Kevin Chen

3/18/17

JavaScript Notes

**Getting Started**

Introduction to JavaScript

* One of the 3 core languages of web pages (in addition to HTML and CSS)
  + HTML = content, CSS = presentation, JavaScript = behavior
* A scripting language
  + Only works in a web page within a web browser as opposed to in the OS.
  + Can’t access local files, directly access a database, or access hardware
* JavaScript is a client-side language: it is sent to the user’s computer and runs there
  + This is in contrast to server-side technologies (e.g. PHP, ASP.NET, Ruby on Rails), which runs the code on the web server.
  + Thus, JavaScript can be disabled.
  + JavaScript could also be rendered differently depending on the browser, but this isn’t that much of an issue anymore.
* JavaScript has been popping up as a scripting language in other applications and server-side products (e.g. Adobe Acrobat, Photoshop, Node.js, Google Apps Script)
* Reference guide: https://developer.mozilla.org/en/JavaScript/Reference

Creating your first JavaScript

* Use opening and closing <script> tags to define JavaScript.
* Create your “Hello World” program:
  + Put alert(“Hello, world!”); inside a set of script tags at the end of the HTML body.

Getting to know the tools and applications

* Use whatever text editor you want to develop JavaScript. There is no single platform.
* Firefox has great JavaScript debugging tools. Get the Firebug extension.

**Core JavaScript Syntax**

Understanding the structure of JavaScript code

* JavaScript is an interpreted language, not a compiled language. It is not compiled, converted into machine code, and run on the OS. Instead, the web browser takes care of interpreting it and running it on the machine.
* JavaScript is case sensitive.
* JavaScript consists of statements, each often written by itself on one line.
  + Each statement ends with a semicolon.
  + JavaScript may be forgiving if you leave the semicolon, but don’t leave it off.
  + JavaScript is whitespace insensitive.
* Comments: //, /\* \*/
* JavaScript is executed as soon as the browser sees the code. So the positions of the scripts matter.

Where to write your JavaScript

* In-line code (putting JavaScript statements inside the HTML file) is only good for testing
* Put the JavaScript statements a separate file like you do for CSS. File extension is .js.
  + Don’t copy over the <script> tags; just copy over the statements. Use the src attribute of the <script> tag to specify which JavaScript file to run there. Can use relative link, absolute link, or even on another server.
  + Often you also put the type attribute inside the script tag with value “text/javascript” to indicate this is not just script, but JavaScript. But if you leave it off, the browser will assume it is JavaScript in HTML5.
* Often you can see the JavaScript in the head of the HTML, but this could cause a delay in rendering the page snice JavaScript is parsed and interpreted as soon the browser hits it.
  + This sometimes is okay (e.g. using JavaScript to create the content of the page), but generally, you want it at the end of the body.

Creating Variables

* Syntax: var *name*; Example: var year;
  + Can consist of letters, numbers, the underscore, and dollar sign. Can’t begin with numbers.
* Initialize it: var year = 2011;
  + Technically, var isn’t required but don’t omit this.
  + This means that if you misspell a variable, a new one is created instead of JavaScript giving you an error.
  + Initializing many variables: var year = 2011, month = 10, day = 31;
* JavaScript is a weakly typed language
  + Don’t declare a type. The type that a variable holds onto can change. (e.g. you can assign a variable to a number, than a boolean, then a string.
  + Use double quotes or single quotes for strings

Working with conditional code

* if (condition) { /\* code to execute in this code block \*/ }
  + Curly braces can be omitted if you have just one line of code.

Working with Operators

* +, -, \*, /, %
* =, +=, -=. \*=, /=, %=, ++, --
  + Note that assignment returns true, so don’t use this in an if-statement.
  + prefix vs. postfix for the ++ and -- is the same as Java.
* <, <=, >, >=, ==, === (strict equality operator: value and type much match), !=, !==
  + For example, 5 and “5” are considered the same for ==, but not for ===
* &&, ||
* Ternary operator: condition ? value1 : value2;
  + If condition is true, value1 is returned. Otherwise, value2 is returned.

Sending messages to the console

* Use console.log(*stringmessage*) to display a message to the console with Firebug.
* This is not part of the JavaScript language; it’s a Firebug add-on.
* Can also use console.debug, console.info, console.warn, and console.error

Working with loops

* while, do-while, for-loop. break, continue.

Creating functions

* Syntax:

function *nameOfFunction*(*parameters, separated, by, commas*) {

// Block of code to execute;

// Optional return statement. Function does not have to be defined differently.

}

* Parameters DON’T have “var” in front of them.
* Calling the function: *nameOfFunction*();
* It doesn’t officially matter where you put your functions, but best practice to define them before you call them.
* If you specify too many parameters when calling a function, JavaScript ignores the extra parameters on the right end. If too few are provided, the missing ones on the right side will be passed in as undefined.
* Scope: local variables vs. global variables
  + Local variables can’t be accessed outside of a function, but they can be accessed outside of everything else (loops, if-statements, and so on)!

**Types and Objects**

Working with arrays

* Initialize with square brackets.

var multipleValues = [];

multipleValues[0] = 1;

multipleValues[1] = “String”; // Arrays can store element of different types.

* Or can initialize them all during declaration: var multipleValues = [1, “String”];
* Or can initialize them using Array objects: var multipleValues = new Array();
  + Or you can omit the “new”
* Arrays are dynamic (can change size).
* Use dot to access the properties and methods of an object, just like in Java.
  + Arrays have length property, reverse() method, join() method (join the values in an array together and out them as a single string), sort() method.

Working with numbers

* All JavaScript numbers are 64-bit floating point numbers. Don’t worry about int, etc.
* Implicit conversion from int to string: 5 + “5” = “55”
* Expressions that don’t make sense result in NaN. Example: 5 \* “b”
  + isNaN is a function that accepts a variable and returns if the variable is NaN.
* (Number(*variable*) converts the *variable* into the number and returns the number, or returns NaN if unsuccessful)
* The Math object
  + Math.round(*variable*), Math.max(*variables, separated, by commas*), Math.min(*variables, separated, by commas*), etc.

Working with Strings

* Can’t have single quotes directly in a string surrounded by single quotes. Same for double quotes.
* Escape characters:
* String properties/methods: length, toUpperCase(), toLowerCase(), split(*delimiter*) (takes a string, splits it based on the delimiter, and returns an array), indexOf(*string*) (finds the first occurrence of *string* and returns the index of it, or -1 if not found), lastIndexOf(*string*)
* Obtaining parts of a string:
  + slice(*beginIndex*, *endIndex*) takes the string between *beginIndex* (inclusive) and *endIndex* (exclusive).
  + substring(*beginIndex*, *endIndex*) is just like slice.
  + substr(*start*, *length*) uses length instead of *endIndex*.
* Comparison
  + Use double equals sign to see if they are equal to each other
  + Can also use greater than or less than sign, which is based on ASCII alphabetical order.

Working with Dates

* Create a new Date object creates an object with the current date and time.
  + Or pass in your own year, month (0 to 11), and day
  + Or pass in your own year, month, day, hours, minutes, seconds
* Stored as the number of milliseconds since January 1, 1970.
* getMonth (returns 0-11), getFullYear (YYYY), getDate() (1-31 day of month), getDay() (gets day of week, 0-6 with 0 = Sunday), getHours() (0-23), getTime() (milliseconds since 1/1/1970)
* Equivalent setter methods
* Note that comparing objects directly (using == or ===) is done by memory location, just like Java (are the same objects?). For Dates, you can compare using the getTime() values.

Working with objects

* First way to create an object:
  + First create a generic object container: var *objectName* = new Object();
  + Then specify all of its attributes. Example: player1.score = 10;
* Second way to create an object (shortcut):
  + var *objectName* = {*property1*: *value1*, *property2*: *value2*, … };
* To create object methods, first define a function normally, and then assign the function to a property of the object. Example:

var player1 = { name: “Fred”, Score: 10000, rank: 1};

function playerDetails() {

console.log(this.name + “has a rank of: ” + this.rank + “ and a score of ” + this.score);

}

player1.logDetails = playerDetails(); // Sets playerDetails() as a member function.

player1.logDetails(); // Calls the function

* + Notice of the use of the keyword “this” to access the object that this method is a member of.
* Can test if a function exists or not: *objectName*.*functionName* will return if the function exists or not.

**Understanding the Document Object Model**

What is the DOM?

* Allows us to reach from JavaScript into a web page, and vice versa.
* What it means:
  + Document: the web page.
  + Object: the elements/components/individual pieces of the document
  + Model: the tree-diagram representation of all the objects in the document. Consists of a set of standards we use to refer to each of the objects.

Nodes and elements

* Three main types of nodes: elements (type 1), attributes (type 2), and text (type 3).
  + Each element node can consist of zero-to-many other element nodes, zero-to-many attribute nodes, and up to one text node.

**Working with the Document Object Model**

Accessing DOM elements

* Recall that the id attribute in a HTML tag is unique, whereas the class attribute is not.
* document.getElementById(*stringID*) returns the element with the given ID
* document.getElementsByTagName(*stringTag*) returns all the elements of the given tag in an array format.
* Some properties of nodes: nodeType, innerHTML (the HTML between the start and end tags), childNodes
* The “document” in document.getElementByID and document.getElementsByTagName can be replaced with an element to search for just within the subtree with the subroot being that element. (document itself is an element: the root element)

Changing DOM elements

* First, you need to get the element before you can change it
* *elementObject*.getAttribute(*stringAttribute*) gets an attribute by the specified name.
* *elementObject*.setAttribute(*attribute*, *value*). If the attribute does not exist, then it creates a new attribute.
* Setting the innerHTML property: *elementObject*.innerHTML = *text*
  + This is good for small tags. For larger ones, you want to be able to create new tags

Creating DOM elements

* First you create the element, and then you add it to the document
* Create the element: var *elementObject* = document.createElement(*string type*);
  + To add text (inner HTML), you can use the innerHTML property. But a better way is to do var *textObject* = document.createTextNode(*stringText*); and then append *textObject* as the child to the new *elementObject* (look below).
* Add the element to the document:
  + As a child: *parentNode*.appendChild(*newElementObject*); This will add the element as the last child of the parent.
  + Inserting before: *parentNode*.insertBefore(*newElementObject*, *existingElement*); This will add the element as the child before the existing element.

**Working with Events and Event Listeners**

Introduction to JavaScript event handling

* Events are already defined (built in JavaScript); the only thing you have to worry about is which events do you care about?
  + You write the event handler/listener
* Method #1: write JavaScript directly in HTML, using the onclick attribute for a particular element and the value being JavaScript code.
  + Problems: script is mixed in with HTML, code can’t be reused, can’t make this friendly if JavaScript is disabled.
* Method #2: *elementObject*.*eventType* = function() { /\* event handler code here /\* };
  + Example: window.onload = function() { };
  + This is using anonymous functions.
  + Note that there is a semicolon at the end. This has nothing to do with putting a semicolon at the end of a function; it has to do with the fact that this whole thing is a statement (which consists of an assignment, just like var a = 5;).
* Method #3: document.addEventListener(*stringEvent*, *function*, false);
  + *stringEvent* omits the “on”. So onload would instead be just load.
  + *function* is the function that runs when this event occurs.
  + document can be replaced with a single element object if you don’t want to add multiple listeners.
  + Equivalent for removeEventListener
  + Benefit: lots of flexibility to dynamically add and remove event listeners
  + Drawback: difference between the browsers. Prior to IE 9, IE does not have addEventListeners; its equivalent is attachEvent. Thus, often you have to first check if *elementName*.addEventListener returns true before using the method.
  + Best to use jQuery (or another library) instead to write cross-browser code.

Working with onClick and onLoad events

* Generally speaking, putting the reference to the JavaScript at the end of the HTML file is okay, but it won’t work if not everything has already been loaded yet.
  + For example, if you are adding an onClick event to an image but the image hasn’t been loaded yet, the event will not be added.
* Use the window.onload event to know when everything has been loaded.
  + The window object is actually one level higher than the document object.
  + Put all the JavaScript code inside the window.onload event.
  + You should have only one window.onload event and function. If you write it multiple times, only the last one will be paid attention to.

Working with onBlur and onFocus events

* Useful for forms
* Every form element has an onFocus event that is triggered when you click/tab into it. When you leave it, the element gets the onBlur event.
* Use these two events to set a default value into the form field, or to make that default value disappear when you click/tab into it.

Working with timers

* setTimeout(*function*, *milliseconds*) runs the function after the specified amount of time in milliseconds.
* setInterval(*function*, *milliseconds*) does the same thing except it repeats it.
* clearTimeout(*timer*) and clearInterval(*timer*) stops the timer or interval from counting.
  + Pass in the timer by obtaining a reference to the timer by storing the return value of setTimeout or setInterval.

**Debugging JavaScript**

Common JavaScript errors

* Syntax errors
* Calling a non-existent function (typos with DOM methods, using a non-existent object method)
* Using document.getElementById before the element is part of the DOM (before the page is loaded). No error is created; it just doesn’t work
* Assignment instead of equality (= vs. ==)
* Missing parameters in function calls

Using Firebug

* Can view the DOM or navigate to the DOM using HTML
* “View Source” gives you the old version of the HTML of the page (before JavaScript runs), whereas the Firebug debugger tool gives you the most recent version
* Script tab shows the JavaScript used by the page: lets you do JavaScript debugging

Going through a debugging session

* Use of breakpoints:
  + Rerun: jump back in time and cause an event to happen again
  + Continue: go again till you hit another breakpoint
  + Step Into: step into the function (or step over if there is no function)
  + Step over: run the next line of code
  + Step out: get out of the current function call and go to the function that called the function
* The “watch window” in the right shows variables
* Can cause a breakpoint to stop running the code only if a condition is true (do this by right clicking the breakpoint).

**Building Smarter Forms**

Accessing form elements

* Can use the ID property: document.getElementById(*stringID*)
* Can use the name property to get the form: document.forms.*formName*
  + Can also get individual elements within the form: *form*.*fieldName*.
* Textfields
  + Main Properties: value
  + Main Events: onFocus, onBlur, onChange, onKeyPress, onKeyDown, onKeyUp
* Checkboxes and radio buttons
  + Main Properties: checked
  + Main Events: onClick, onCheck
* Select lists
  + Main Properties: type (it’s either select-one or select-multiple since certain lists allow you to choose one or mulitple), selectedIndex (for select-one), options[*index*].selected (for select-multiple)
  + Main Events: onChange
* Form
  + Main Event: onSubmit (when the submit button is pressed). You can interrupt the event before it actually gets sent to the server. You can even prevent the form from submitting by returning false from the event handler.
  + But you should be doing form validation on the server-side regardless

Preventing a form from being submitted

* Return false in the onsubmit event handling code to prevent a form from being submitted

Hiding and showing form sections

* Use CSS to show or hide a div section
  + *elementObject*.style.display = “block”; shows the div section
  + *elementObject*.style.display = “none”; hides the div section.
* Progressive enhancement: use JavaScript to first enable certain behaviors. If the user doesn’t have JavaScript, everything still works exactly the way it was before.

**UI Enhancement**

CSS and JavaScript

* JavaScript allows making the presentation and styles dynamic, just like we have been doing for making the actual content in the HTML dynamic
* Most direct way is to set inline styles on an element directly
  + Use the syntax: *elementObject*.style.*styleAttribute* = *stringValue*;
  + The style attributes in CSS may be slightly different in JavaScript since everything is camelCased in JavaScript. (e.g. background-color in CSS becomes backgroundColor in JavaScript). CSS uses hyphens, whereas JavaScript uses camelCase because hyphen is used for subtraction in JavaScript.
  + To set the class, you can’t use “class” as the attribute because “class” is reserved in JavaScript. Instead, use “className” and set this property to some string value. If you want to clear it, set it to the empty string.

Removing and applying CSS classes

* Example of applying CSS classes to an element: document.getElementById(“mainContent”).className = “example”;

Changing inline styles

* Example of applying CSS inline styles to an element: document.getElementById(“join”).style.position = “absolute”;

**JavaScript Best Practices**

Understanding JavaScript style (what you *should* do)

* Naming convention for variables and functions: camelCase.
* Naming convention for objects: capital letter for each word.
* Brace style (same line as the if/for/etc.): Just like Java

if (someCondition) {

//…

}

* Don’t need braces if you only have a one-line block after an if-statement, but this is bad practice. Always put braces.
* You should define functions before you call them
* Always use semicolons to end a statement.
* Always use var when declaring a variable.
* Research “JavaScript Style Guidelines” by Yahoo, Google, Mozilla, etc. for more details.

Minifying your code

* Minification compacts code. This reduces file size and decreases load time. It does NOT “compile”, nor does it intentionally obfuscate.
* Minification tools: JSMin, YUI Compressor, Google Closure Compiler
  + Google Closure Compiler might do optimization in addition to minification

Using JavaScript code checkers

* One best known JavaScript code quality checker: JSLint
* (“strict” mode does not allow sloppy JavaScript. For example, it require the use of var. Look at the section “Using Strict Mode” for details.)

**JavaScript Libraries**

Introduction to JavaScript libraries

* A JavaScript library is just a bunch of JavaScript somebody else wrote.
* Some general libraries: Google’s Closure Library, MooTools, Yahoo YUI Library, jQuery (the clear winner of all libraries)
* Some specialized libraries: Lightbox (popup images), script.aculous

Linking to multiple JavaScript files

* Our default way to link to other JavaScript files is to put one more script tags at the end of the file.
* But you want to keep the number to a minimum, and the order matters
  + Keep the number to a minimum because the JavaScript files are downloaded sequentially, not simultaneously. Thus, causes delays if there are a lot of files.
  + Order matters: if you have a file of JavaScript that inside it calls a function in a second JavaScript file, you want to load the second file first.

Introduction to jQuery

* To use it, go to jQuery.com and download the JavaScript. Then add the script tag in the HTML and link to the jQuery file.
* Easier to select items in the DOM. Syntax is: jQuery(*stringSelector*).*someAction*(*some params*);
  + Examples of string selector: “#myDiv” (id of myDiv), “.someClass” (class of someClass), “p” (all paragraph tags). These selectors are much like CSS selectors
  + Can combine selectors: “p.description” returns all paragraph tags with class description.
  + Further refine the selector: :first, :last, :contains(), :visible
  + Example of *someAction*(): addClass(“highlight”), removeClass(“highlight”), toggleClass(“highlight). A benefit of addClass as opposed to modifying the className attribute in normal JavaScript is that it preserves existing classes.
* Can replace “jQuery” in the statement with “$”. The $ is just an alias.
* Some effects jQuery provides: hide(m*illiseconds*), fadeout(*milliseconds*). Call these on a particular set of selected elements
* Using Events in jQuery:

*selectedElements*.*event*(function() {

/\* *event handling code* \*/

});

* + Get a reference to the element that triggered this event with the keyword “this”
  + Use $(document).ready instead of onload. Unlike onload, you can call it multiple times without having to worry about the issue of the last one overriding the rest

Using a content distribution network to deliver JavaScript files

* You can download jQuery to have a copy of the file, but a better way is to let someone else (e.g. Google or Microsoft) do it by getting that file directly from their servers
  + This is done via a Content Distribution Network (CDN)
  + Improved speed/redundancy, improved bandwidth, improved parallel downloads
  + The file is shared across sites because it is cached in the user’s web browser.
* Take the script tag that links to the jQuery file and replace the source with another distributor’s link to the file.
* Replace the “http://” or “https://” with just “//” to use whatever protocol the page is using
* See other Google API’s by searching “Google CDN” online.

**JavaScript and HTML5**

JavaScript in HTML5

* HTML 5 main features: better video support, audio support, Canvas element (draw on screen directly), offline storage, new form elements, drag-and-drop support, geolocation, local storage
* JavaScript additions: Currently we can only getElementsById and getElementsByTagName, but now there is getElementsByClassName. This feature is really only associated with HTML 5; for earlier versions, use jQuery.
* HTML 5 video comes with a bunch of JavaScript methods/events for the video element object. Examples: play(), pause(), currentTime, ended event, etc.
* HTML 5 storage: offline storage, Web SQL, IndexedDB
* Web Workers: load JavaScript and make it run in the background regardless of what else is going on (basically multithreading).
* When can you use these new features? We test for feature detection, not browser detection. (This is because HTML 5 is not deployed all at once.)
  + Example: document.getElementsByClassName will return if this method exists

Using Modernizr

* Modernizr app helps build HTML5-focused apps while staying aware of backwards compatibility: easy feature detection of HTML5 and CSS3 features
* Can copy the script file from online. Put the script tags for this file in the head of your file since it needs to run before the body tag loads
* After running this, you will have a Modernizr object that contains around 40 booleans, one for each feature that tells you if the HTML5 or CSS3 feature is supported
* Progressive enhancement: if the user has access to a feature, we will enhance the site with it; if not, we might do nothing or fall back to another method, but we don’t give something that is broken
* The production section of Modernizr allows you to generate a smaller version that only checks for certain features.

Using Strict Mode

* Allows you to make your code be held to higher standards than normal JavaScript, which lets you be quite sloppy.
* At the top of your file, put “use strict”; as its own line. Or do it at the top of the function if you one to do it function-by-function.
* Strict mode is good, but there are two things to be careful of
  + Because it’s common to combine multiple JavaScript into one when trying to optimize performance, make sure that you are not mixing the usage of strict mode with some files that don’t use strict mode.
  + To fix the above issue, you can use strict mode on a per-function basis, or you can wrap the strict-mode section of the file with a function:

(function() {

“use strict”;

//Rest of file…

}());

* + Second problem is that you shouldn’t turn on strict mode and then check your code in browsers that don’t support it. Instead, test in multiple browsers.

**Advanced JavaScript Features**

Knowing the JavaScript to avoid

* document.write(*string*), which writes a string directly to the current page
  + Problems: doesn’t work in XHTML, doesn’t understand the DOM properly, clunky/inflexible, can only be used when the page is initially loading (if you run it after the page is loaded, it will erase the contents of the document).
  + Instead work with innerHTML, or better yet, work with individual elements and modify their properties, etc.
* Browser-sniffing code: detects an actual browser or actual platform. Common browser-sniffing code contains navigator.userAgent or navigator.appName.
  + This is a very old practice. Instead, you should detect features.
* Eval function: take an expression and execute it as code. Example:

var a = “alert(‘”;

var b = “hello”;

var c = “’);”;

eval(a + b + c);

* + Eval will take the concatenated string and run it as if it were code.
  + There are almost better ways of doing it, is inefficient, and poses a security risk with regards to code injection attacks.
* Pseudo-protocol: way of making a link or button directly call a function. For example, the href=“javascript:*someFunction()*” attribute would be pseudo-protocol.
  + Problems: bad to combine JavaScript with HTML. And if JavaScript doesn’t work, this is useless (this link doesn’t do anything).
  + Somewhat preferable way is to have href attribute point to an actual page and the onclick attribute point to: “*someFunction*(); return false;” This causes *someFunction*() to be executed if JavaScript is enabled; otherwise we would call the HTML link. This still isn’t that good since JavaScript should all be in its separate file and enhance the already-working HTML file.

Introduction to regular expressions

* Regular expressions are odd-looking sequences of characters that describe and match patterns and strings
* Two main steps: create the regular expression, and then apply it to something else to see if it matches
* Creating regular expression syntax:
  + Syntax: var *regEx* = /*regularExpression*/; -- or – var *regEx* = new RegExp(“*regularExpression*”);
  + Example: var myRE = /hello/; /\* or \*/ var myRE = new RegExp(“hello”); This tests for if “hello” is present within the string
* Applying the regular expression syntax:
  + *regExVariable*.test(*string*) returns whether the pattern was found or not.
  + Example: if myRE contains the regular expression and myString contains the string we want to test for, then you use myRe.test(myString).
  + search instead of test returns the position of the first match.
* Creating more complex patterns using special symbols. Examples:
  + ^ means at the start. So /^hello/ would require “hello” to be present at the start of the string. /hello$/ if you want end of string instead.
  + + means the previous character must appear once or more. So for /hel+o/, these strings would match: “helo”, “hello”, “hellllllllllllo”. \* means zero or more, so “heo” would match. ? means zero or one.
  + | means either or. So /hello|goodbye/ would match any string that contains “hello” or “goodbye”
  + . means any character. /he..o/ would match “hello”, “heaah”, “hefjo”
  + \ is used to denote certain character. \w means alphanumeric or \_. \b means word boundary (e.g. space or new line)
  + [] are for range of character, one of which must appear. For /[crnd]ope/, “cope”, “rope”, “nope”, and “dope” would match. Different letter would not match.
* String them together for more complicated patterns:
  + /^[0-9]{5}(?:-[0-9]{4})?$/ checks for a valid zipcode. {5} means 5 of the previous item. ? means optional.
  + Validating email addresses are hard.

Working with AJAX

* AJAX = Asynchronous JavaScript and XML
* What is AJAX: after a web page has opened in the user’s browser, JavaScript can communicate behind the scenes back to the server, even passing and retrieving data and updating parts of the page without the entire page needing to be reloaded.
* To do this, we need to create the request, and then deal with the response from the server.
  + This is done separately because we don’t know how long the response will take, and we don’t want the page to hang while we wait (asynchronous tasks)
* First do a feature check before creating the HTTP request:

var myRequest;

if (window.XMLHttpRequest) { // Firefox, Safari

myRequest = new XMLHttpRequest();

} else if (window.ActiveXObject) { // IE

myRequest = new ActiveXObject(“Microsoft.XMLHTTP”);

}

* Second, prepare to accept the response by creating an event handler:

myRequest.onreadystatechange = function() {

// Event handling code for when the request is sent back.

};

* + But this event is quite chatty: the request will cause the event to be called multiple times: once for each stage of the request. (You can see the state of the request by calling the object property readyState on the request object.)
  + Thus, check if readyState is equal to 4, which means it is done with the request.
* Third, configure and send the request:

// Can specify “GET” or “POST”. *stringURL* can be PHP file.

myRequest.open(“GET”, *stringURL*, true);

myRequest.send(null); // or pass in some other parameters

* Get the content of the response through the responseText property of the request object.

Working with objects and prototypes

* Prototypes allow us to formalize or objects.
* Step 1: create a constructor function to make our new objects. Syntax: function *ClassName*(*parameters*) { /\* constructor code \*/ }
  + Recall that “this” can get us the current object.
  + Note that this isn’t really a class. This is just a function (object).
* Step 2: create the object: var *objectName* = new *ClassName*(*parameters*);
* Use the prototype property of the *ClassName* function to attach functions to *ClassName*.
  + Example: Player.prototype.logInfo = function() { /\* some code \*/ };

**Putting It All Together**

Example: Countdown

* Math.floor(*number*) rounds *number* down.

Example: Resize

* It’s good to make a webpage that looks different depending on the size of the screen
* Create two CSS files: one for normal view and one for smaller screens
* Get a reference to the link element that contains the CSS file, and then swap the value of the href attribute (using the setAttribute method on the element object).
* The event that occurs when the window is resizes is window.onresize (note that this method may be called multiple times when resizing the window).
* Ways to find the width (don’t memorize this, copy this somewhere else or use a third party library): window.innerWidth, document.documentElement.clientWidth, document.body.clientWidth.
  + You need to check if each of these properties exist before calling them.
* Note that onresize would not be called if the window was originally small. Thus, call the method also on window.onload.

Example: Accordion

* Use jQuery and jQueryUI for a more visually pleasing presentation.
* jQuery’s Accordion allows you to expand certain content by clicking on tabs.
* To use Accordion, put a div around the entire expanding section, put div’s around each of the expanding paragraphs, and then put meaningless links in each of the headers. Then in the JavaScript, get a reference to the entire expanding section div, and call the accordion() method on this element object.
* In jQuery UI, you don’t have to download every UI feature: you can build a custom download.

**Conclusion**

Where to go from here

* Looking online can be okay, but be very careful. The advice for JavaScript is abysmal compared to other languages
* Some good sites are developer.mozilla.org, dev.opera.com, Internet Explorer Developer Center, 3rd party sites for 3rd party libraries, developer.yahoo.com/javascript/ (developer.yahoo.com/performance), stackoverflow.com (forum)