredients.length; i++){

var item = recipe.ingredients[i];

console.log(recipe.ingredients[i]);

}

}

};

recipe.printRecipe();

* + - (slide 12) have students create an array named readingList, which contains a series of objects. Each object is a book with a title, author, and a Boolean named readYet. Add at least three items to the array.
    - Then create a function which iterates through the array and alerts the title and author.
    - Then add conditionals to change the alert depending on whether you’ve read the book or not.
* The DOM
  + Explanation
    - (slide 13) To be able to manipulate webpages, we have to understand how HTML, JavaScript, and the browser interact.
    - When browser receives an HTML file, it reads through and builds up a new object (the kind we just learned about) called “document”.
      * The document object has a tree structure, made up of all the elements on the page. Each step on the DOM tree is called a “node”.
    - (slide 14) Open up sample page at <http://bit.ly/DOMsample> (or <http://jsbin.com/penah/1/edit?html,js,output>)
      * So, this is a very simple page. The way the DOM is created, sibling elements become branches – so…
      * We start with the html element, which contains everything else.
      * At the next level, <head> and <body> are siblings – they’re not nested inside each other – so the DOM branches off.
      * Continue to trace out HTML and match to DOM chart on slide.
    - (slide 15) So what does this mean for us? If we want to use JS to change or affect a webpage, we have to know how to tell JS where to find an element on the page.
    - Our general strategy for this will be:
      * Find an element in the DOM and create a new var to store its location.
      * Then we can use that var to change that element’s attributes, style, content, or add new nodes to it.
  + Activity
    - (slide 16) the first step to finding an element is starting with the document object. We can then use a couple of different methods of the document object to find the element we want.
    - One way to do this is to head up and down the DOM using relationships – parent or child or sibling.
    - For instance, let’s find the heading:
      * First, let’s create a var to save the element’s location. In JS window, create a new var (var header).
      * Then, after the equal sign, we’re going to start with “document”, so JS knows which object to look in.
      * The “body” is a node of document – and can be referenced by the “body”, so add a .body to your document
      * Looking at the DOM tree, the h1 is a child of body, so we could, theoretically, do .firstChild. Add .firstChild to document.body.
      * Now that we’ve stored it, let’s change it.
        + Have class write “header.innerHTML = “I did it!”;
        + Nothing will happen.
      * Well, that didn’t work. Try adding nextSibling onto the document.body.firstChild
        + It worked! What is happening?
        + (slide 17) Well, it turns out that our DOM tree here is a little idealized. Most browsers will read blank space in your HTML as a “text node” in the DOM. So – the first child of body is actually a text node that holds the new line and tab before our h1. The h1 is our second child.
      * *Teacher’s Tip:* if there’s time, and you think class can handle it, you can show them document.body.childNodes, and iterate through the children changing the innerHTML to “this is node number” + i.
  + Explanation
    - The key point is – you can access the DOM this way, but it’s confusing and can take a lot of typing.
    - (slide 18) Another option is to find elements by id or class
  + Activity
    - So let’s find the img this way –
    - Have class clear JS, write “var pic = document.getElementById(‘mainpicture’);
    - Now have them change an attribute, like pic.width = ‘50’
  + Explanation
    - Much easier! We can also find things by class.
    - The important thing to remember is that classes can appear multiple times on a page. So if we use getElementsByClassName (notice the plural “elements”), we’re going to get multiple hits. What datatype holds multiple things?
      * Arrays! Searching by class produces an array. So to change a specific member of that array, we have to refer to its index in the array in bracket notation.
  + Activity
    - So let’s use the classes on the li’s for an example. Have class write: var places = document.getElementsByClassName(‘placename’);
    - To show what we can do with it, write a loop that replaces the innerHTML of each li with its index number:   
      for (i=0; i<places.length; i++){  
      places[i].innerHTML=’this is item ‘ + i;}
  + Explanation
    - (slide 19) Pretty cool! We can do the same thing by tag name as well – again, because there will (probably) be multiple occurrences of the same tag on a page, getElementsByTagName will return an array – and you’ll then have to pick out an individual in that array from their index, if you need to.
    - (slide 20) We already saw that you can change HTML attributes (remember the width on the img?). We can call out any HTML attribute with dot notation and then change it in JS.
      * If time, try changing img src to something else.
    - (slide 21) We’ve also seen innerHTML – one thing we didn’t note is that you can use HTML tags in the innerHTML statement – so you can inject any HTML you need directly into the page.
      * Just be careful how you use it – innerHTML overwrites any HTML that’s currently inside an element.
      * As an example, write document.body.innerHTML = "holy crap I erased the whole page!" and watch everything disappear
    - (slide 22) Of course, having access to the document means that we can add new things as well as delete or overwrite them. The process is a little more complicated:
      * We first create a new element – an img or p, for example – with createElement.
      * If that element also has a text node – a p needs text to really be a paragraph, after all – we createTextNode
      * We then append these to a node in the DOM.
  + Activity
    - (slide 23) So let’s see this in action. We’re going to add a new image and a new paragraph.
      * Let’s start by identifying where we want to add these on to.
      * We’re just going to add these elements at the end of the page – they won’t be children of anything, so we can just append them onto the body.
      * So we first start by saving the location of “body” in a var:   
        var pageNode = document.body;
      * Now to add the image. We first have to create an img element – we’re going to store that new element in another var so the browser can remember it until we can get to appending it.  
        var newImg = document.createElement(‘img’);
      * For those of you who remember your HTML, just creating an img tag won’t actually add picture – we have to specify a source, or where the browser can find the picture. Now that we’ve created the var newImg, we can use dot notation to change attributes of that element:  
        newImg.src = 'http://placekitten.com/g/200/200';
      * Now that we have the img and the src, let’s actually add the new element to page. We do this with the appendChild method. We start with the var we’re attaching the child to, and then put the name of the child inside parentheses:  
        pageNode.appendChild(newImg);
      * (slide 24) We’re going to follow the same procedure to add a new paragraph. We’ve already saved the body – the parent node – in a var, so we don’t have to repeat that, so let’s create a new element:  
        var newParagraph = document.createElement(‘p’);
      * The difference here is that we need to add text to our element. To do that, we use the createTextNode method.  
        var paragraphText = document.createTextNode(‘Kittens are a close second to coffee.’);
      * Now we have to append the text node to the element, and then append the element to the body:  
        newParagraph.appendChild(paragraphText);  
        pageNode.appendChild(newParagraph);
  + Activity
    - Solo activity time! Let’s pull together everything we’ve learned in the last three classes.
    - Have class use the DOM example page we’ve been working with.
    - Ask them to add a button to the HTML with the following markup:  
      <button onclick="changeLi()">Click me to change the list</button>
    - Ask them to create an array named myPlaces and fill it with three places that they would drink coffee.
    - Then create a function called changeLi that finds the li elements on the page and then loops through them and replaces them with the places in their array.
    - Give the class time to try and work through it themselves – after 5-10 minutes, work through the problem as a class. One solution:  
      var myPlaces = ['Puerto Rico', 'huddled under the table', 'sitting atop the Mars rover'];

function changeLi() {

var listItems = document.getElementsByTagName('li');

for(i = 0; i < listItems.length; i++) {

var listItem = listItems[i];

listItem.innerHTML = myPlaces[i];

}

}

**(x)** **Conclusion**

* Go over handout, review material, and emphasize contact info & further resources on handout.
* Any questions? Final comments?
* Remind patrons to practice; assign take-home-practice - remind them they can ask for help
* Remind to take survey.

Appendix

What This Lesson Does Not Cover

* [Insert items that are not covered in this lesson, and specify why it is not]

Key Decisions

1. I’ve decided to use jsbin in class, largely because you can set the text size without having to zoom in the whole screen. There are many other tools you can use, including Codepen and repl.it.

Links

[INSERT LINKS TO RESEARCH, LINKS, NOTES, AND REFERENCES USED TO CREATE LESSON PLAN, INTENTIONALLY OMITTED, OR POTENTIALLY ADDED TO LESSON IN THE FUTURE]