

JavaScript II:

JavaScript Harder

Lesson Plan

In this class we will cover conditionals, loops, and arrays.

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
* print handouts

Lesson Prerequisites

● It is recommended that students have some knowledge of HTML and CSS and have attended JavaScript I.

Lesson Outline

The lesson is completed in one 120 minute class session.

*(5)* Introduction

* Introduce instructor, students.
  + Ask students at introduction: Did anyone have a chance to practice since last week? What problems did you run into?
* 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.

*(10)* Review

* Go over terms from class one
  + *Activity:* look at slide 2 of slideshow together. Have students identify:
    - Statement
    - Variable
    - Expression
    - Operator
    - Function
    - Argument
    - You should then point out the data types and review the concept of scope. Point out that tipPercent has local scope, whereas billTotal has global scope

(60) Conditionals

* If statements
  + *Explanation*
    - With JavaScript, much of what we’ll be doing is building things that respond to the user. To do that effectively, we have to make use of conditionals. Conditionals are groups of statements that are only executed if a certain condition is true
    - The syntax is simple:
      * If
      * Then parentheses – inside the parentheses is the condition we are testing.
      * Then curly brackets – if the condition is true, then whatever we put inside the curly brackets will be run.
  + *Activity: Have class write a simple condition*
    - 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 – On the left side, type in   
      var cupsCoffee = 4;  
      if (cupsCoffee > 3) {  
       console.log(“You’ve had too much coffee!”);  
      }
    - Step 3 – Click the “run” button on the top left of the left hand side.
    - The class should all see “You’ve had too much coffee” show up in the console. If they don’t, have them check their code against yours and try it again.
    - Have the class change the value of cupsCoffee to less than 3 and run again.
* Comparisons
  + *Explanation*
    - We just used a comparison as a condition – if the variable cupsCoffee is greater than 3, then do X. Let’s see some other comparisons we can make:
      * === Equality
      * !== Inequality
      * > Greater than
      * >= Greater than or equal to
      * < Less than
      * <= Less than or equal to
    - Be sure to not confuse = with == or ===.
      * = is assignment – the value of X is Y.
      * == is comparison of value - X is equal to Y
      * === is comparison of value and data type – X is equal to and the same data type as Y.
      * In general, we’re going to use the strict equality (===), because, in general, we don’t want to mix up the string “8” and the number 8.
    - *Activity:* have class experiment with different comparison operators in sample conditional.
* Logical Operators
  + *Explanation*
    - (slide 5) Often we’ll want to check if more than one thing is true or false. We can check multiple conditions by combining them with an “and” (&&) or an “or” (||). We can also negate something by putting an exclamation mark in front of it.
    - (slide 6) && is only true if both sides are true. || is true if either side is true.
      * Keeping this in mind, JS will read your condition from left to right and stop evaluating your condition as soon as it has an answer.
        + This means that, if you have an &&, if the first condition is false, JS won’t read the second condition, or
        + If you have an || and the first condition is true, JS will, again, not read the second condition.
        + This is important to keep in mind if you want to make sure that one of your conditions is always checked – place it first.
  + *Activity:* (slide 7)play with logical operators in our sample conditional.
    - Step 1: have students edit our current conditional until it looks like this:  
      var cupsCoffee = 4;

var sleep = 6;

babyCrying = false;

if ((cupsCoffee<=0 && sleep<8) || babyCrying===true) {

console.log(“I may not be in the best mood.”)

}

* + - Step 2: ask students to say out loud, in normal English, what the current condition is. The answer should be “if I haven’t had coffee and less than 8 hours of sleep, or the baby is crying, then I may not be in the best mood.”
      * Ask them to look at the current variable values and figure out if what’s inside the curly brackets will be run, i.e. if I’m in a bad mood.
        + Answer: The condition is false. I’ve had less than 8 hours of sleep, but I’ve had more than enough coffee, and those are part of an “and” statement. For an “and” to be true, both parts have to be true.
        + Run the program in the console to check your answer.
      * Change the variable babyCrying to “true”. Will the condition be true or false?
        + Answer: true: I’ve had too little sleep, but enough coffee, so the part on the left is false. The right-hand part is true, though, and it’s part of an “or” statement. For an “or” to be true, only one part has to be true. So we should see the statement run in the console.
        + Run the program in the console to check your answer.
      * Have students play around a little with different variable values.
* Truthiness/falsiness
  + *Explanation*
    - (slide 8) We don’t have to use a comparison operator in our conditional. We could just provide a value. JS then checks to see if that value is “true” or “false”. This case is often referred to as “truthy” or “falsey”, though, because it isn’t necessarily intuitive.
  + *Activity*
    - This makes sense if something is a Boolean –
      * On screen, write:  
        var coffeeRules = true;

if (coffeeRules) {

console.log('Yay coffee!');

}

* + - * If we run it, we see “Yay coffee!” in the console, because I’ve defined the variable coffeeRules as true.
    - What happens if I just put a number?
      * Demonstrate – it should still read as true.
    - Or a text string? *Teachers tip:* If you really want to blow their minds, give coffeeRules the value of ‘false’ (in quotation marks). This makes it a string, which is evaluated as true.
      * Demonstrate – still true.
    - But what is false? Show the class that an empty string, 0, null, an undefined variable, and, of course, false are all evaluated as false.
* If … else if … else + prompt & alert
  + *Explanation*
    - (slide 9) We’re often going to want to have more than one action – or a default action that happens if our condition isn’t true. That’s where “else” comes in.
    - In this case, let’s try prompting the user for input. We use prompt() to do that – simply write prompt, and then the question that you’ll be asking the user in between parentheses and quotation marks.
    - alert() works the same way, but it just creates a pop-up, without the text box for user input.
    - The else{} simply follows the last curly bracket of our if. We don’t need to specify a condition for else, because it’s a catch-all. An if … else statement can be read in plain English as, “If this is true, do this. But *everything else*, do this.”
  + *Activity*
    - Have class write up what’s on board:  
      var cupsOfCoffee = prompt('How many cups of coffee have you had today?');  
      if (cupsOfCoffee > 4) {

alert('Slow down there, champ!');

} else {

alert('You need another cup of coffee!');

}

* + - Have class run their program with different answers.
    - (slide 10) What if we want multiple conditions? We can use else if.
    - Have class invent another condition to add to example.
    - *Teacher’s Tip:* Students will probably try to write an condition and not make it exclusive – i.e., if x > 4 // else if x > 8 – and get confused when they can’t trigger the second condition. Remind them of && and || and that computers are dumb: if something satisfies the first condition, the browser will run any actions associated with that condition and exit.
  + *Activity*
    - (slide 12) Give class 10-15 minutes to attempt exercise on their own: Write a function named helloWorld that:
      * takes 1 argument, a language code (e.g. "es", "de", "en")
      * returns "Hello, World" for the given language in an alert, for at least 3 languages (try using Google Translate). It should default to returning English.
      * Prompt the user for a language code and then call the function using the user's input.
      * BONUS: alert an error message if the user has entered something that isn't one of your language codes.
    - One solution (with bonus):  
      function helloWorld(lang) {

if (lang != 'fr' && lang != 'en' && lang != 'es'){

alert('I am sorry, that is not a valid code.');

} else if (lang === 'fr') {

alert('Bonjour tout le monde');

} else if (lang == 'es') {

alert('Hola, Mundo');

} else {

alert('Hello, World');

}

}

var code = prompt('Enter two-digit language code:');

helloWorld(code);

1. Loops
   * *Explanation*
     + (slide 13) Loops let us repeat a piece of code as many times as we want – why would we want to do this?
     + Timer count downs, search results, adding images to a slideshow, repeating a question until a user gives a valid answer, etc.
     + There are two types of loops: while and for.
     + (slide 14) While()
       - We set the condition we want to check inside the parentheses. *While* that condition is true, JS will run the code inside the curly brackets and then return to the top to check the condition again.
   * *Activity*
     + - Look at following loop:  
         var coffee=6;

while (coffee > 0) {

alert('I’ve got ' + coffee + ' cups of coffee!');

coffee--;

}

* + - * Let’s walk through how this code would run:
      * We set up a variable named coffee.
      * When the browser gets to this line of code, it checks to see if the value of coffee is greater than 0. It is – coffee is 6, so we’ll run the code inside the brackets.
      * We’re going to alert “I’ve got 6 cups of coffee” (review concatenation if necessary) and move on to the next line.
      * What does the “--" mean? This is the decrement operator. coffee-- is a shorthand for “coffee = coffee-1”. Each time this line runs, we’re going to subtract 1 from coffee.
        + What would happen if we didn’t have this decrement here?
        + The loop would run FOREVER (coffee would always be greater than 0) and it would crash our browser.
      * Now we loop back up to the top and check our condition (coffee=5), so it’s still greater than 0, so we run the loop again.
      * Repeat a couple steps this way, then have class write loop and test it out. People could even try to write an infinite loop – just make sure they’ve saved whatever they need. If using JSbin, it shouldn’t crash the browser – they have built-in protection against that sort of stuff.
        + IMPORTANT: To get loops to run in JSbin, you have to place //noprotect at the top of your code. Be sure to warn class that you’re taking the training wheels off, so they can crash their browser now…
  + *Explanation*
    - (slide 15) Another type of loop is the for loop. The for loop has all of our conditions up at the top. If we rewrite our while loop as a for loop, it would look like this:  
      for (coffee=6; coffee > 0; coffee--) {

alert('I still have ' + coffee + ' cups of coffee!');

}

* + - There are three parts in the parentheses up top:
      * We initialize a counter variable (for loops often just use “i" for this variable), then a semicolon
      * We give the condition under which the loop will continue to run. In this case, for as long as “coffee” is greater than 0, execute the loop. Then a semicolon
      * The last part is our update – whether we increment or decrement the counter each time.
    - *Teacher’s Tip*: if using JSbin, you’ll have to add //noprotect to your JS to get it to run the loop.
  + *Activity*
    - Have students type in and run example loop.
* Break
  + *Explanation*
    - (slide 16) Ask class to describe what this loop does:  
      for (x = 100; x < 200; x++) {

console.log(x);

if (x % 7 === 0) {

console.log(‘Found it!’);

break;

}

}

* + - * It goes from 100 to 200 and logs each number. If it finds a number that is divisible by 7, it says “found it!”. i.e., it’s looking for the first number divisible by 7 after 100.
    - Two things to note –
      * First, here we have an if in our loop, so we can change the behavior on each iteration of the loop with conditionals, which is pretty sweet.
      * Second, we have a “break” here. Break lets you exit from a loop – in this case, once we’ve found the first number divisible by 7, we don’t want to go any further. This “break” exits the loop once we’ve done what we want to.
    - What’s another way we could have done this without a break? An example:  
      for (x = 100; x % 7 != 0; x++) {

console.log(x);

if ((x+1) % 7 === 0) {

console.log('Found it! ' + (x+1) + ' is divisible by 7!');

}

}

* Continue
  + *Explanation*
    - (slide 17)Sometimes we’ll want to be able to skip an iteration, or make one iteration do something special. We can use continue to do that.
  + *Activity –* practice using loops and conditionals.
    - (slide 19) Ask class to write a loop that iterates from 1 to 20, and logs whether each number is even or odd. Give them 10 minutes to try and figure it out on their own. One possible solution:

for (var i = 0; i <= 20; i++) {

if (i % 2 === 0) {

console.log(i + ' is even');

} else {

console.log(i + ' is odd');

}

}

1. Arrays
   * *Explanation*
     + (slide 20) The variables we’ve used thus far have only held one value. Occasionally we’re going to want to store a list or group of things – for that, we have a datatype called an array.
     + The syntax starts off the same as any other variable – var name = - but the values that make up the array are inside square brackets and separated by commas.
     + Think of an array as a series of boxes. Each box can hold any datatype – string, number, even another array! – and the boxes are all numbered. The first box is 0 (because computers start counting at 0), the second is 1, etc.
     + We can use the .length method to find out how many items are in an array.
   * *Activity*
     + Have students make an array called favoriteThings that holds three strings.
       - Ex: var favoriteThings = ['coffee', 'punk rock', 'beer'];
     + Have them log the length of the array to the console:
       - console.log(favoriteThings.length)
   * *Explanation*
     + (slide 21) We can access any value in an array using bracket notation. Write the name of the array, then the index of the value you want inside square brackets.
   * *Activity*
     + Have class log the value of index 1 from their array:
       - console.log(favoriteThings[1]);
   * *Explanation:*
     + (slide 22) We can also use bracket notation to change the values in or add new values to an array.
       - Just how we can reassign a variable by writing its name, an equal sign, and a new value, we can do the same with an index in an array by writing the array name with index, equals, and the new value.
       - If the index value I use isn’t full, I’ll just add a new value to the array.
       - If I want to add a new value to an array, but I don’t know how many items are in the array, I can use the push method to add a new array item at the end of the list.
   * *Activity*
     + Have students change value of the item at index 0.
     + Have students add a new item using both bracket notation and the push method.
   * *Explanation*
     + We can use a for loop to easily loop through an array and access each index in turn. To do this, we need to put together a few of the things we just learned: the .length method and bracket notation.
   * *Activity*
     + Working together, create a for loop which logs each item in their array.
       - for (i=0; i < favoriteThings.length; i++){  
          console.log(favoriteThings[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

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

Teaching Materials: http://www.teaching-materials.org/javascript/

Oz Girl Develop It: http://cathylill.net/courses/js101/

Girl Develop It: http://www.girldevelopit.com/materials