
Day 8

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

Cheng Gong

Table of Contents

| | |
|------------------|---|
| Recap | 1 |
| JavaScript | 5 |

Recap

- Last time we ended on this [xkcd¹](http://xkcd.com/327/) comic that depicted a SQL injection attack, or an attack on a database where a bad guy, or adversary, sends special strings that injects code that accesses or modifies the database, through single quotes and semicolons that changes the meaning of the original query statement.
- The owner of this car was either trying to be funny, or get the automated traffic plate recognition systems at a traffic light or such to delete its database, if its inputs weren't sanitized:

¹ <http://xkcd.com/327/>



- I'm a little confused about atomicity vs. transactions. Wikipedia tells me that atomicity means a series of operations that either all occurs or nothing occurs. But it seems to me that this definition is for "transactions" from today's lecture. I thought atomicity means that instead of adding additional rows of data, we will just update the existing rows. Which definition is accurate?

Atomicity is the property that everything in a group is done together or not at all, and transactions is one implementation where everything in a transaction is guaranteed by the database to be completed or not at all. Atomicity isn't limited to particular operations. Rather, one example of a situation where we might want atomicity is

when we want to do both checking and updating the rows at the same time. If we didn't have this guarantee, we would have a race condition where whoever updated the row first would get the correct result, since two users might not return at the same time after checking some row.

- If my webpage requires users to create an account with a password with maximum X characters and they include a bunch of "dangerous" characters, couldn't they exceed the maximum number of characters?

Websites should actually want you to use symbols to make your passwords more secure and harder to guess. To implement this, we could use pseudocode like this:

```
.....  
if form.validate():  
    # let user register  
else:  
    # don't let user register  
.....
```

`form` is some variable, and `validate` is some function that looks over it and returns true or false depending on whether the inputs were valid, so the condition allows us to validate input.

- Should I be skeptical about saving my passwords for various sites on chrome? What kind of encryption does it use? Is it easily hackable? Thanks!!

Two-factor authentication is using a second method, in addition to a password, such as having a code texted to your phone, to log into a website, so even if you lost your password it would be difficult for someone else to gain access. But the tradeoff is the inconvenience it takes to log in, or if you lost your phone.

But to answer the question, based on Chrome's documentation, "These passwords are stored in the same system that contains your saved passwords from other browsers. On a Mac, Google Chrome uses the Keychain Access to store your login information." So this is just stored on your computer, which might not be secure against roommates, but just as secure as everything else you save. In addition, "When you sign in to Chrome and enable sync, Chrome keeps your information secure by using your Google Account credentials to encrypt your synced passwords. Alternatively, you can choose to encrypt all of your synced data with a sync passphrase. This sync passphrase is stored on your computer and isn't sent to Google." So you can choose to encrypt it with your account information (and Google would have access), or a completely different passphrase (which is

just like a password but longer), and again we have to make the tradeoff between convenience and security.

In addition, we recommend these password managers, where these are applications that remember and save your passwords for you, and encrypt them with one longer password, so you don't have to remember all of them:

agilebits.com/onepassword

lastpass.com

...

These prevent the problem of using the same password for all the websites, but someone would have all your passwords if they had your computer and the password. Another benefit is that they can randomly generate passwords with numbers and symbols that are much harder for hackers to crack by guessing all possible combinations, and since the apps save them, we don't have to worry about how complex the passwords are.

Facebook Login is a different situation, where websites allow users to login with a Facebook account, so they don't have to take as much time to sign up for a separate account, which might increase the chances that they do create one.

And Incognito Mode is only telling your browser on your computer to not keep history or cookies, but a keystroke logger can still record passwords, and the server on the other side will still see your IP address and requests.

- Very minor thought: when you ask at the end of an assignment how much time it took, I don't know if you're asking about just filling out the answers on the form, or the assignment overall.

We're asking about the assignment overall!

- I noticed that some of the links we received for downloads start with "cdn." I was wondering what does cdn stand for and how is this a part of the network structure / where is this data kept?

CDN stands for content delivery network, or a fancy way of describing servers that someone else owns that we put our static content (files that don't change) on, so people can access them more quickly. Dynamic or generated content, like parts of a website that require the Python backend, are kept on application servers that does processing first. CS50 in particular uses Amazon as its CDN.

- I've been staring at Project 2 for the past hour and don't know where to even start. In the `quote.html` file and the other python file, are there any tips on how to start adding code? Any help much appreciated, just so I can get a start on the project.

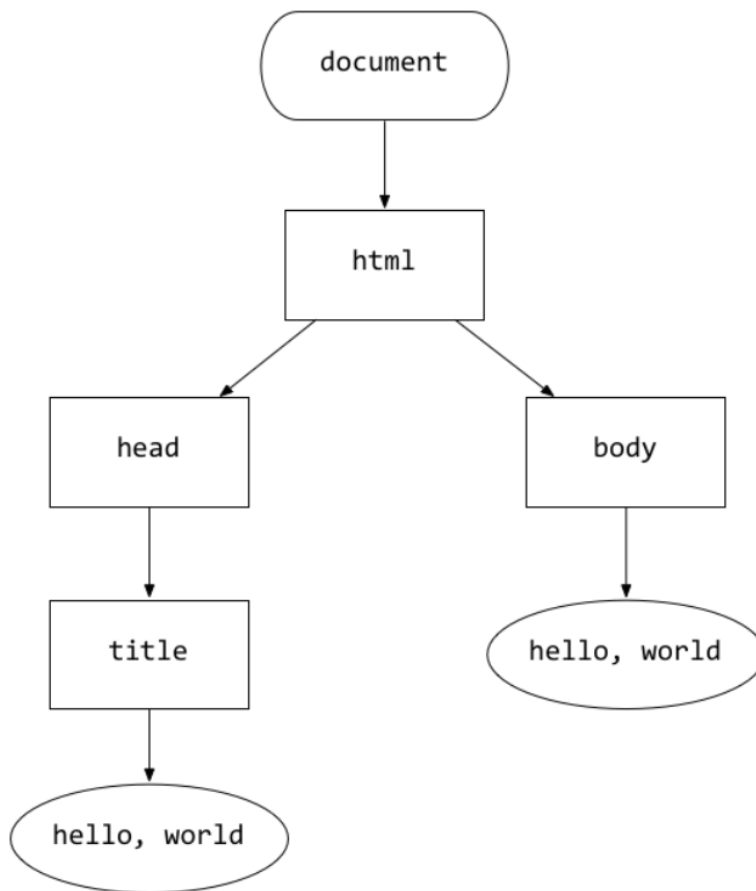
This question makes us sad, only because it was sent through the anonymous form so we couldn't really reply. Use CS50 Discuss or email us in the future for questions like this. For Project 2 in particular, we needed to make two changes. We start by looking at `index.html`, from which we can see the form is sent to the URL `/quote`, or the server's domain name and then `/quote`. The form will submit a value for `symbol`, and since we know the method is GET, it will add `?symbol=`, followed by the name of the stock, to the URL. So we move on to `quote.html`, where we see that `{{` and `}}` are used to put the values in, and then going back to `application.py`, we needed to add the actual processing that got the stock quote from the Yahoo Finance library, from the symbol passed in through the HTTP request. We can figure out how to do this by reading existing code or examples, Googling around, trying different things, and reading the documentation.

JavaScript

- So far, we've been doing server-side programming, where the code is being run on the server. JavaScript, even though we will save its code on the server, will be downloaded and run by a browser when it visits a webpage that includes it.
- Back in the 90s, pop-ups were popular, and all it took to implement them was a line like `window.open` that opened a new window with JavaScript, but now browsers tend to recognize code like that and not run them by default.
- Dragging and dropping the map in Google Maps, too, is an example of JavaScript in use, where more information is being downloaded from the server without having to reload the entire page.
- Generally, we want to think of HTML as a tree, called the Document Object Model:

```
<!DOCTYPE html>

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



- With JavaScript, we can modify this tree and elements within the tree.
- Turns out, the code is pretty similar to what we've seen so far with Python.
- Conditions look like this:

```
.....  
if (condition)  
{  
    // do this  
}  
else if (condition)  
{  
    // do that  
}  
else  
{  
    // do this other thing  
}  
.....
```

Notice that now we do need the curly braces to contain our branches.

- Booleans look like this:

```
.....  
if (condition || condition)  
{  
    // do this  
}  
.....
```

```
.....  
if (condition && condition)  
{  
    // do this  
}  
.....
```

`||` means `or`, and `&&` means `and`.

- Switches look like this:

```
switch (expression)
{
    case i:
        // do this
        break;

    case j:
        // do that
        break;

    default:
        // do this other thing
        break;
}
```

This is like an `else if` where `expression`, like a variable `x`, is matched with a case `i` like `1` or `2`, and whatever the expression matches will be the branch of code that is executed.

- Loops look like this:

```
for (initializations; condition; updates)
{
    // do this again and again
}
```

We'll see an example of this soon.

```
while (condition)
{
    // do this again and again
}
```

```
do
{
    // do this again and again
}
while (condition);
```


This is like a `while` loop, except we always run the code inside once, before checking the condition, as opposed to checking the condition first.

- Arrays look like this:

```
var numbers = [4, 8, 15, 16, 23, 42];
```

An array is like a list of the (generally) same type of item, over and over again.

- Strings are declared like this:

```
var s = "hello, world";
```

JavaScript is loosely typed, which means that we can just call all variables `var` instead of its actual type, and the interpreter figures out what the type should be.

- And objects:

```
var quote = {symbol: "FB", price: 82.04};
```

An object is a data structure with key and value pairs, in this case the keys are `symbol` and `price` and the values are `FB` and `82.04`. We can have an array of `["FB", 82.04]`, but then there won't be any metadata about what each item is.

- We can even do a loop like:

```
for (var i in object)
{
    // do this with object[i]
}
```

- There are also event handlers:

onblur

onchange

onclick

This is when the mouse clicks, or goes down and up.

onfocus

onkeydown

onkeyup

onload

onmousedown

This is when the mouse is held down, like for dragging.

onmouseup

onmouseout

onmouseover

onmouseup

onresize

onselect

This is when a text box is selected.

onsubmit

...

- An event in a program is something that happens, and we can "listen" for events like the mouse being clicked, or elements on the page being dragged, and write event handlers, or code that reacts to them. We'll see examples of these, too.

To make this happen, basically web browsers have an infinite loop that check for these events, and the more resources we try to listen for, the more resources it might take, which might cause the entire web browser to slow down.

Some websites that prevent you from typing certain characters might be using `onkeydown` to detect when a key is pressed, and allow or disallow whether the character actually gets typed in.

- Let's take a look at `form-0.html`²:

² <https://cdn.cs50.net/2015/mba/classes/8/src8/form-0.html>

<!--

form-0.html

A form without client-side validation.

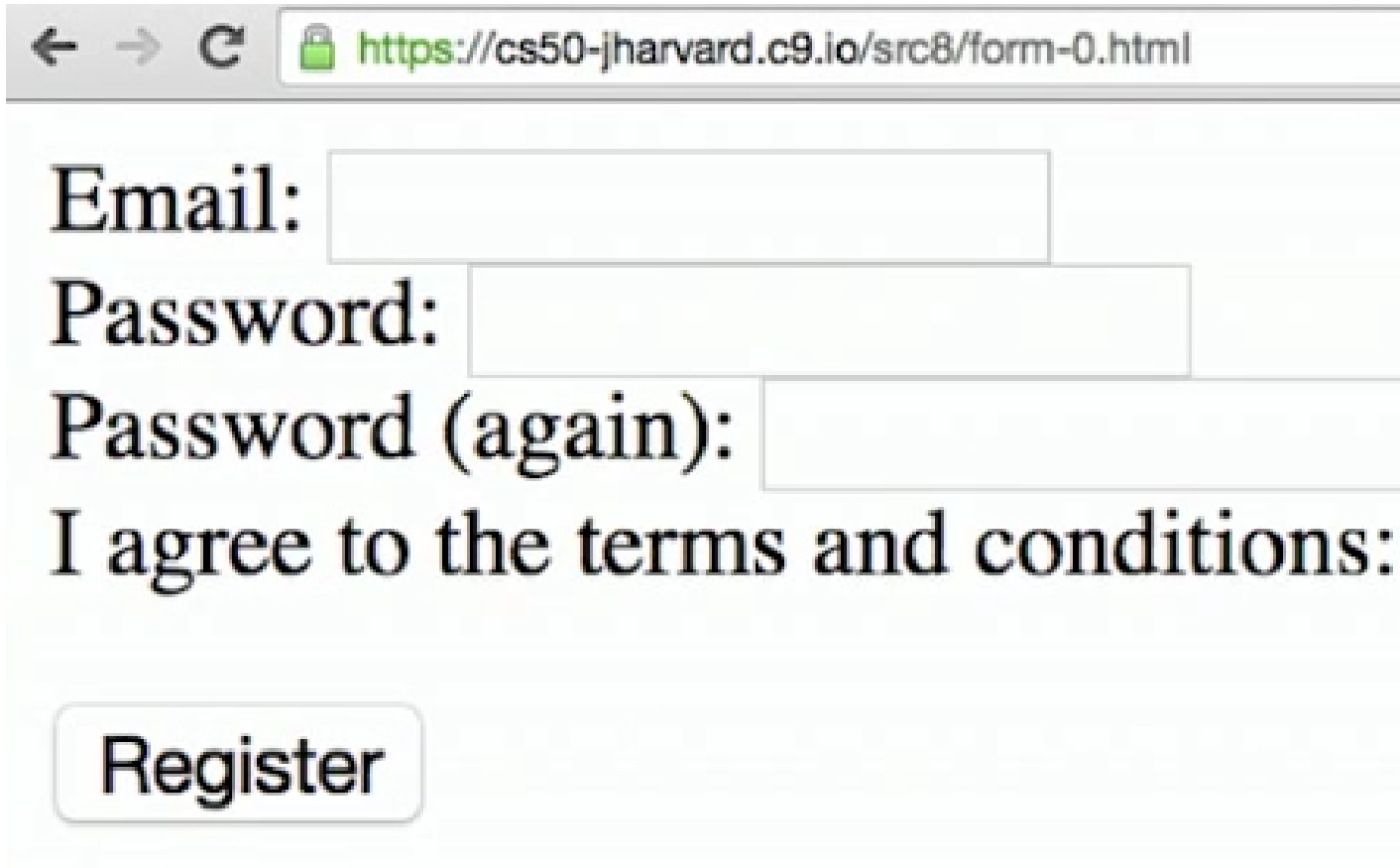
David J. Malan
malan@harvard.edu


-->

```
<!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'll turn on our webserver by using `apachectl start`, and if we go to that page, we see this:



← → ↻  <https://cs50-jharvard.c9.io/src8/form-0.html>

Email:

Password:

Password (again):

I agree to the terms and conditions:

- If we press `Register`, though, we see this:

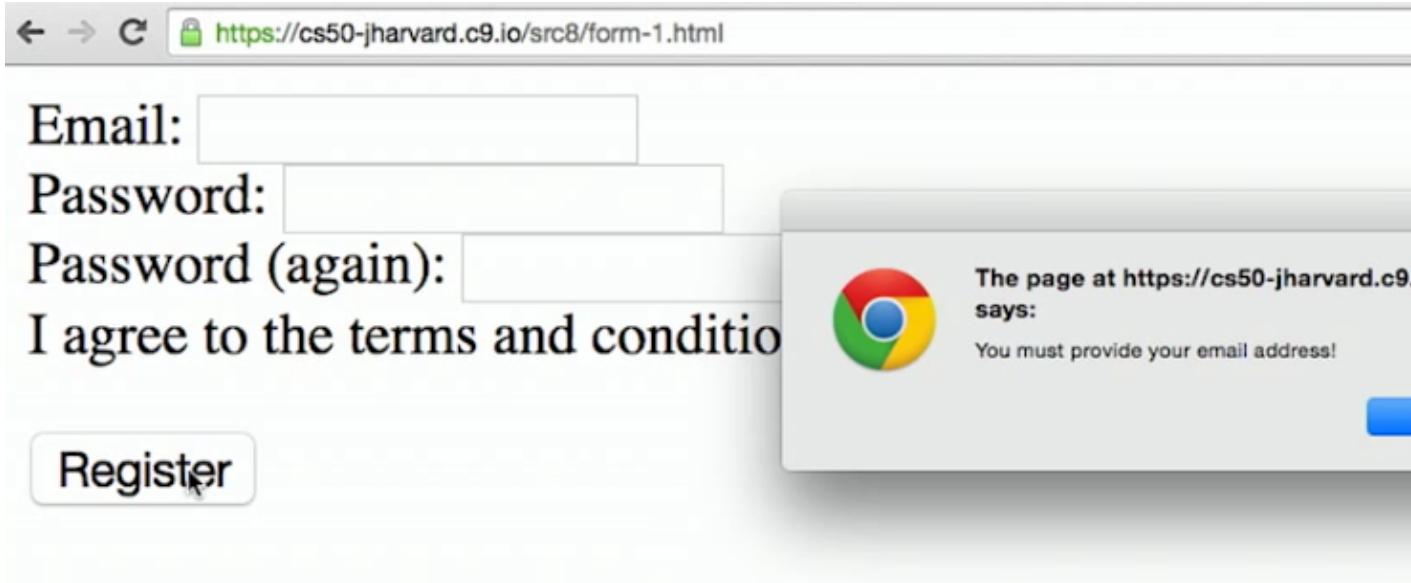


← → ↻  <https://cs50-jharvard.c9.io/src8/register.php?email=&password=&cont>

You are registered! (Well, not really.)

- This example has no validation, which is why it accepted the blank form, so we could use the `onsubmit` event listener to run JavaScript code when the form is submitted, and control whether the form is actually sent to the server or not.
- Now if we look at `form-1.html`³ and try to submit a blank form, we get an error message:

³ <https://cdn.cs50.net/2015/mba/classes/8/src8/form-1.html>



- We can look at the [source code](https://cdn.cs50.net/2015/mba/classes/8/src8/form-1.html.src)⁴:

⁴ <https://cdn.cs50.net/2015/mba/classes/8/src8/form-1.html.src>

```
<!--
```

```
form-1.html
```

```
A form with client-side validation.
```

```
David J. Malan  
malan@harvard.edu
```

```
-->
```

```
<!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');
```

```
      // onsubmit
```

```
      form.onsubmit = function() {
```

```
        // validate email
```

```
        if (form.email.value == '')
```

```
        {
```

```
          alert('You must provide your email address!');
```

```
          return false;
```

```
        }
```

```
        // validate password
```

```
        else if (form.password.value == '')
```

```
        {
```

It looks like we have a `form` element that goes to another page called `register.php` in line 19 with some fields and a `submit` button.

On line 30 we have a new tag called `script` that just puts JavaScript code into our webpage. In this code, first, we make a new variable (`var`), that we call `form`. We could call it anything we want, since it's like a label pointing to some object. And the object that we want to label `form` is an object on the page (`document`) that has an `id` named `registration`. (`getElementById` is a function that we can call to get elements.) And if we look up back at line 19, we gave our HTML form an `id` of `registration`, so we can find it later in our code.

On line 35, we start a block of code that runs when the `form` is submitted, i.e. when the `onsubmit` event happens. Inside, we see that we're checking for if the `email` field's `value` is empty, `'`, and if it is, we create an `alert`, error message, and `return false`, or stop the form submission. We do the same for all the fields, and if no errors were created, we `return true`, which allows the form to be sent.

We should also validate the form on our server, like we did with the `frosh-im` examples, since some browsers might have JavaScript turned off. But having this validation in JavaScript gives a better user experience, since the form doesn't have to be submitted to the server, so it's faster.

We don't validate that the email is in a correct format, but if we search on Google for something like "validate email address in JavaScript", we can find snippets of code that does this, by checking if the string is of a certain pattern ([regular expression](#)⁵).

- Let's look at `dom-0.html`⁶:

⁵ https://en.wikipedia.org/wiki/Regular_expression

⁶ <https://cdn.cs50.net/2015/mba/classes/8/src8/dom-0.html.src>

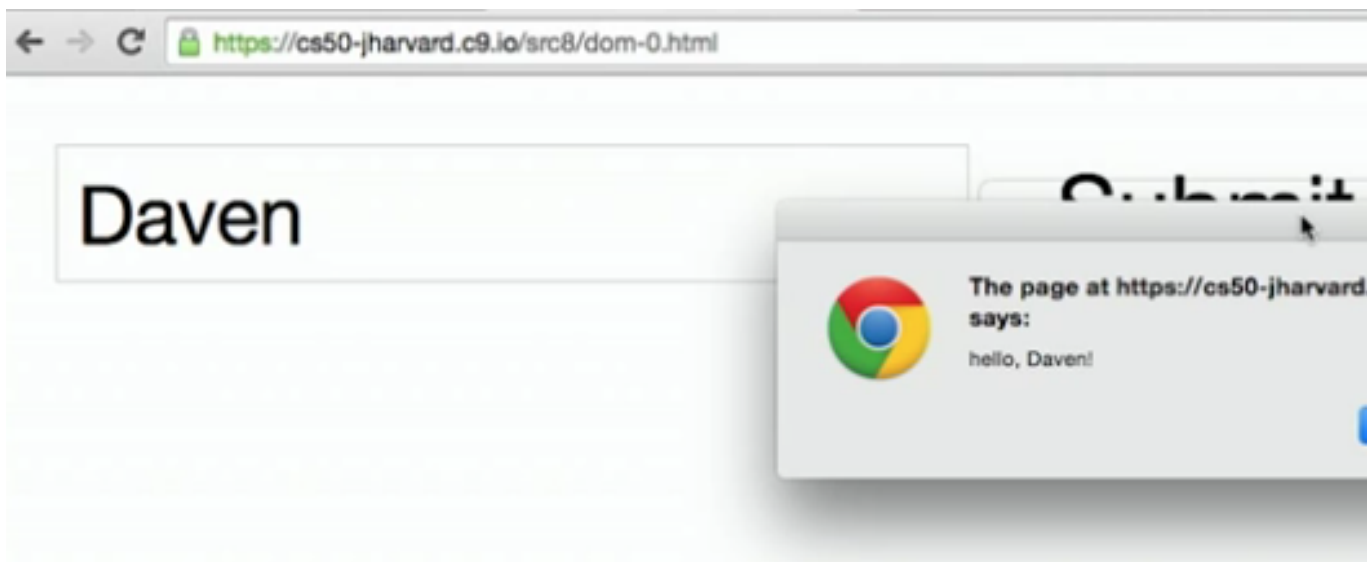
```
<!DOCTYPE html>

<html>
  <head>
    <script>

      function greet()
      {
        alert('hello, ' + document.getElementById('name').value +
'!');
      }

    </script>
    <title>dom-0</title>
  </head>
  <body>
    <form id="demo" onsubmit="greet(); return false;">
      <input id="name" placeholder="Name" type="text"/>
      <input type="submit"/>
    </form>
  </body>
</html>
```

If we go to [the page⁷](https://cdn.cs50.net/2015/mba/classes/8/src8/dom-0.html) and submit a name, we see that we get it back in an alert:

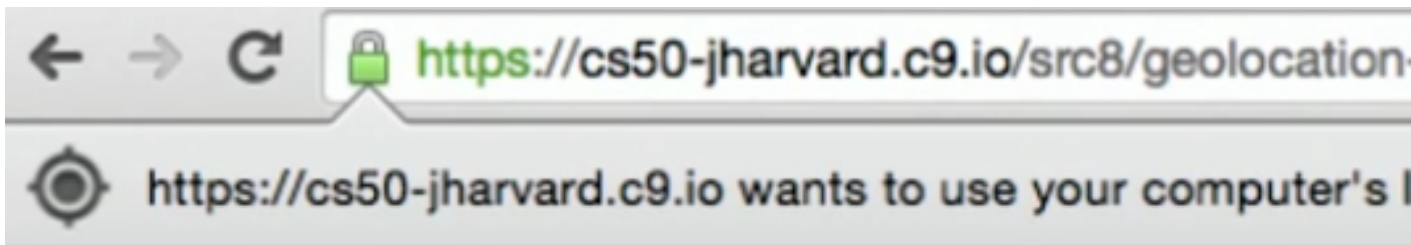


⁷ <https://cdn.cs50.net/2015/mba/classes/8/src8/dom-0.html>

We've moved our `script` element to the `head` section of our HTML page, which is slightly better design, because it's now separated somewhat from our HTML that represents the body of the page, and inside we have a function called `greet`. That function calls another function, `alert`, which tells the browser to show that little pop-up, and display `hello, ` and whatever the `value` of the `name` element on the page is. (The `+` symbols combine strings.)

Now if we look at the `form` HTML on line 16, we see that instead of going to another page when it is submitted, the `onsubmit` action calls the `greet` function and stops by returning `false`.

- `dom-1.html`⁸ is slightly different, with our script moved to the bottom of the page. This way, the body of the page will be loaded first and displayed, so the website doesn't seem super slow if the script was on someone else's server, or a really big library.
- `dom-2.html`⁹ uses a library called `jQuery`¹⁰, which has more pre-written functions that we can use.
- Now let's try `geolocation-0.html`¹¹, which creates this:



- Then we get this:

⁸ <https://cdn.cs50.net/2015/mba/classes/8/src8/dom-1.html.src>

⁹ <https://cdn.cs50.net/2015/mba/classes/8/src8/dom-2.html.src>

¹⁰ <https://jquery.com/>

¹¹ <https://cdn.cs50.net/2015/mba/classes/8/src8/geolocation-0.html>



- An older feature was blinking text on the page, which we can look at an example of in `blink.html`¹²:

¹² <https://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>

The text is in a `div`, or division, of the page that we've given an `id` of `greeting`, and in our JavaScript code we have a function that changes the `visibility` from `visible` to `hidden` and vice versa (the precise syntax and CSS properties that control visibility we've looked up via Google), and call this function once every 500ms with the `window.setInterval` function.