Day 9

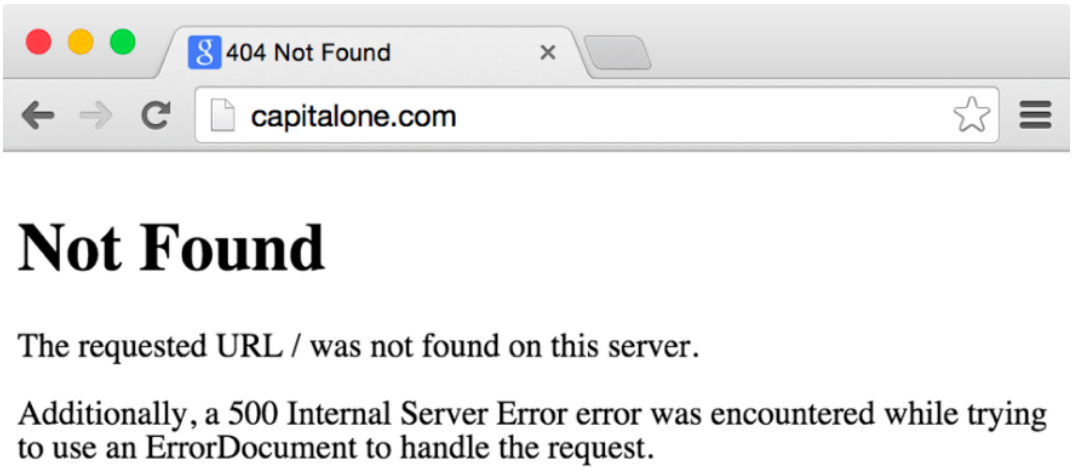
This is CS50 for MBAs. Harvard Busines School. Spring 2015.

Cheng Gong

Table of Contents

Capital One

- This morning, David tried to go on Capital One's website, and saw this:

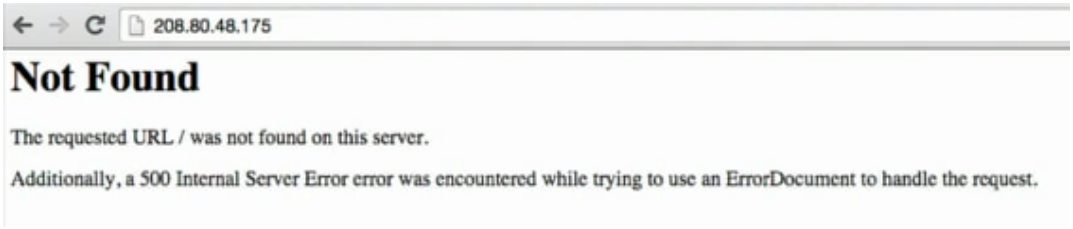


- On top, we notice that the title says `404 Not Found`, but on the page, we also have this `500 Internal Server Error`.

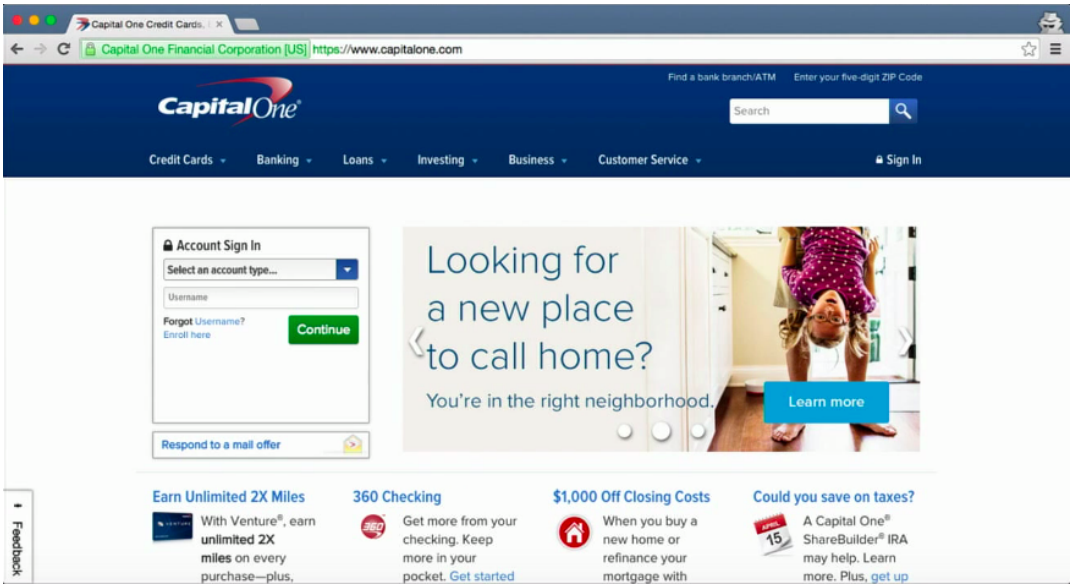
- To troubleshoot, we'd probably first check the DNS lookup of `capitalone.com`:

```
% nslookup capitalone.com
Server:         199.94.20.69
Address:        199.94.20.69#53

Non-authoritative answer:
Name:   capitalone.com
Address: 208.80.48.175
Name:   capitalone.com
Address: 208.80.50.175

%
```

- We can copy and paste the first IP and see what happens:



- Hm, that resulted in the same, so DNS is probably working okay.
- Maybe that one server is down, and in the case of scaled websites with many servers, we just got unlucky and reached a broken one.
- Large websites use servers that are called load balancers, which send users to many other servers that actually have content, in order to balance the load. If we somehow knew that, for instance, a cookie sent us to this particular server for content, maybe we can fix this by clearing our cookies. Alternatively, we can open an Incognito tab in Chrome, and see if that works, but alas, it doesn't.
- But maybe only part of the site is broken, and indeed trying `www.capitalone.com` worked:



- So this brings up the question of why websites have this `www` in their address, or if they need it at all. The short answer is that most companies choose to standardize and have it one way or the other, and simply redirect if people enter the other address. For example, CS50's website used to live at `cs50.net` and then both `cs50.net` and `cs50.harvard.edu`, but now `cs50.net` simply redirects to `cs50.harvard.edu`. A practical reason is that cookies can be specific to entire domains, like `hbs.edu`, or just `www.hbs.edu`, and so having more control over that might be necessary.

Questions

- "I understand that JavaScript is pretty ubiquitous when it comes to developing interactive websites, but I found a lot of information on why Python is better and very little on why JavaScript is so popular. Can someone explain why JavaScript is actually better for carrying out certain tasks?"

  # Browsers generally only support JavaScript code to detect events from the user, like keystrokes, so other languages wouldn't work for those cases. It's possible to compile Python into JavaScript that a browser understands, but that comes with tradeoffs like speed and compatibility. A website can actually run both Python and JavaScript, too, if the server has Python to generate HTML pages for the user, and the page itself has some JavaScript code for user interactions.

- "If you do validation both client side and on ruby or php, and use a library to generate the additional/duplicate code, then if you change the original coding do you have to run the library again?"

  # So we would use the library by calling some function in that library, and only the relevant code would be run. If we changed the underlying code, then the short answer would be yes, where the library to generate the new code would need to be used again. And remembering our discussion last time, validation on just the client side is insufficient because a user can turn off JavaScript or use the command-line to pretend to be a browser. But it's good because the user can see the results of validation much faster than having their input sent to the server and waiting for a response, since it happens within their browser.

- "I just logged into the survey via the Harvard PIN system to submit today's assignment, and I noticed that the web address is: `https://harvard.qualtrics.com/…/?email=&fullname=&huid=*&…`. The asterix at the end was actually just my full HUID. Given what we talked about before, perhaps it would be more secure if they move to POST? Obviously perhaps that long seemingly gibberish string in the middle would prevent me from logging in as someone else, but if not then by the looks of it, I could pretend to be anyone when submitting assignments!

  # One reason GET is being used is because it's easier to redirect users, and Qualtrics, the survey vendor Harvard uses, in particular only accepts GET requests to generate pages and receive inputs. As for the security aspect, the Harvard PIN system probably passes something along with the HUID, in a mechanism called single sign-on[1]. Facebook Login, for example, authenticates users without passing passwords around, but rather other cryptographic keys that validate users' identities.

- In case anyone is interested - LastPass offers 6 month free of LastPass premium for students, which gives you access to your lastpass library on mobile devices so you don't get stuck on different devices/away from your computer without your password (and in iOS8, auto fills within your safari browser): `https://lastpass.com/edupromo.php`

- Following up on today's question, when you ask us how much time it took to complete the assignment, do you just mean the questions we submit or also the "do as much as you want" part?

[1] https://en.wikipedia.org/wiki/Single_sign-on

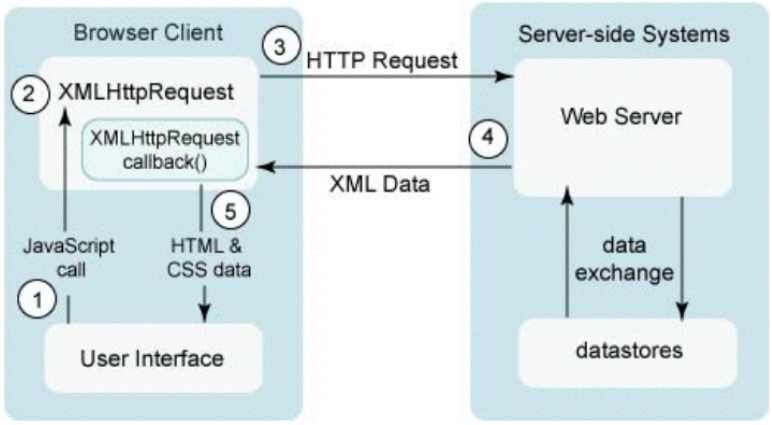
We're interested in the total experience, so to speak, of the assignment: from the time you read the first word to the time you press submit on the form, or the total time you spent on the assignment, all things considered.

Wat

- Now we'll watch a quick talk[2], referencing weird features of languages, meant for fellow programmers. For example, in JavaScript an empty array plus an empty array somehow results in an empty string (wat)!

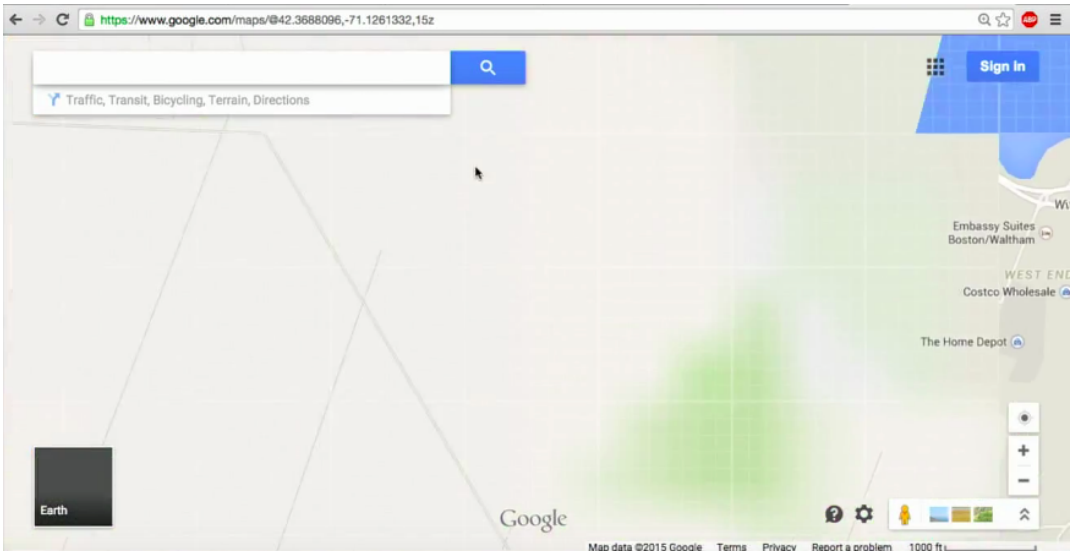
Ajax

- Let's take a look at this picture:

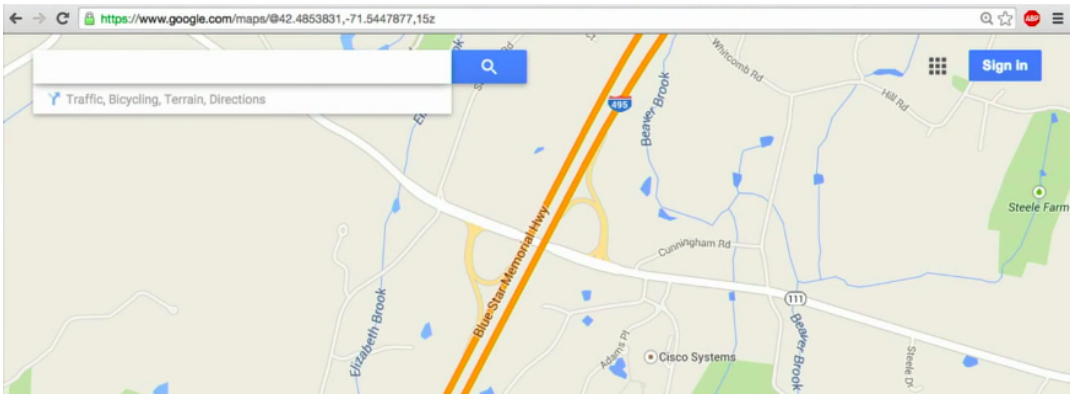


- It might be a litle bit hard to understand, so let's go through it. Back in the day (and for many sites today), when you loaded a webpage and everything was done, you'd have a page of content. And to get new content, you would click a link, changing to another page for that new content. But Ajax is a new technique that allows webpages, using JavaScript, to load new information from the server without changing the page. Examples include Facebook's news feed that updates, its chat box, Google's chat windows, Google Maps, and lots of others.
- On Google Maps, we can see this happening if we drag the map really fast:

[2] https://www.destroyallsoftware.com/talks/wat



- The parts that haven't fully loaded look like that, but after a second or two the details come in:



- The JavaScript code on that page is sending HTTP requests to Google's servers, in response to the map being dragged by the user, to get more information to draw that new part of the map.
- Looking back at the picture, we also realize that Ajax, which stands for "asynchronous JavaScript and XML", receive XML data (which is a markup lanuage similar to HTML) that the JavaScript code can then display on the page.
- Unlike HTML, which has set tags, XML allows for any tag:

```
<class>
    <students>
        <student id="123">
```

```
            <name>David</name>
            <email>malan@harvard.edu</email>
        </student>
    </students>
</class>
```

- This is kind of like a text-based database, where you have a hierarchy and type of data you can define and store information in. However, for every object we have to have an opening and closing tag, which adds up for size and time.
- JSON (JavaScript Object Notation) is an alternative where we have text that looks like this:

```
var student =
{
    id: 1,
    house: "Eliot House",
    name: "Willy Xiao"
};
```

- This JavaScript object named `student` has key-value pairs, three of which to be precise.
- The variable `staff` is an array of objects, since we see the square brackets `[]` wrapping multiple objects that have curly braces `{}`.

```
var staff = [
    {
        "id": 1,
        "house": "Eliot  House",
        "name": "Willy Xiao"
    },
    {
    "id": 2,
    "house": "Mather House",
    "name": "Gabriel Guimaraes"
    },
    ...
];
```

- So we can represent the same data in JSON as we could in XML, but maybe more efficiently.

- Let's look back to `blink.html`[3] from Day 8 that had a source like this:

[3] http://cdn.cs50.net/2015/mba/classes/8/src8/blink.html

```
<!--

blink.html

Flashes a greeting.

Computer Science 50
David J. Malan

-->

<!DOCTYPE html>

<html>
    <head>
        <script>

            // toggles visibility of greeting
            function blink()
            {
                var div = document.getElementById('greeting');
                if (div.style.visibility == "hidden")
                {
                    div.style.visibility = "visible";
                }
                else
                {
                    div.style.visibility = "hidden";
                }
            }

            // blink every 500ms
            window.setInterval(blink, 500);

        </script>
        <style>

            #greeting
            {
                font-size: 96pt;
                margin: 240px;
                text-align: center;
            }

        </style>
        <title>blink</title>
    </head>
```

- We have a `div` with an `id` of `greeting`, and the lines of code within the `script` tag took that `div` and made it appear and disappear as though it were blinking.
- `window.setInterval` is calling a function `blink` every `500` milliseconds, but notice that we're passing in `blink` instead of `blink()` with the parentheses, because we aren't calling the `blink()` function at that moment but rather telling `window.setInterval` to call the function with the name `blink`.
- Let's open `geolocation-0.html`[4]:

[4] http://cdn.cs50.net/2015/mba/classes/8/src8/geolocation-0.html

```
<!--

geolocation-0.html

Geolocates a user.

David J. Malan
malan@harvard.edu

-->

<!DOCTYPE html>

<html>
    <head>
        <script>

            function callback(position)
            {
                alert(position.coords.latitude + ', ' +
 position.coords.longitude);
            }

            function geolocate()
            {
                if (typeof(navigator.geolocation) != 'undefined')
                {
                    navigator.geolocation.getCurrentPosition(callback);
                }
                else
                {
                    alert('Your browser does not support geolocation!');
                }
            }

        </script>
        <title>geolocation-0</title>
    </head>
    <body onload="geolocate()"></body>
</html>
```

- First of all, how might Chrome know our coordinates? Our laptop probably doesn't have GPS like our phones might. One answer is that our IP address might reveal some idea of our location, but a more detailed location comes from the wireless access points around us. At some point in the past, companies drove around capturing wireless network IDs and their locations, so now our computer, given the signal strength of wireless networks around it, can sort of triangulate our location with high accuracy, even if some of them have changed. Our phones, too, transmit this data to Apple and Google about our location and nearby access points, for even more recent and accurate information.
  - # This sounds creepy, but unfortunately we probably all agreed to this when we hit "OK" on the Privacy Policies of apps and accounts we have.
  - # Turning off "Location Services" or the like might turn it off, but some apps, like Snapchat, require that permission in order for some of its functionality to work, so we end up having to enable it anyways.
- Looking back up at the source for `geolocation-0.html`, we see that the HTML really has one line, where the `body` element has an `onload` attribute that has the value of `geolocate`, presumably indicating that the `geolocate` function should be called when `body` has loaded.
  - # In fact, `onload` is an event handler that determines what should be done when something has loaded.
- So, looking for this `geolocate` function, we see it inside a `<script>` tag within the `<head>` element of the page.
  - # Within the `geolocation` function, we see an if-else condition that checks if the browser, or `navigator`, has the `geolocation` feature supported. If it is, we call the `getCurrentPosition` function of the `geolocation` feature of the browser.
  - # And we also see another function, `callback`, both on top of the `geolocation` function within the `<script>` tag, and also passed in to `getCurrentPosition`. "Callback" is actually a technical term for a function that runs after asynchronous code completes. That just means the code doesn't wait for the first function to finish running and then run the callback function, but rather go ahead and do other things, and when the first function finishes, then the callback function will be executed. This helps the browser, or programs in general that use this method, not appear to be frozen. (The downside to this is that by the time the first function returns, something might have changed and made the callback unnecessary or incorrect. Synchronous

code, on the other hand, avoids this problem by blocking on the function, or waiting for it to finish, but also has the tradeoff of sometimes appearing to be frozen.)

  # The `callback` function, then, takes the `position` argument that the `getCurrentPosition` passes to it, and looks inside for the `latitude` and `longitude` values, that we only knew were there by looking at JavaScript documentation.

  # `+ ', ' +` part just combines the two parts with a comma, to display it to the user in a nice way.

- Let's look at `geolocation-1.html`[5]:

[5] http://cdn.cs50.net/2015/mba/classes/8/src8/geolocation-1.html

```
<!--

geolocation-1.html

Geolocates a user, demonstrating an anonymous function.

David J. Malan
malan@harvard.edu

-->

<!DOCTYPE html>

<html>
    <head>
        <script>

            function geolocate()
            {
                if (typeof(navigator.geolocation) != 'undefined')
                {

 navigator.geolocation.getCurrentPosition(function(position) {
                        alert(position.coords.latitude + ', ' +
 position.coords.longitude);
                    });
                }
                else
                {
                    alert('Your browser does not support geolocation!');
                }
            }

        </script>
        <title>geolocation-1</title>
    </head>
    <body onload="geolocate()"></body>
</html>
```

Now we no longer have a `callback` function, but what looks just like a function is still where it would be, passed into `getCurrentPosition`.

- This is called an anonymous, or lambda, function, where the function isn't named but rather placed where it would be executed, and is common for callback functions that are only called once. And functions called multiple times in different places should still be named, since it would be bad design otherwise.
- Asynchronous code is particularly important in JavaScript because lots of events, like page loading, clicks, and key presses might all happen rapidly, and waiting on each one would make the page appear a lot slower.
- Another feature of many browsers is local storage, as seen in `storage.html`[6]:

[6] http://cdn.cs50.net/2015/mba/classes/9/src9/storage.html

```
<!DOCTYPE html>

<html>
  <head>
    <meta name="viewport" content="width=device-width">
    <script>

        function greet()
        {
            if (typeof(localStorage) !== 'undefined')
            {
                if (typeof(localStorage['counter']) === 'undefined')
                {
                    localStorage['counter'] = 0;
                }
                alert('You have visited ' + localStorage['counter'] + '
 time(s) before!');
                localStorage['counter']++;
            }
            else
            {
                alert('Your browser does not support localStorage!');
            }
        }

    </script>
    <title>storage</title>
  </head>
  <body onload="greet()"></body>
</html>
```

- This feature allows webpages to store information in each user's browser locally.
- By using the `localstorage` variable in JavaScript, we can put data inside it that we can retrieve and edit later.
- We can remember things like preferences and settings, that we don't really need to save on our own server, and it's also useful for offline work.
- Let's look at `ajax-0.html`[7]:

[7] http://cdn.cs50.net/2015/mba/classes/9/src9/ajax-0.html

```
<!--

ajax-0.html

Gets stock quote from quote.php via Ajax with jQuery, displaying price
 with alert.

David J. Malan
malan@harvard.edu

-->

<!DOCTYPE html>

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

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

        </script>
        <title>ajax-0</title>
    </head>
    <body>
        <form onsubmit="quote(); return false;">
            Symbol: <input autocomplete="off" id="symbol" type="text"/>
            <br/><br/>
            <input type="submit" value="Get Quote"/>
        </form>
    </body>
</html>
```

- If we type in a stock symbol and press `Get Quote`, we see the alert with the price, but the page hasn't changed:

- Looking at the HTML of the page, we see that the `form` element has an `onsubmit` attribute (that presumably runs when the form is submitted) and calls the `quote` function and then `return false`, or stopping the page from doing anything else.
- Looking up inside the `<script>` tag, we see this `quote` function that somehow "get a quote via JSON." First, it builds a `url` variable that links to `quote.php` on our server and includes the `val`, value, of the `symbol` element in the page (the `input` element` of the form has an `id` of `symbol`). Then it calls the `getJSON` function that's part of the jQuery library (the `$` indicates we're using a jQuery function, and all this we know from reading documentation online) with that `url` and displaying the `price` attribute of `data` when we get it.
- If we manually type in the URL, we'd see something like this:

  which is in JSON, which JavaScript can turn into an object we can access and use.
- Now let's look at `ajax-1.html`[8]:

[8] http://cdn.cs50.net/2015/mba/classes/9/src9/ajax-1.html

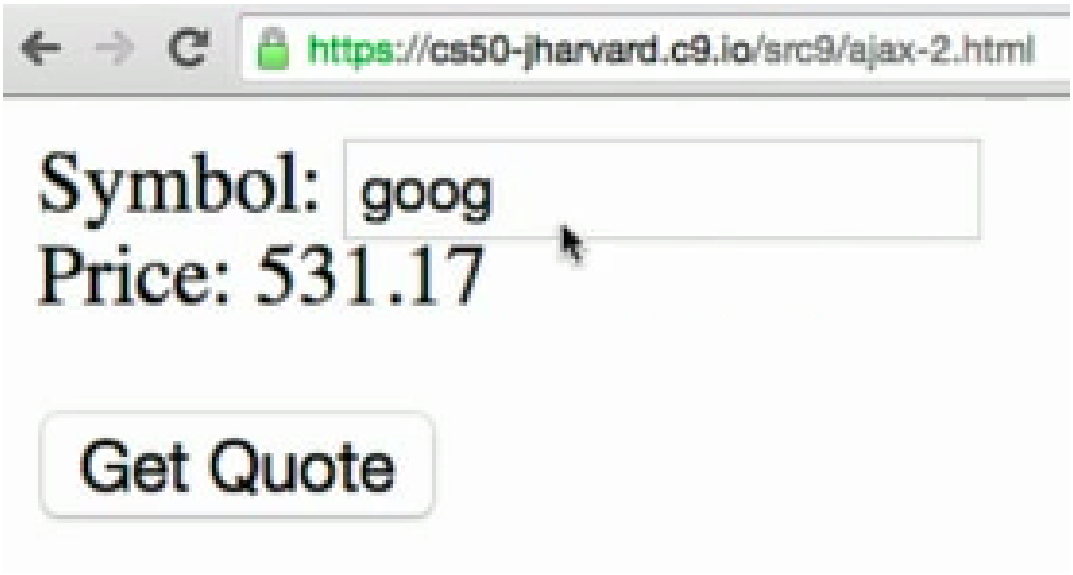
```
<!--

ajax-1.html

Gets stock quote from quote.php via Ajax with jQuery, displaying price
 with alert.

David J. Malan
malan@harvard.edu

-->

<!DOCTYPE html>

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

            $(function() {

                $('#quote').submit(function() {

                    var url = 'quote.php?symbol=' + $('#symbol').val();
                    $.getJSON(url, function(data) {
                        alert(data.price);
                    });
                    return false;

                });

            });

        </script>
        <title>ajax-1</title>
    </head>
    <body>
        <form id="quote">
            Symbol: <input autocomplete="off" id="symbol" type="text"/>
            <br/><br/>
            <input type="submit" value="Get Quote"/>
        </form>
    </body>
</html>
```

- This is another way of doing the same thing, but the HTML no longer indicates the function that should be run when the form is submitted. Instead, the JavaScript code finds the element with `id` of `quote` (which is the form), and attaches the function that gets the data to the `submit` event.
- Now let's look at `ajax-2.html`[9]:

[9] http://cdn.cs50.net/2015/mba/classes/9/src9/ajax-2.html

```
<!--

ajax-2.html

Gets stock quote from quote.php via Ajax with jQuery, embedding result in
 page itself.

David J. Malan
malan@harvard.edu

-->

<!DOCTYPE html>

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

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

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

- This time, the page itself changes, rather than having an alert:



- We've added this `<span>` tag within the form, and it's like a `<div>` tag in that it marks the beginning and end of a section of text. It's given an `id` of `price`, and in the JavaScript code we've set the `html`, or the actual HTML content inside the `<span>`, to be whatever `data.price` is.
- If we opened Chrome's inspector on a site like Twitter and looked at the Network tab, we'd probably see HTTP requests every once in a while, indicating that their page is indeed getting new data from a server.
- In the next class we'll talk about scalability, but for now, this video[10] will humorously examine the pros and cons of two popular database systems, MongoDB and MySQL.

[10] https://www.youtube.com/watch?v=b2F-DItXtZs