Week 9, continued

This is CS50. Harvard University. Fall 2014.

Cheng Gong

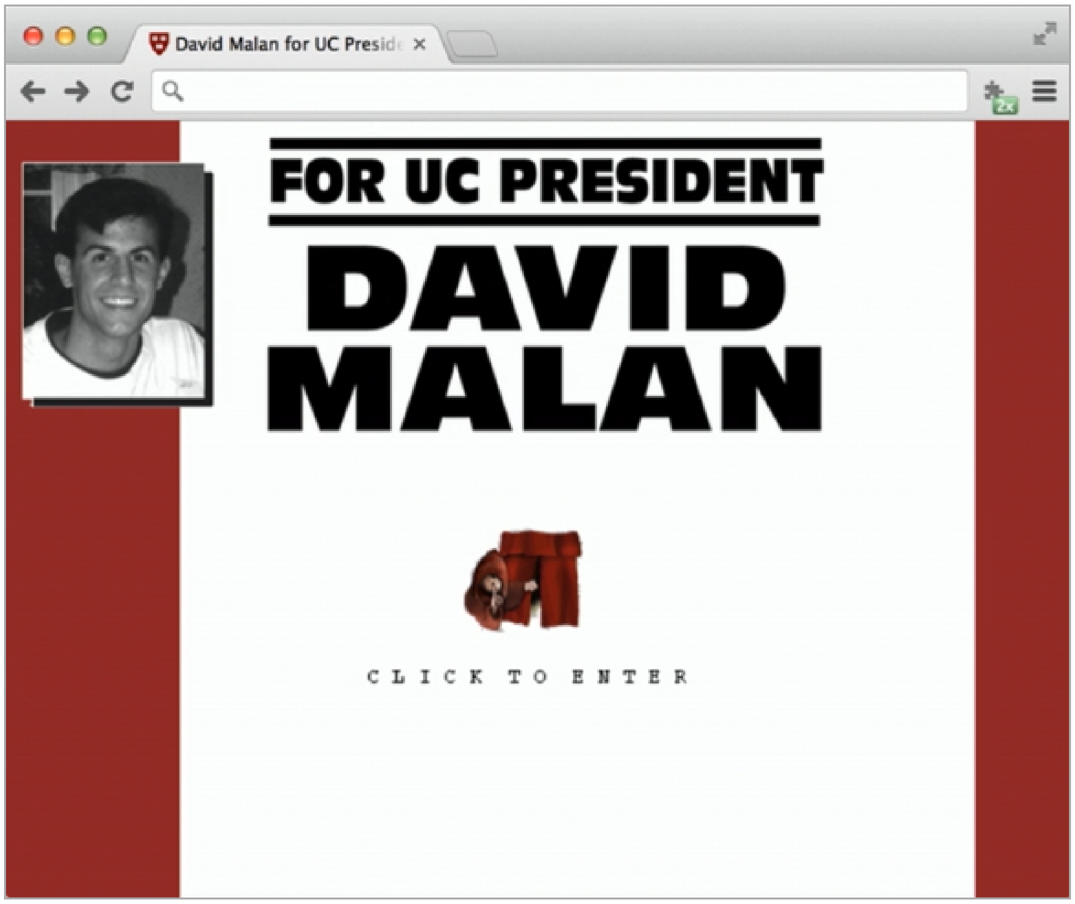
Table of Contents

Demos and News

- David starts lecture with the fancy new Myo armband, which wraps around your forearm, senses your muscle movements, and then sends those movements to a computer which recognizes them and does things as a result.
- A volunteer, Maria, comes on stage to try the armband, but it has trouble recognizing her movements. The armband we have is actually the developer kit, and on the bleeding edge of advanced technology, so bugs like this are to be expected! (But check out this demo video[1] or the Myo's API[2]!)
  - And late Happy Birthday to Maria, whose birthday was yesterday!
- Steve Ballmer '77, who joined Microsoft when it had only 30 employees and retired recently (with the company employing over 100,000 people!), will join us next Wednesday, 11/12, as a guest lecturer.
  - Seats may be limited, so sign up at http://cs50.harvard.edu/register
- A recent article entitled Five Tickets To Compete for UC Presidency, Vice Presidency[3] reminded David of his experience running and losing, which then motivated him to work on his public speaking and teaching skills.

[1] http://www.youtube.com/watch?v=aXoDK0EHdzM
[2] https://developer.thalmic.com/docs/api_reference/platform/index.html
[3] http://www.thecrimson.com/article/2014/11/5/five-tickets-uc-elections/

- Back in the late 90s when David was here campaigning, he actually made a website that looked like this:



  # Check out that monk with the curtain that you had to click to see the actual website … we give you a 0/5 for design, David of the late 90s.

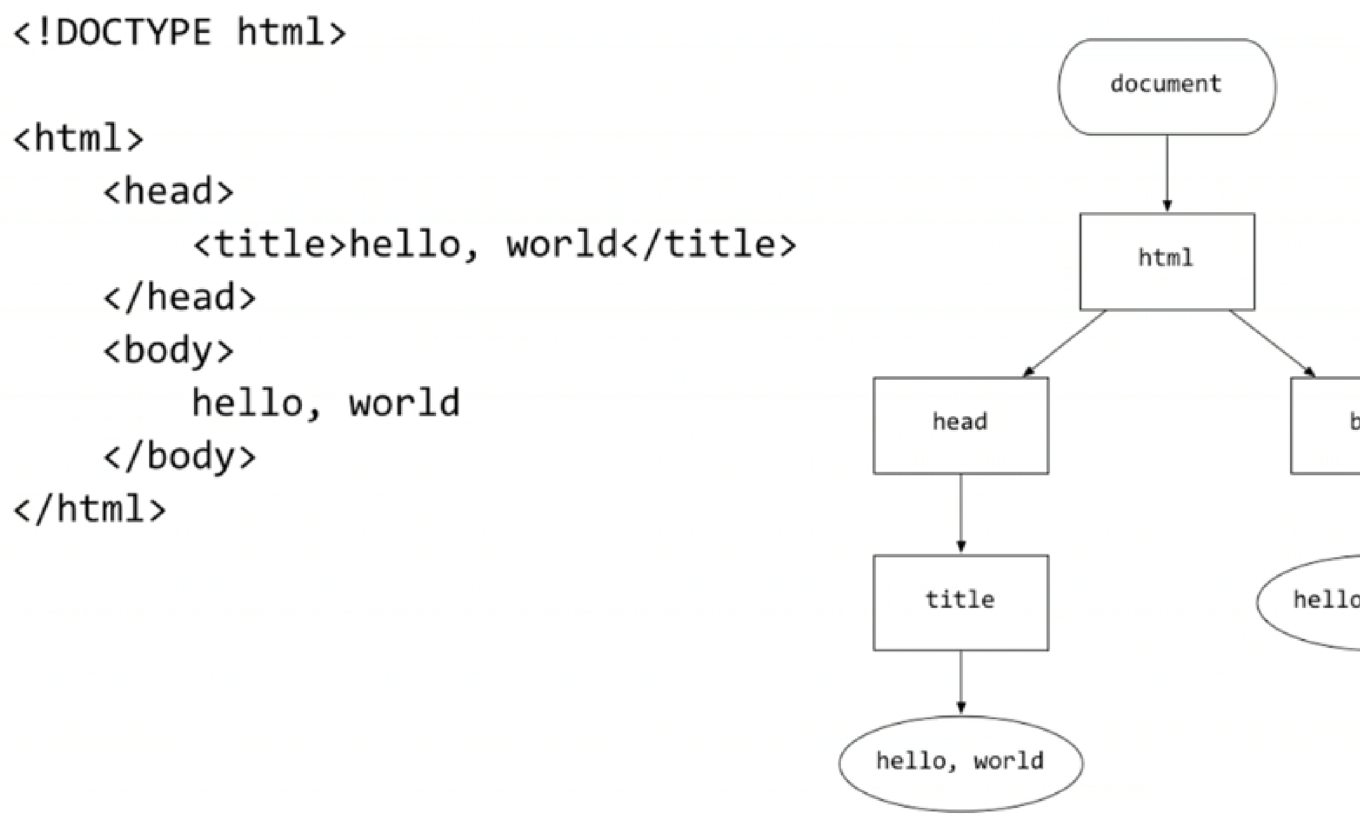
Recap

- HTML is a markup language that lets us structure a webpage (using headers, footers, headings, etc.).
- CSS lets us design or stylize elements, with boldfacing or colors or positioning. For example, a default table in HTML is pretty ugly and hard to read, so we can use CSS to make it prettier.
- PHP lets us generate dynamic content, and since it's a programming language, we can use it to communicate with a database or other servers, and do basically anything we can tell a computer to do.

- SQL is yet another language, used to talk to databases, with `INSERT`s, `DELETE`s, `UPDATE`s, and other functionality yet to be explored.

JavaScript Validation

- On Monday we introduced the last language we'll be formally learning about, JavaScript. Unlike PHP, which is interpreted on the server, JavaScript runs on the client-side, or in the browser. These days most websites include a bunch of `.js` files, that your browser then runs in a sandbox — i.e., it generally restricts JavaScript from, say, deleting your files or sending emails.
  - As an aside, JavaScript can also be run on a server, but we won't talk about it in that context.
- Recall the following HTML page and its corresponding tree structure that we talked about a few weeks ago:

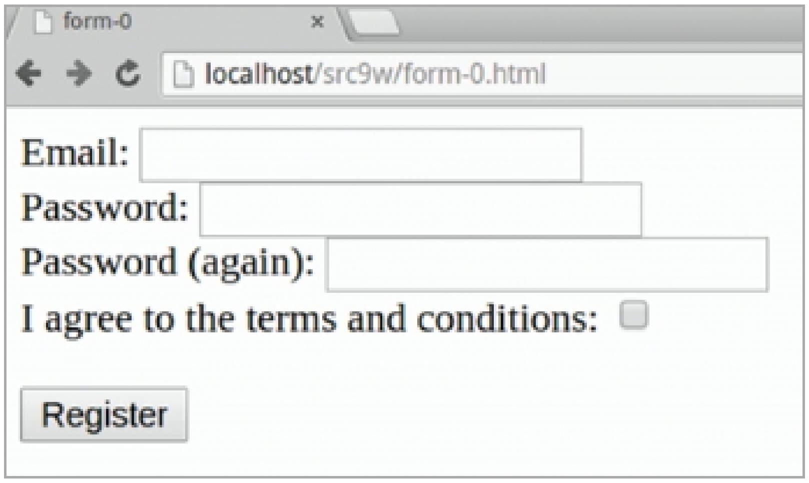


```
<!DOCTYPE html>

<html>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        hello, world
    </body>
</html>
```

# We think of the Document Object Model (DOM) like this because our JavaScript interacts with the webpage in your browser based on the relationships shown by this tree. For example, websites that have a chat feature might implement individual messages as `li` elements, or `div`s, and as you get new messages, they are being added by some JavaScript function as another node to this tree.

- Let's look at an example, `form-0.html`[4]:



Email:
Password:
Password (again):
I agree to the terms and conditions:

# It looks super simple, with no CSS, and the code is fairly straightforward as well:

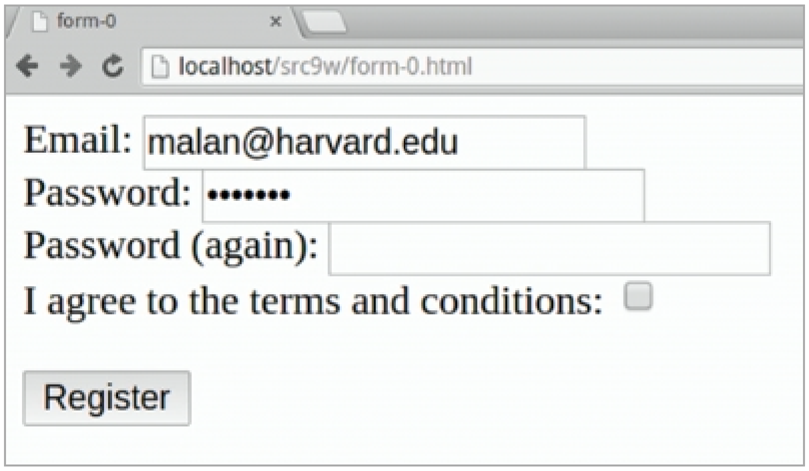
[4] http://cdn.cs50.net/2014/fall/lectures/9/w/src9w/form-0.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>form-0</title>
    </head>
    <body>
        <form action="register.php" method="get">
            Email: <input name="email" type="text"/>
            <br/>
            Password: <input name="password" type="password"/>
            <br/>
            Password
 (again): <input name="confirmation" type="password"/>
            <br/>
            I agree to the terms and
 conditions: <input name="agreement" type="checkbox"/>
            <br/><br/>
            <input type="submit" value="Register"/>
        </form>
    </body>
</html>
```

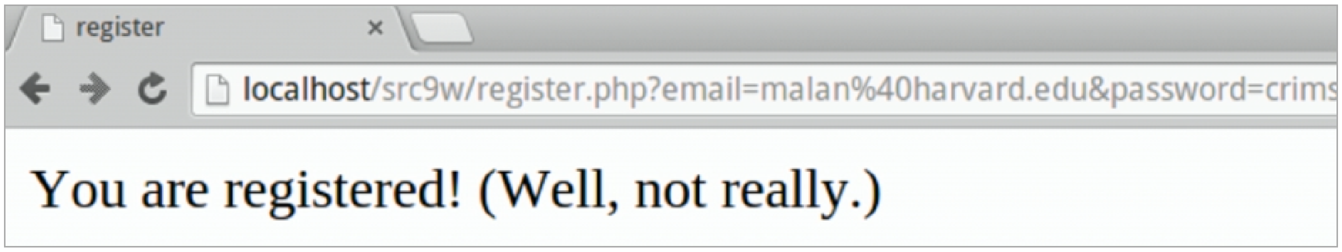
# We have a `form` tag that will go to some file called `register.php`, and then we have a `text` field, two `password` fields, and a `checkbox` field.

- Now if we fill out the form like this:



Email: malan@harvard.edu
Password: •••••••
Password (again):
I agree to the terms and conditions:

- We get:

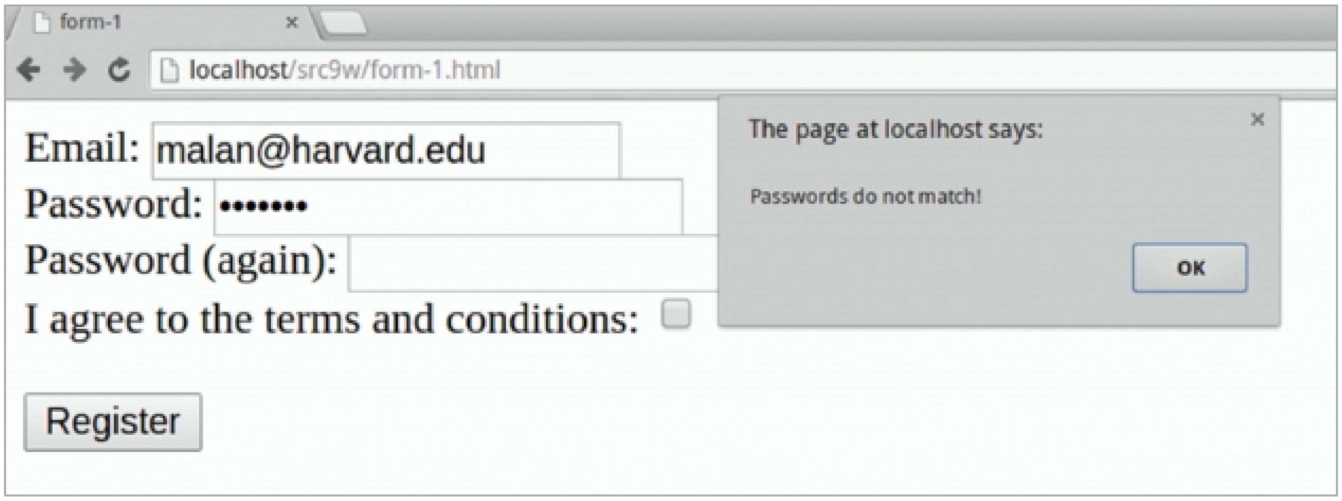


localhost/src9w/register.php?email=malan%40harvard.edu&password=crims

You are registered! (Well, not really.)

- We notice that the URL changed, but we probably should have checked for errors like the blank password confirmation and the unchecked box. We already learned how to do this on the server side, but lots of websites now do it on the client side, in the browser, so you can get instant feedback without refreshing the page.

- Let's open `form-1.html`[5], which looks the same, but now when we submit an incomplete form, we get:



localhost/src9w/form-1.html

Email: malan@harvard.edu
Password: •••••••
Password (again):
I agree to the terms and conditions:

The page at localhost says:

Passwords do not match!

  # This error comes from the `alert` function in JavaScript, which we use for now but in general should be avoided since it's fairly annoying.

  # Notice that the URL did not change, meaning we didn't need to ask the server and waste its time, and the user will also get instant feedback.

- The source code for `form-1.html` looks like this:

[5] http://cdn.cs50.net/2014/fall/lectures/9/w/src9w/form-1.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>form-1</title>
    </head>
    <body>
        <form action="register.php" id="registration" method="get">
            Email: <input name="email" type="text"/>
            <br/>
            Password: <input name="password" type="password"/>
            <br/>
            Password (again): <input name="confirmation" type="password"/>
            <br/>
            I agree to the terms and
 conditions: <input name="agreement" type="checkbox"/>
            <br/><br/>
            <input type="submit" value="Register"/>
        </form>
        <script>

            var form = document.getElementById('registration'); ❶

            // onsubmit
            form.onsubmit = function() { ❷

                // validate email
                if (form.email.value == '') ❸
                {
                    alert('You must provide your email address!');
                    return false;
                }

                // validate password
                else if (form.password.value == '')
                {
                    alert('You must provide a password!');
                    return false;
                }

                // validate confirmation
                else if (form.password.value != form.confirmation.value) ❹
                {
                    alert('Passwords do not match!');
                    return false;
                }
```

# Notice that the HTML looks structurally the same, with the same form, but we now have a `script` in the second half of the file. (And note that this isn't the only, or most elegant, way to implement this, but it's the most straightforward.)

# In line 21 we get the element with ID `registration`, and store it in a variable called `form` so we can use it later easily. We do this with `document.getElementById`, which we can think of as a function in JavaScript that finds the element in the DOM (think of the tree) and returns a pointer to it, so we can modify it.

# In line 24, the syntax looks a bit different than in C, but we can break it down. On the left side of the `=`, we have `form.onsubmit`, which means we're storing something into a field called `onsubmit` of the variable called `form`, just like how we might store something into a field into a `struct` in C. But browsers expect DOM elements to have a function in the `onsubmit` field, rather than a value or string. So on the right side of the `=` we see `function () {`, which means that we're about to define some anonymous function, and that's what the `onsubmit` field in `form` will refer to. We can leave this function anonymous because browsers are programmed to execute whatever function is in `onsubmit` for a form when the form is submitted.

# Also note that `function () {` has the opening curly brace on the same line, whereas in C the curly brace is on the next line. Just a stylistic thing that JavaScript folks tend to prefer!

# And that function will be everything inside the curly braces.

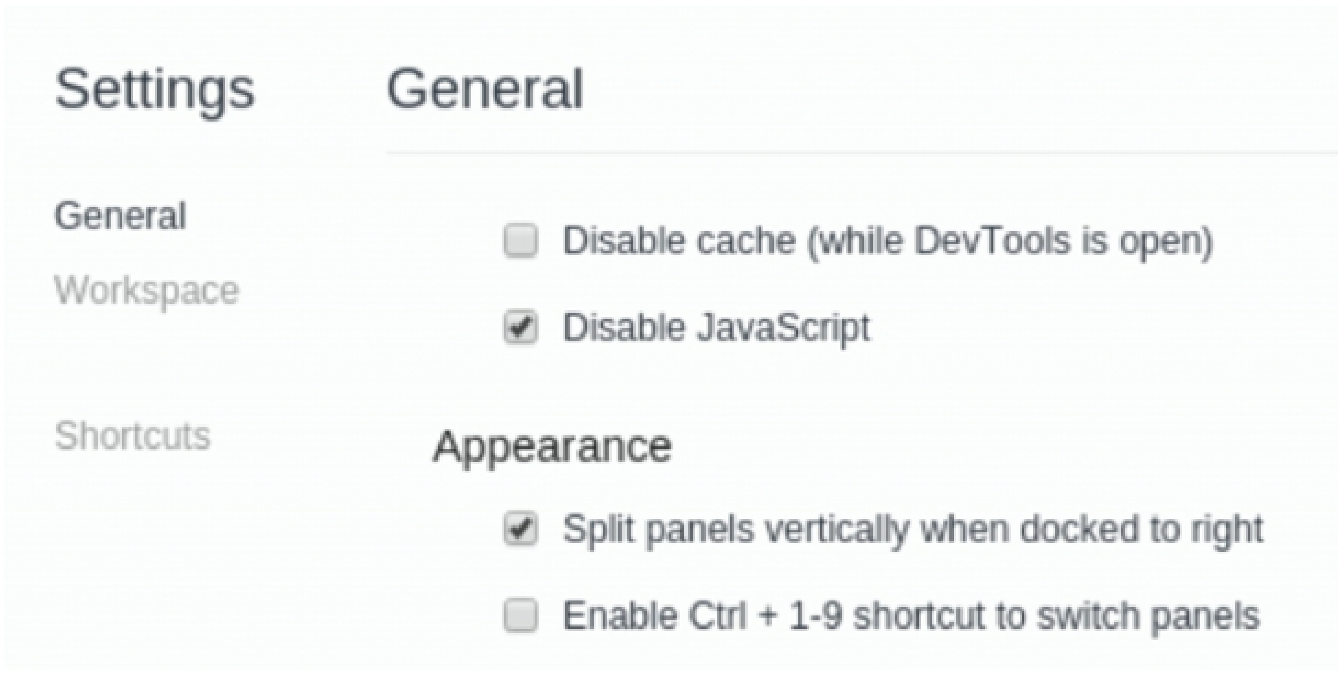
# So on line 27, for example, we check whether the `value` of the `email` field in the `form` is empty, or `''`, and if it is, call the `alert` function to tell the user. Then `return false;` will stop the form from being submitted, so the user can fix the error.

# We do the same for both password fields, but then in line 41 we check whether the two strings are the same, with `!=`. (JavaScript allows us to compare strings like this directly, unlike C!)

# Then in line 48, we check whether the `checked` field was checked, and if it's not (with the `!`), we tell the user.

# Finally, we `return true` in line 55 to allow the form submission to continue.

- We could easily do this in PHP after the form is submitted, but remember the whole point of this is to make responses faster and the user experience better overall. Imagine if you had to refresh Gmail every time you had a new email, or (God forbid), refresh the entire Facebook page every time you got a chat message. (Back in the Dark Days of the Internet when David ran for UC, that's what the user experience was actually like … shudder.)
- But just because we are doing this on the client side now, doesn't mean we should abandon server-side checking. We want to check for errors in both places, since some users or (more likely) web crawling bots might have JavaScript disabled, and not execute the checking code.
- Indeed, we can easily go into Chrome's **Settings**, scroll a bit down, and check a single box to **Disable JavaScript**:

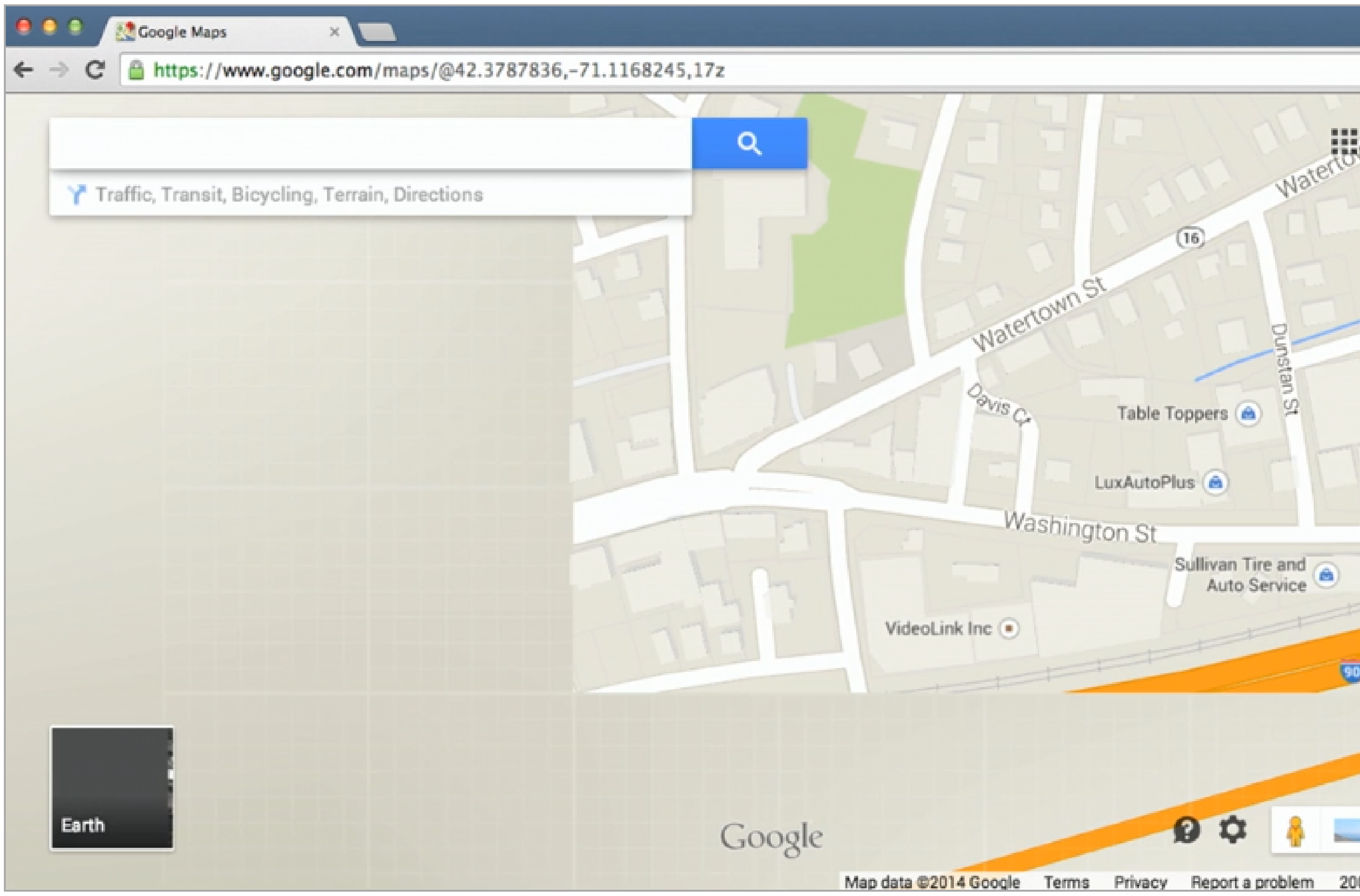


☐ Disable cache (while DevTools is open)
✓ Disable JavaScript
✓ Split panels vertically when docked to right
☐ Enable Ctrl + 1-9 shortcut to switch panels

- And we could also use `curl` or `telnet` to send messages to servers that would certainly not be error-checked.
- So using JavaScript is more for the users' benefit than as an improvement in security or validation.
- In `form-2.html`, we implement the same functionality in a slightly different way, with `$(document).ready(function() {`, but we'll explain this method more fully in a walkthrough in Problem Set 8! And this method uses a popular JavaScript library called jQuery.

Ajax

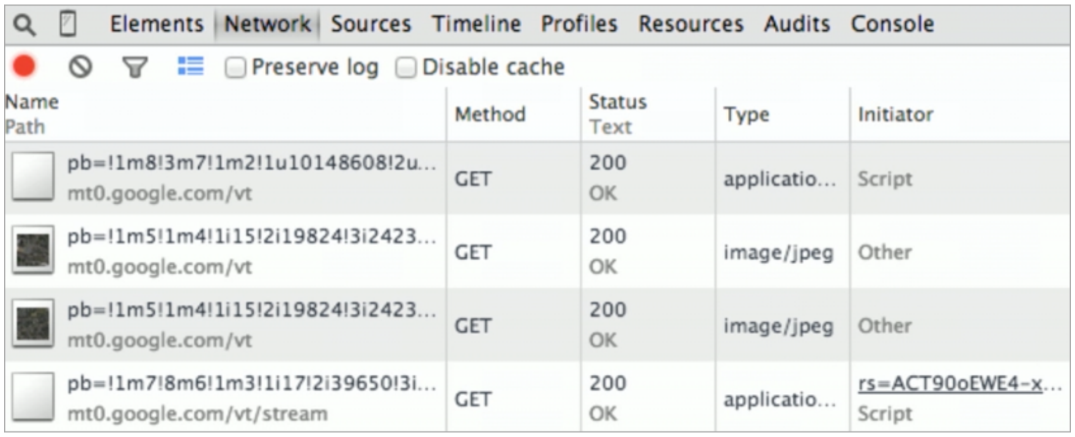
- Let's look at some cooler applications, using a technique called **asynchronous JavaScript and XML** (**Ajax**), even though these days we tend to use JSON as opposed to XML. This is how something like Google Maps works.
  - # Back in the day, MapQuest and other sites would have arrows around a small square map, as links that you'd have to click on to pan the map.
- Nowadays, we can just click and drag our way around. But if we do this fast enough, we see some small glitches:



  - # These blank squares are tiles that are missing from the map, until the map data is loaded.
  - # As an aside, Comcast has an online form where you can put in an address and it tries to autocomplete by filling in the rest of the address. But it starts with the first character, so as soon as you type in `33`, it will suggest every single

possible address that starts with `33`. So there are good ways and bad ways to use JavaScript.

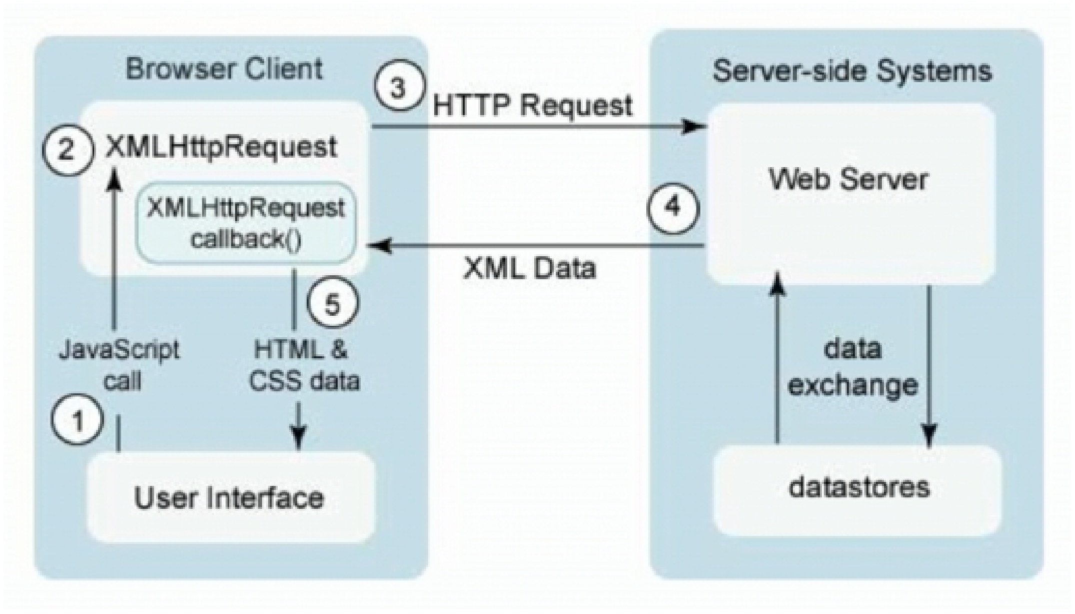
- In any case, we can open the **Network** tab in Chrome, and as we pan around we see that lots of `GET` requests are being made, many of which are `image/jpeg` files:



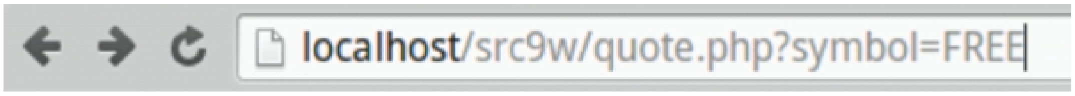
Name Path	Method	Status Text	Type	Initiator
pb=!1m8!3m7!1m2!1u10148608!2u... mt0.google.com/vt	GET	200 OK	applicatio...	Script
pb=!1m5!1m4!1i15!2i19824!3i2423... mt0.google.com/vt	GET	200 OK	image/jpeg	Other
pb=!1m5!1m4!1i15!2i19824!3i2423... mt0.google.com/vt	GET	200 OK	image/jpeg	Other
pb=!1m7!8m6!1m3!1i17!2i39650!3i... mt0.google.com/vt/stream	GET	200 OK	applicatio...	rs=ACT90oEWE4-x... Script

# So it looks like each tile is getting downloaded and added to the DOM, so the user can see the updated tile.

- So this is done with Ajax, and this complicated-looking chart has lots of details about the technique:

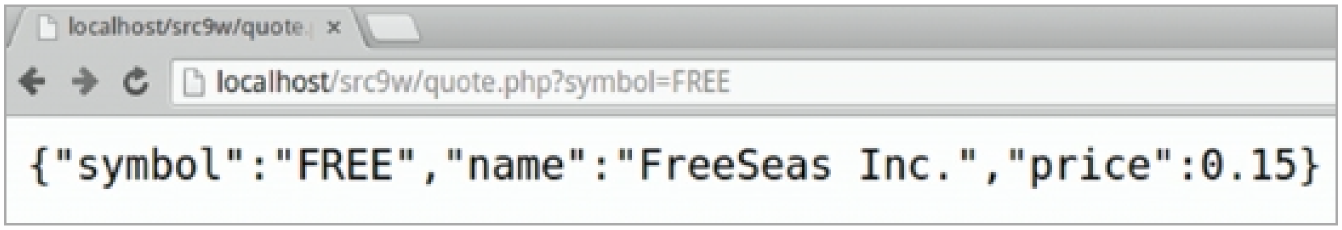


- But we will first try to explain it by example. Let's open `quote.php`[6], but pass in a query:



localhost/src9w/quote.php?symbol=FREE

- And this is what we get back:



localhost/src9w/quote.php?symbol=FREE

```
{"symbol":"FREE","name":"FreeSeas Inc.","price":0.15}
```

  # The braces and quotes remind us of JavaScript code, and indeed this is a JavaScript object, outputted in JSON.

- **JavaScript Object Notation** (**JSON**) is just a way for us to represent data in JavaScript with key-value pairs. For example, we can define a `student` as follows:

```
var student =
{
    id: 1,
    house: "Winthrop House",
    name: "Zamyla Chan"
}
```

  # This is like a `struct`, where we can assign certain values to fields in the same data structure.

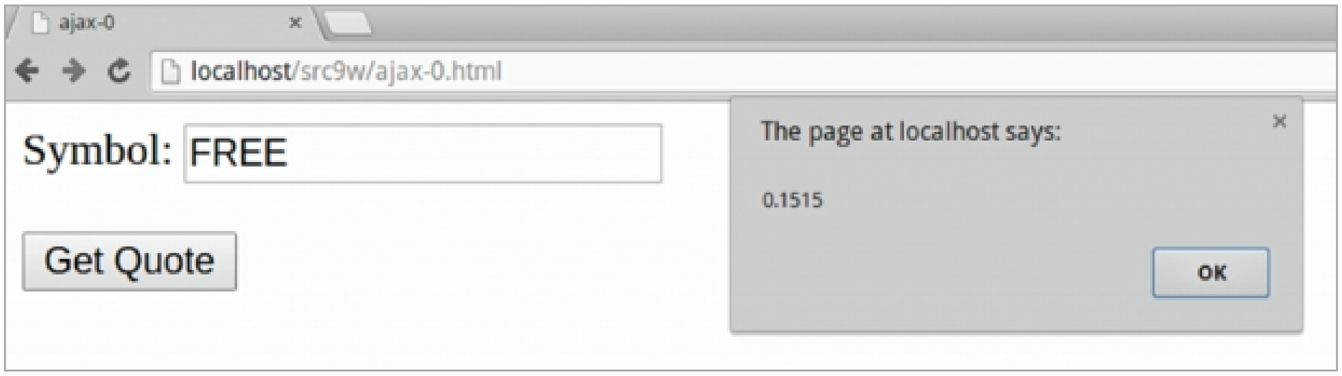
- But we can also have an array of these objects:

6 http://cdn.cs50.net/2014/fall/lectures/9/w/src9w/quote.php

```
var staff = [
    {
        "id": 1,
        "house": "Eliot House",
        "name": "Joseph Ong"
    },
    {
        "id": 2,
        "house": "Winthrop House",
        "name": "R.J. Aquino"
    },
    {
        "id": 3,
        "house": "Mather House",
        "name": "Lucas Freitas"
    }
];
```

# Notice the syntax with the brackets and commas, separating three objects that are otherwise identical to the one containing `Zamyla`.

- So having data in this format allows us to easily use it in our programs. Let's open `ajax-0.html`[7], and this is what happens when we type in `FREE`:



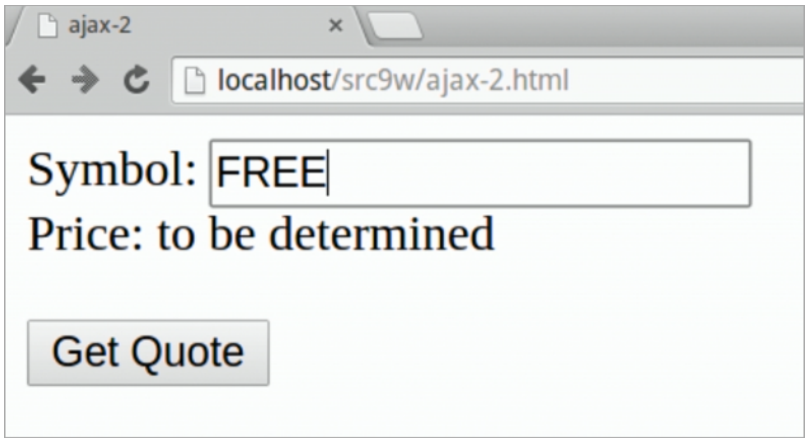
The page at localhost says:

0.1515

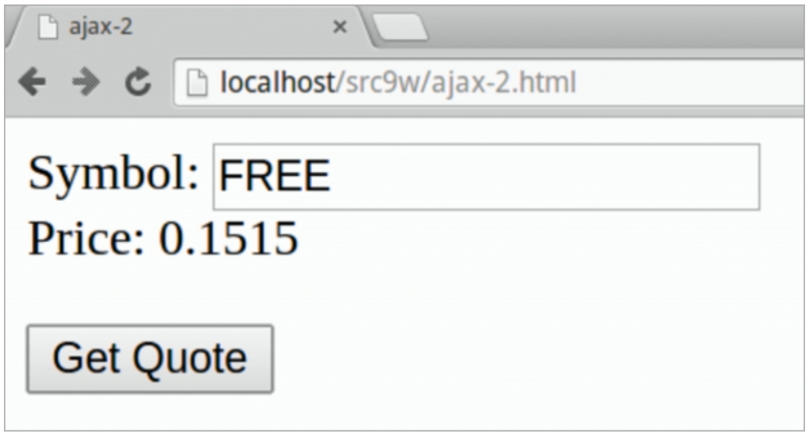
# Noice that the URL hasn't changed, and that we got an alert with the current stock price for `FREE`.

- Let's open `ajax-2.html`[8], and do the same:

7 http://cdn.cs50.net/2014/fall/lectures/9/w/src9w/ajax-0.html
8 http://cdn.cs50.net/2014/fall/lectures/9/w/src9w/ajax-2.html



- Before we click **Get Quote**, the "Price:" field is just text that says "to be determined." And when we do click **Get Quote**, it looks like the data was dynamically updated on our page, without the URL changing:



- Let’s open the source code for `ajax-2.html`:

```
<!DOCTYPE html>

<html>
    <head>
        <script src="http://code.jquery.com/jquery-latest.min.js"></
script> ❶
        <script>

            /**
             * Gets a quote via JSON.
             */
            function quote() ❷
            {
                var url = 'quote.php?symbol=' + $('#symbol').val(); ❸
                $.getJSON(url, function(data) { ❹
                    $('#price').html(data.price); ❺
                });
            }

        </script>
        <title>ajax-2</title>
    </head>
    <body>
        <form onsubmit="quote(); return false;"> ❻
            Symbol: <input autocomplete="off" id="symbol" type="text"/> ❼
            <br/>
            Price: <span id="price">to be determined</span> // 26
            <br/><br/>
            <input type="submit" value="Get Quote"/>
        </form>
    </body>
</html>
```

# Let's start by looking at our HTML `form`. On line 23, we start our `form` as usual, and on line 24 we turn `autocomplete` `off` so the browser doesn't show that dropdown with everything we've ever typed in. That text field is given the `id` of `symbol`.

# In line 26 we have something new, a `span` tag, which we can think of like a `p` or `div` tag, except it's an inline element, meaning it'll stay in the same line on the

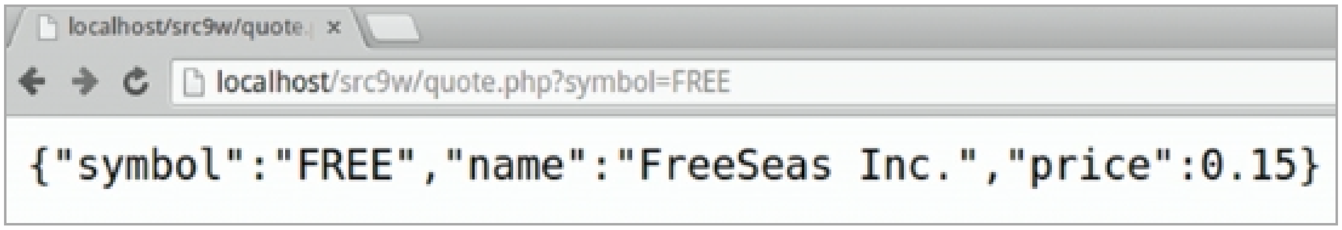
page, and not move down. That element is given an `id` of `price`, so we can identify it later.

# Now let's look up towards the top. On line 5, we write a `script` tag that references the latest jQuery library so we can use it, just like `#include` in C or `require` in PHP.

# Then in line 11 we start our own function, apparently named `quote`. In line 13, we create our own string, named `url`, by putting together the text `quote.php?symbol=` (the single quotes create a string) and whatever is in the form field with ID `symbol` (`$('#symbol').val()` is just jQuery syntax for getting that value. `#symbol` refers to the DOM element with ID `symbol`, the `$('')` around it is telling jQuery to select it for us to use as an object, and then `.val()` is just a method inside this object that we can call to get the value.)

# In the beginning of line 14, we have a dollar sign `$` by itself, which is equivalent to saying `jQuery`, a global variable that the jQuery library gives us, to access certain methods that aren't element-specific. (It does *not* mean the start of a variable name, like it does in PHP.) So `getJSON` is a function that does what it sounds like, getting JSON from a server. It gets it from the URL, which we pass in as the string that we called `url`, and a `function` is called as soon as the browser gets the data back. Remember that the first A in Ajax stands for asynchronous, and that just means the browser will send the request to the server, and not have to wait for a response before doing anything else. `function(data)` is what we call a **callback function**, which means that the function will be automatically called when the server responds. `data`, whatever the server responds with, will be passed in to that function.

# Finally, once the server responds, we'll set the `html` of `#price`, the `span` element on the page that the user can see, to the value of the `price` field of `data`, the JSON object that's returned. Remember that `quote.php` returns JSON that looks like this,



```
{"symbol":"FREE","name":"FreeSeas Inc.","price":0.15}
```

with a `price` field of `0.15` (The stock's price must have risen from 0.15 to 0.1515 in the few minutes since that example!). Indeed, we could be even more clear and format the data like this:

```
{
    "symbol": "FREE",
    "name": "FreeSeas Inc.",
    "price": 0.15
}
```

- So lots of moving parts here, but the key takeaway is that, with JavaScript and Ajax, we can make HTTP requests and change elements on the page, all without having to reload the entire webpage.
  - Indeed, these days you can write entire software applications with JavaScript, but we won't go into that much.

Another Demo

- Instead, let's do something fun and demo another device, the Leap Motion. This device plugs into a USB port and uses infrared beams to detect your gestures when your hand is held over the device, and then converts that to input.
  - We demo a few games with David, and then a volunteer, Laura. (Check out some of Leap Motion's demo videos[9], or their API[10]!)
- That's all for today!

[9] https://developer.leapmotion.com/gallery
[10] https://developer.leapmotion.com/documentation/skeletal/csharp/devguide/Intro_Skeleton_API.html