

JavaScript IV:

Live Free or JavaScript Hard

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 events are and how to create an event listener
* Perform simple animations of attributes and style using the window object

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
  + *Discussion:* (slide 2) Briefly review
    - Variables
    - Arrays
    - Conditionals
    - Loops
    - Functions

(x) Events

* What are they?
  + *Explanation*
    - (slide 3) When we want to make a page respond to user input, we need to learn about events – these are objects created when a user does something: clicks a mouse, enters data in a text box, submits a form, etc.
    - To respond to these events, we need to create something that watches our page for any changes – an event listener.
    - (slide 4) The syntax to add an event listener looks like this:  
      target.addEventListener(type, listener);
      * The target is what we’re sticking the event listener on to. So we would make a new var that looks for an object on the page – a button, say – and the name of that var would be the “target”.
      * The type is what type of event we’re listening for. There’s a huge list of these, but we’ll stick mostly with click and mouseover today.
      * The listener is the object that is notified when the event happens – usually a function which we only want to run when the event is triggered.
    - (slide 5) There are tons of different events we can look for – mouse, keyboard, forms, window – these are just a few of the options available to us.
      * Teacher tip: you can pull up the full list of events from MDN at <https://developer.mozilla.org/en-US/docs/Web/Events>
  + *Activity: make your first event listener*
    - Step 1 – Have class open jsbin.com
      * Clicking on the bar at the top of the window, close everything but the HTML, JS, and Output windows
    - Step 2 – (slide 5) first, we’ll create a button to click in our HTML. Add:  
       <button id="counter">0</button>
    - Step 3 – now, let’s make a var in our JS so we can find the button and refer to it easily.  
      var counterBtn = document.getElementById('counter');
    - Step 4 – Now, let’s write the function which will add one to the counter each time we click the button:  
      function onButtonClick() {

counterBtn.innerHTML = parseInt(counterBtn.innerHTML) + 1;

}

* + - * Teacher tip: the parseInt is necessary to force the addition to work properly. The innerHTML of an element will be a string data type – and we’ve seen before that the “+” with a string is concatenation, not addition. ParseInt takes that string and turns it into an integer.
    - Step 5 – finally, we need to add an event listener to the button to listen for clicks.   
      counterBtn.addEventListener('click', onButtonClick);
    - Step 6 – click on the button to make sure it works!
* Why event listeners and not onclick?
  + *Explanation*
    - (slide 7) In the previous class, so that we didn’t introduce you to too many things at once, we used onclick in our HTML to trigger a function in our JS. This is actually bad practice:
      * For one thing, it doesn’t separate your JS and HTML cleanly, making maintenance harder.
      * Onclick can be overwritten if you, say, use innerHTML on an element – which would break your site.
      * You can’t attach multiple functions to a single event with onclick.
    - So, in short: you should use event listeners instead.
  + *Activity: Mad Libs!*
    - (slide 9) The class will try to solve the problem on their own, then we’ll come together to work out the answer.
    - Step 1 – have the class go to <http://jsbin.com/xomuv/1>
    - Step 2 – Now, in the JS, do the following:
      * Make a var that accesses the button named libButton.
      * Make a function called makeMadLib which takes the .value from each text box and stores them in separate variables, then writes a story to the webpage in the story div, like “Nate really likes sassy squirrels.”
      * Add an event listener that will call the function when the user clicks on the button.
    - One possible solution:

var btn = document.getElementById(‘libButton’);

function makeMadLib() {

var n = document.getElementById('noun').value;

var a = document.getElementById('adjective').value;

var p = document.getElementById('person').value;

document.getElementById('story').innerHTML = p + ' really likes ' + a + ' ' + n + '.';

}

btn.addEventListener('click', makeMadLib);

* The window object
  + *Explanation*
    - (slide 10) The window is an object created when you run JS in a browser. It’s the assumed global object – the object that holds all the other objects, so when you call a method without specifying an object, you’re actually calling the window. Like alert().
    - (slide 11) We can use the window object to animate things, using the window object to keep track of time for us.
      * To understand this, we need to have some understanding of animation –
        + Whether in a Bugs Bunny cartoon or when a menu bar slides down on a website, animation relies on our brain filling in gaps between two states.
        + In a hand-drawn cartoon, the artist will draw several frames for someone walking, for instance. (demonstrate by pausing while walking in front of class) But she doesn’t draw every millisecond of my walk – maybe 5-10 frames would work. Our brain fills in the gaps between the frames.
        + If there aren’t enough frames, or if the frames go by too quickly, the animation will look jerky. The more frames there are, the smoother the animation.
        + So what kind of things would we animate on a webpage?

Dropdown menus

Sprites in a game

Fancy headings

Just please don’t go nuts with it.

* + - * Knowing this, there are two methods of the window object which we can use for timing animation – setTimeout and setInterval.
        + setTimeout calls a function after a specified delay in milliseconds.
        + setInterval calls a function every specified interval in milliseconds.
      * What does this look like in practice?
        + JavaScript engines are single-thread, meaning only one thing can be executed at a time. JS queues events: so if the engine is busy with something else, it starts a line of events to be executed.
        + SetTimeout awaits your delay, then executes your function. If something else is happening, it will be delayed until the next possible point of execution – meaning that the delay in setTimeout will always be at least the time you set, if not more.
        + SetInterval attempts to fire the function every X milliseconds you set in the interval. If it can’t – say the function you trigger is longer than the delay – it will queue up. So you may have interval events happening back to back. If there is already an interval event in the queue, most engines won’t add another one, so intervals can be skipped if there isn’t enough room to run them.
        + Intervals can give you smoother animation – but they’re also harder on browser processing and can slow things down. Timeout gives a guaranteed pause to the browser to do other things.
  + Activity: animate element attributes
    - Step 1 – (slide 12) add a kitten picture to the current page – <img src=”<http://placekitten.com/g/300/200>”>
    - Step 2 – we’re going to make a function which makes the img grow. Let’s first make our function:  
      function makeImageBigger(){}
    - Step 3 – now let’s find our img using one of the methods we just learned.  
      var img = document.getElementsByTagName(‘img’)[0];
      * Teachers Tip: ask class – why do we need the 0 in brackets? Answer – getElementsByTagName returns an array, and we need to pick the first item in that array, the kitty picture.
    - Step 4 – we’re now going to animate an attribute of that image. Let’s make the “width” attribute change at a set interval.  
      img.width += 10;
      * Teacher Tip: remind class that we learned how to do this in the lesson on objects – we can use dot notation to refer to the properties of any element. The DOM creates an object out of each element in our HTML, and the attributes of that element are its properties. So the width attribute can be changed by referring to it as a property.
      * The shorthand we’re using here is new – we’re adding 10 to the existing width. Saying x += 10 is the same as saying x = x+10, so instead of writing img.width = img.width+10, we can write img.width += 10.
    - Step 5 – now we set up the timer to animate the property.  
      window.setInterval(makeImgBigger, 1000);
      * Inside the parentheses, we first call out the function we want to execute, and then specify how often we want to call that function, in milliseconds (so 1000ms = 1s). Try changing the ms value and see what happens.
  + Activity: animating styles
    - Step 1 – (slide 13) Alongside attributes, we can also animate CSS styles. This is a little trickier, so hang on.
      * If you remember your CSS, we specify units for things like positions (px, em, %, etc). For us to be able to change these values, we have to strip the units off and get a clean number, add to or subtract from that value, and then add the unit back on.
      * We can refer to CSS styles using the style property, followed by the CSS property. So let’s add some positioning to our img.
      * Have class move the img var out of the function, then add the following two lines:  
        img.style.position = ‘absolute’;  
        img.style.top = ‘0px’;
    - Step 2 – now let’s write a function to make the kitty fall from the top of the page. We’ll name it kittyFall:  
      function kittyFall() {}
    - Step 3 – first, we’re going to take the top value and strip the units off, storing this in a var. We’ll do this by using parseInt to convert it to an integer – this also allows us to do math with it.  
      var oldTop = parseInt(img.style.top);
    - Step 4 – now let’s change that value by adding ten to it:  
      var newTop = oldTop + 10;
    - Step 5 – we then have to add the units back on so the CSS works properly:  
      img.style.top = newTop + ‘px’;
    - Step 6 – now we set up a timer to animate this:  
      window.setInterval(kittyFall, 100);
  + Explanation: stopping an animation
    - (slide 14) Now, you’re going to want to stop an animation at some point. If we let this run forever, it will just keep sliding down the page forever.
    - To do this, we first need to store our interval or timeout in a var.
      * Why? Look at our current code – we just call an interval, which starts running. If I want to change anything about that interval, there’s no way for JS to access it – it doesn’t have a name!
    - Then we use the clearInterval or clearTimeout method on that var.
  + Activity: stop the falling kitty!
    - Let’s say we want to stop the animation if the kitty gets 400px down the page.
    - Step 1 – let’s name our interval so we can refer to it later. Give our existing interval the name fallTimer:  
      var fallTimer = setInterval(kittyFall, 100);
      * Note: if we run right now, the interval starts just like it did before. But now we have a name we can use to stop the animation!
    - Step 2 - If we want to do something only when a certain condition is true, what do we use? A conditional – an if! Inside our kittyFall function, let’s add a conditional:  
      if (newTop > 400){}
    - Step 3 - If that’s true – if the img has moved 400 px down the page - we want to stop the interval. So let’s put that inside the brackets:  
      window.clearInterval(fallTimer);
    - Step 4 – run it and make sure it works!
  + Activity: final exercise, kitty moonwalk!
    - Step 1 - (slide 16) Add a new img to your HTML:   
      <img src="http://gdriv.es/dpljs/cat\_walking.gif">
    - Step 2 – give students assignment: make cat walk from right side to left side of screen. Point out bonus exercise.
    - Step 3 – have students attempt to solve first part themselves. This will look identical to the previous exercise, but changing the left property in stead of top, so they shouldn’t have too much of a hard time.
    - Step 4 – encourage students to try bonus by themselves. To make this work, they’ll probably have to create a “switch” – a variable which keeps track of whether the cat should be moving forward or backward. It can look like this:  
      var imgAnim = document.getElementsByTagName('img')[0];

imgAnim.style.position = 'absolute';

imgAnim.style.left = '0px';

var forward = true;

function catWalk(){

var oldLeft = parseInt(imgAnim.style.left);

var newLeft = oldLeft + 10;

var newRight = oldLeft - 10;

if (oldLeft > 400){

forward = false;

} else if (oldLeft < 0) {

forward = true;

}

if (forward) {

imgAnim.style.left = newLeft + 'px';

} else {

imgAnim.style.left = newRight + 'px';

}

}

var catWalkTimer = window.setInterval(catWalk, 50);

* **Wrapping Up**
  + There’s tons of resources available to help you keep learning!
  + [Codecademy](http://www.codecademy.com/) has self-paced lessons in JS for free
  + [Code Combat](http://codecombat.com/) teaches you JS through RPG combat
  + The [Mozilla Developers Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript) has a full JS reference plus links to further resources
  + [Eloquent JavaScript](http://eloquentjavascript.net/) is available for free as an HTML book and is an excellent introduction to JS and programming in general
  + And there’s tons of books! Check out the 005s in the stacks….

**(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]