Week 9

This is CS50. Harvard University. Fall 2014.

Cheng Gong

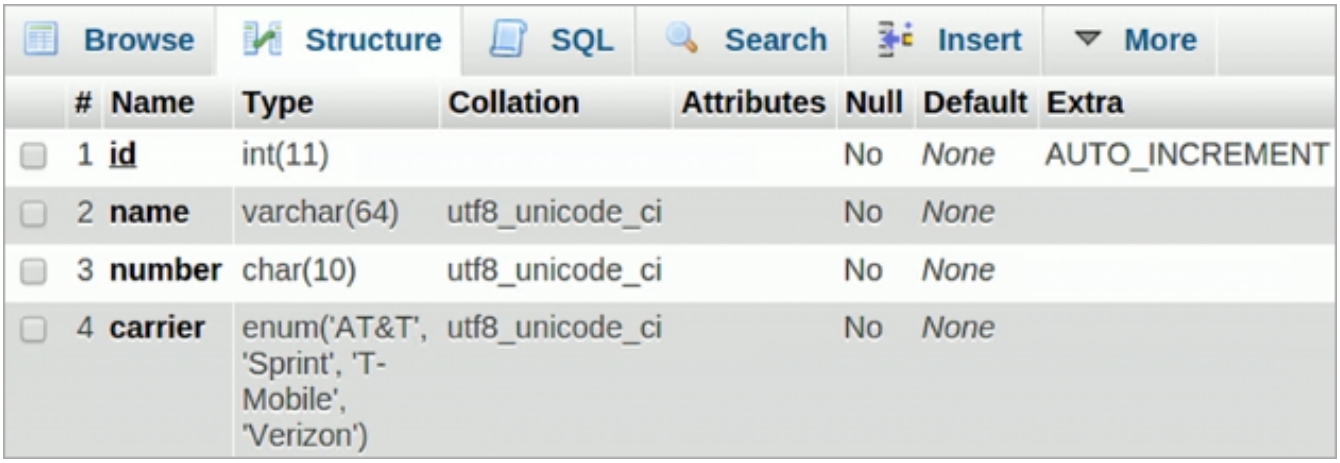
Table of Contents

Texting, fixed!

- Today we'll wrap up PHP and SQL from last time, talk a little about security, and then introduce a client-side programming language called JavaScript.
- Last Wednesday, we left off with a buggy PHP program that was supposed to iterate over the rows of names and numbers in our database and send out text messages.

  # As he was trying to fix it in lecture, he got several texts from friendly students that read "Hi! It's Margot!", "You got this, David!", "We believe in you!", "Nearly there!", and best of all, "you're blowing it!!!!" And later, by watching the livestream, someone even sent "Hello from London, England!"

- So David's been debugging his code (using up all 5 of his late days at once, even though that's against the rules).
- One change he's made is to the `users` table, changing the type of `carrier` from `VARCHAR` to `enum`:



Browse | Structure | SQL | Search | Insert | More

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(11)			No	*None*	AUTO_INCREMENT
2	name	varchar(64)	utf8_unicode_ci		No	*None*	
3	number	char(10)	utf8_unicode_ci		No	*None*	
4	carrier	enum('AT&T', 'Sprint', 'T-Mobile', 'Verizon')	utf8_unicode_ci		No	*None*	

`enum` is a type found also in C, where you can enumerate a bunch of constants and assign them values without hardcoding numbers. In SQL you can do the same, and in this case we are specifying which four strings the `carrier` field can be.

- David also figured out how to get email working, since sending emails to certain address set up by carriers will be sent to phones as text messages. And now if he puts his name into the form, it works.
- Let's take a look at the source code in `src9m/php`[1], in particular `public/index.php`[2]:

```
<?php

    // configuration
    require("../includes/config.php");

    // render portfolio
    render("form.php");

?>
```

- It looks like all we're doing here is rendering `form.php`[3], located in `php/templates`:

[1] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/php/
[2] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/php/public/index.php
[3] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/php/templates/form.php

```
<h1>Here?</h1>
<form action="here.php" method="post">
    <input name="name" placeholder="Name" type="text"/>
    <input name="number" placeholder="Phone Number" type="text"/>
    <select name="carrier">
        <option value=""></option>
        <option value="AT&amp;amp;T">AT&amp;amp;T</option>
        <option value="Sprint">Sprint</option>
        <option value="T-Mobile">T-Mobile</option>
        <option value="Verizon">Verizon</option>
    </select>
    <input type="submit" value="Here!"/>
</form>
```

This form submits to a file called `here.php` with the POST method, and uses a `select` menu to allow you to choose from the four `option`s.

- `here.php`[4], meanwhile, looks like this:

[4] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/php/public/here.php

```
<?php

    require("../includes/config.php");

    // ensure user has provided name
    if (empty($_POST["name"]))
    {
        apologize("Missing name!");
    }

    // ensure user has provided number
    if (empty($_POST["number"]))
    {
        apologize("Missing number!");
    }

    // ensure user has provided carrier
    if (empty($_POST["carrier"]))
    {
        apologize("Missing carrier!");
    }

    // remove non-digits from number
    $number = preg_replace("/[^\d]/", "", $_POST["number"]);

    // ensure number is 10 digits
    if (strlen($number) != 10)
    {
        apologize("Invalid number!");
    }

    // text user
    if (text($number, $_POST["carrier"], "so glad you're here,
 {$_POST["name"]}! lol")) ❶
    {
        render("success.php", ["name" => $_POST["name"]]);
    }
    else
    {
        apologize("Could not text you!");
    }

?>
```

    - There's a lot of error checking, but line 33 is the most important part, calling the `text` function to send a text. (And you can check out the new and improved code in that function in `functions.php`[5].)

- If you wish to send yourself a text, you can go to http://cs50.harvard.edu/here (David has since taken down the form, unfortunately).
    - Though, in lecture, it didn't work for some people, probably because we tried to send too many emails through Harvard's mail servers at once.

Announcements

- RSVP for CS50 lunch this Friday, 11/7, 1:15pm, at http://cs50.harvard.edu/rsvp
- If you are thinking of a concentration or secondary in CS, the SEAS Advising Fair is this Wednesday, 11/5, 4pm - 5:30pm, in the Maxwell Dworkin lobby. (You can also see more detail via Facebook[6])
- Also, Mark Zuckerberg's ID on Facebook was actually 4, not 3, because he used more test accounts. In fact, we can see this information by going to http://graph.facebook.com/zuck:

```
{
   "id": "4",
   "first_name": "Mark",
   "gender": "male",
   "last_name": "Zuckerberg",
   "link": "https://www.facebook.com/zuck",
   "locale": "en_US",
   "name": "Mark Zuckerberg",
   "username": "zuck"
}
```

    - The format here is JavaScript Object Notation (JSON), returned by Facebook's Application Programming Interface (API), which is a fancy way of saying we can use a program to request this information, which will be returned to us in a machine-readable format.

[5] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/php/includes/functions.php
[6] https://www.facebook.com/events/799663196759849/

- CS50 has its own APIs too:
  # CS50 Courses API
  # CS50 Food API
  # CS50 Maps API
    You can query all of these APIs and get back information you can incorporate into your final projects.
- Speaking of which, these are the milestones for the final project[7]:
  # pre-proposal
  # proposal
  # status report
  # CS50 Hackathon
  # implementation
  # CS50 Fair
- For some possibilities, CS50 has a full range of hardware you may borrow for your final project:
  # Philips hue bulbs
    # They actually have an API where each bulb is connected wirelessly to a bridge, which accepts messages like

```
PUT /api/newdeveloper/lights/1/state HTTP/1.1

{"on":true, "bri":128, "transitiontime":0}
```

    + which uses a method we haven't yet seen, `PUT`, to send text that almost looks like an array, but is again in JavaScript Object Notation (JSON). One key is `on`, set to `true`, `bri` means brightness, set to `128`, and `transitiontime` is `0`.

    # To send this message in the Terminal, we can use the `curl` command:

[7] http://cdn.cs50.net/2014/fall/project/project.html

```
curl -X PUT -d '{"on":true, "bri":128, "transitiontime":0}'
 "http://10.0.1.3/api/newdeveloper/lights/4/state"
```

- We're using the `PUT` method to send `-d`, data, in the single quotes, to the URL that follows, and `curl` will handle the rest of the headers and protocol requirements.

We can even change the color or flash them through the API.

Google Glass

Arduino Uno

Tiny computers on a circuit board you can connect things to.

Myo Gesture Control Armband

This just arrived, and this clip from Myo[8] shows the future where we can control our devices with gestures.

Leap Motion Controller

These devices allow you to motion in front of a Mac or PC, as though you had Xbox Kinect, as shown in this clip[9].

…

- So even though we might not know how these devices work at a hardware level, in accordance with the theme of layering and abstraction, we can simply use software to talk to them to make something useful and fun.
- Check out the seminars[10] that are going on if you want to learn more new things!

News

- Cheng (hey, that's me!) recently sent in this article, with the headline I'm terrified of my new TV: Why I'm scared to turn this thing on — and you'd be, too[11].

[8] https://www.youtube.com/watch?v=oWu9TFJjHaM
[9] https://www.youtube.com/watch?v=gby6hGZb3ww
[10] http://cs50.harvard.edu/seminars
[11] http://www.salon.com/2014/10/30/im_terrified_of_my_new_tv_why_im_scared_to_turn_this_thing_on_and_youd_be_too/

- Now that we've learned a bit about HTTP and the web, we can look at articles like these and understand them a bit better.
- David reads a few excepts, that describe how a smart TV, recently purchased by the author of the article, has features like streaming multimedia and web browsing, but also a camera and microphone (for voice recognition) that might be watching and recording everything you do.
- Another article, from The Daily WTF, titled The Spider of Doom[12], describes a developer named Josh whose content management system (for employees to manage a website) suddenly lost all of its content. Turns out, Google's web-crawling spider followed delete links that a user had copy and pasted into another page, and since Google's spider doesn't accept cookies or JavaScript, the CMS was unable to check whether it was logged in, or redirect it away.
- So they had URLs that looked like `http://pset7/sell.php?symbol=GOOG` (which you might see in Problem Set 7), but implementing these actions through `GET` might not be the best idea, since bots, or spiders, might get to those URLs, and sell the stocks.
- Indeed, the real problem in story about the CMS is that it relied only on cookies and JavaScript to check for login status, but you (especially in Problem Set 7) should always check for user logins on the server side, with `login.php` or the like.
- Finally, realize that applications like Snapchat can never promise that photos can only last for 5, 10, or however many seconds. (One easy way is to just take a photo with another phone while you have the photo open.) Screenshots can also be taken, but at least the application can tell you. But worse yet, recently hundreds of thousands of Snapchat pics were leaked[13] because those users used a third-party website to save their snaps, and that website was hacked, leading to the leak. (It should also be noted that Snapchat isn't sending pictures and messages securely to your phone, but allowing a third-party website to receive and store them. If you're interested in more technical details, see http://kennywithers.com/featured-online-marketing-articles/the-snappening-snapchat-accounts-hacked/.)

[12] http://thedailywtf.com/articles/The_Spider_of_Doom
[13] http://www.washingtonpost.com/news/morning-mix/wp/2014/10/13/a-massive-leak-of-private-snapchat-pics-and-an-era-when-even-disappearing-photos-can-reappear/

SQL

- Let's wrap up SQL, with an introduction to a few more things that might not be required in Problem Set 7, but might come in useful for final projects.
- With MySQL, you have the choice of various **storage engines**:
  - # InnoDB
  - # MyISAM
  - # Archive
  - # Memory
  - # …

    And these are essentially the file systems or formats for how your database will be stored as.
- The most popular are the first two, InnoDB, and MyISAM. But first, a story (a retold one from the days when David took CS161, Operating Systems). Suppose you and your roommate share a fridge, and you both like milk. So you come home one day, and your roommate is still in class, and you see that there's no more milk in the fridge. So you leave for CVS to buy some milk. Meanwhile, your roommate gets home and realizes that there's no milk too, and happens to go to the other CVS, and by the time you both return, you have too much milk. This was because you and your roommate didn't communicate with each other (texting didn't exist back in David's time, so this story made more sense then).
- So, in CS terms, we need to **lock** this resource. Imagine that there's a variable called `milk` with the value `0`, and you don't want your roommate to check the value of `milk` and see `0` before you have a chance to update it to `1`, so you need to lock that variable until you're done using it.
- So in Problem Set 7 we give you a line of sample code that looks like this:

```
INSERT INTO table (id, symbol, shares) VALUES(6, 'FREE', 10)
    ON DUPLICATE KEY UPDATE shares = shares + VALUES(shares)
```

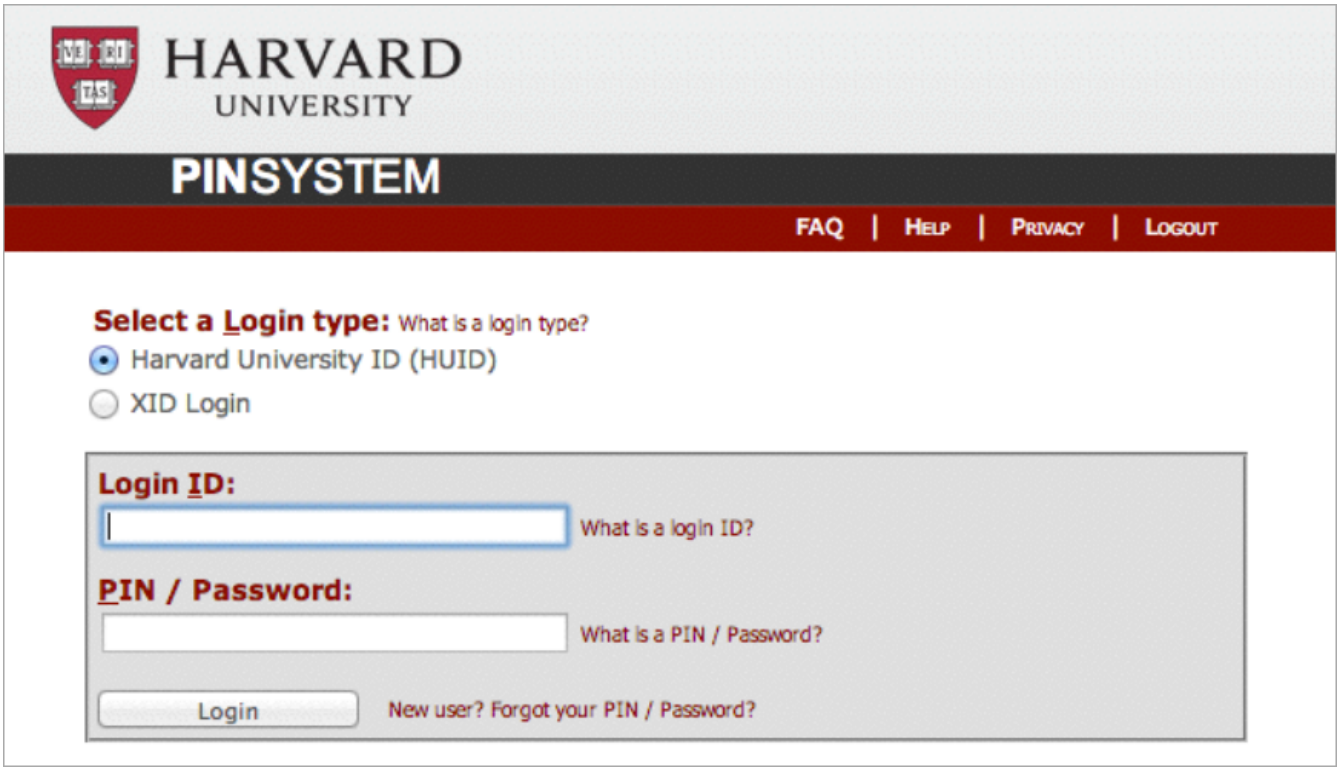
  - # In this line, if we wanted to buy more shares of a stock called `FREE`, we need to do two things at once: check how many shares we have, and update it.

- By combining them into one line, we ensure that these operations happen together or not at all (**atomicity**). Otherwise, an interruption might cause unexpected behavior. For example, let's say you're implementing an ATM for a bank. And you want a feature where users can transfer money from one account to another, but we don't want some scenario where someone can use two ATMs at the same time to deduct some amount of money, but having that recorded only once. So the SQL you want to use might look like this:

```
START TRANSACTION;
UPDATE account SET balance = balance - 100 WHERE account = 2;
UPDATE account SET balance = balance + 100 WHERE account = 1;
COMMIT;
```

  # You can call the `query` function with `START TRANSACTION;` and then as many queries as you'd like, and they won't be completed until you call `query` with `COMMIT`, but then they will be completed together, one after another.

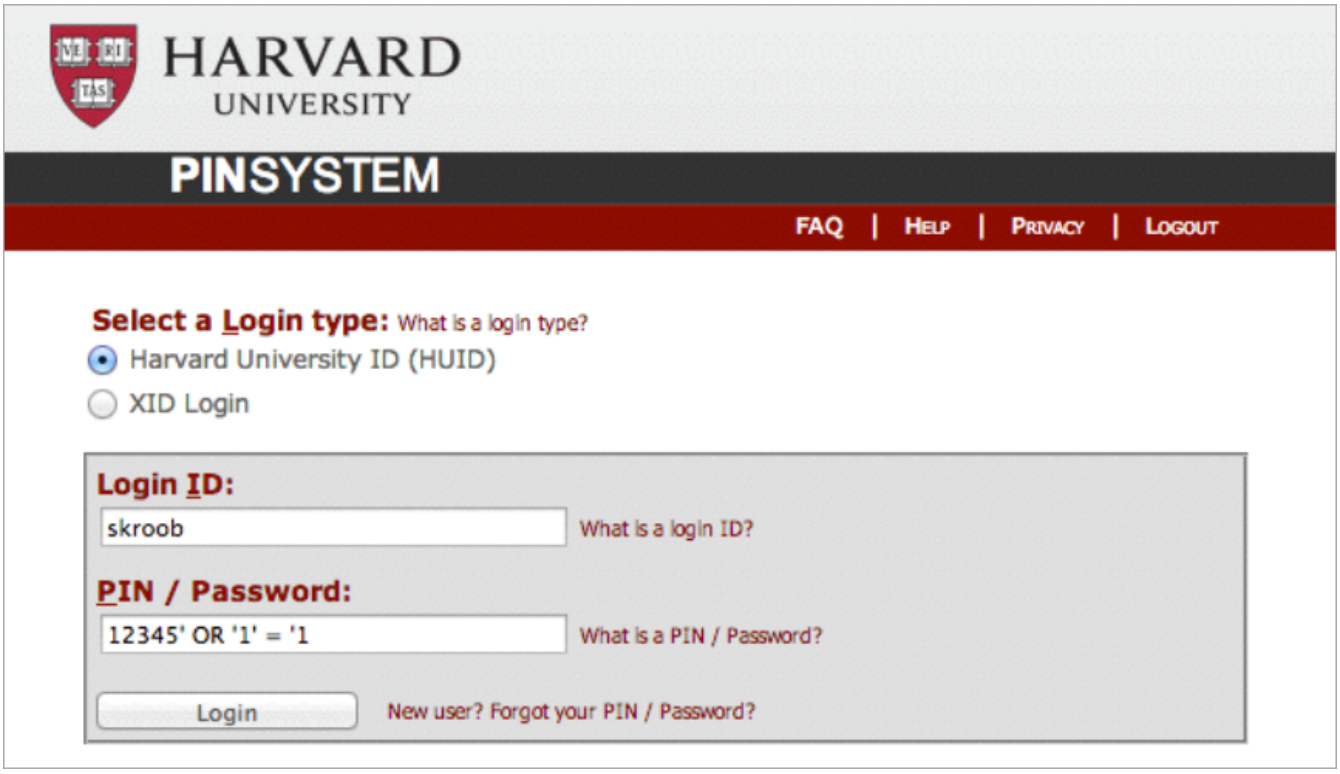
- **SQL injection attacks** are also a concern. Though the familiar Harvard PIN system isn't vulnerable to this attack, we'll use it for the sake of discussion.
- The PIN login screen looks like this:



- Suppose that the code, somewhere behind this page, includes bits like this:

```
$username = $_POST["username"];
$password = $_POST["password"];
query("SELECT * FROM users WHERE username='{$username}' AND
 password='{$password}'");
```

- First we get the variables `$username` and `$password`, and substitute them into our query, which looks (and is) correct.
- But a user could input something like this: (David changed the PIN field's usual bullets to actual text to reveal what the user has typed.)



`skroob` is the username, but the password field is a bit fishy with `12345' OR '1' = '1`, especially with the single quotes.

- If we substitute that info into the form, the query will now look like this (note that `skroob` has left off single quotes on the outside of his password, assuming that the code for the query will, which it does):

```
$username = $_POST["username"];
$password = $_POST["password"];
query("SELECT * FROM users WHERE username='skroob' AND password='12345' OR
 '1' = '1'");
```

So now we're selecting everything from the `users` table, `WHERE` username is `skroob` `AND` password is `12345`, `OR` `1 = 1`. Since `1 = 1` is always true, every row from the table will be selected.

- If we use `?`s with CS50's `query` function, instead of substituting the variables directly, then the single quotes will be escaped:

```
$username = $_POST["username"];
$password = $_POST["password"];
query("SELECT * FROM users WHERE username=? AND password=?", $username,
 $password);
```

- So the moral is to always make sure your input is clean (escaped)!
- And now you can understand this xkcd[14].

JavaScript

- Now let's talk about JavaScript. PHP's syntax is very similar to C's syntax, and fortunately JavaScript is pretty similar too.
- In JavaScript, there's also no `main` function.
- Conditions look like this:

[14] http://xkcd.com/327/

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

- Booleans look like this:

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

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

- Switches look like this:

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

    case j:
        // do that
        break;

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

- Loops look like this:

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

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

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

- Arrays look like this:

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

    # In JavaScript you declare a variable not by a type or `$`, but by saying `var` before it, and that makes JavaScript loosely typed. Variables have types, but you don't need to explicitly declare them.

- Strings are declared like this:

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

- And objects:

```
var quote = {symbol: "FB", price: 79.53};
```

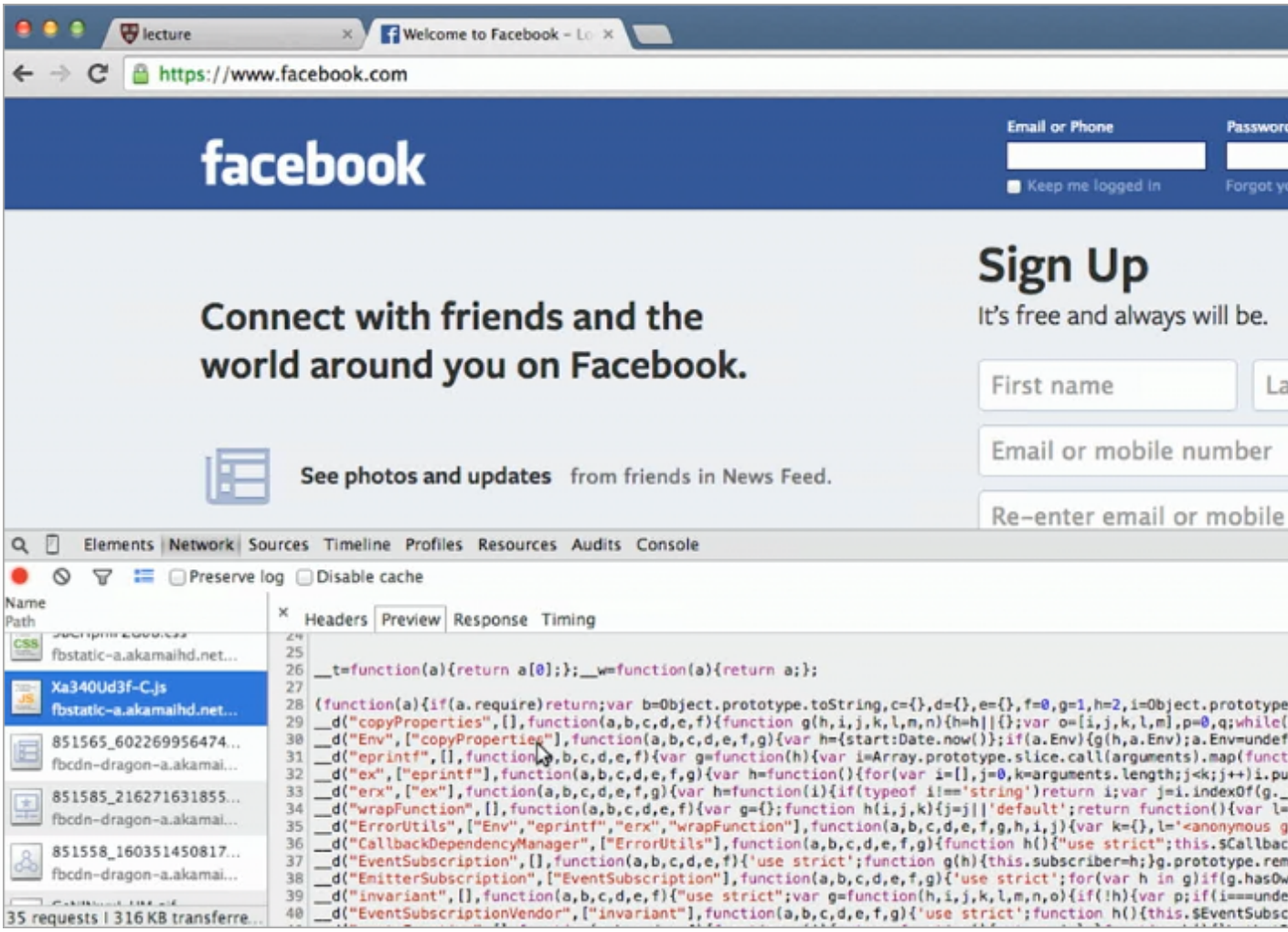
    # We'll see more of this, but this is the most common data structure in JavaScript, since you can associate key-value pairs, just like in PHP's associative arrays.

- JavaScript is an interpreted language, not compiled, but unlike PHP, it runs on the client side, in the browser. It'll be saved on the server, but the browser will download it as code and execute it on the client's computer. Here are some **event handlers**, which just means that JavaScript can hook into some events on the browser:

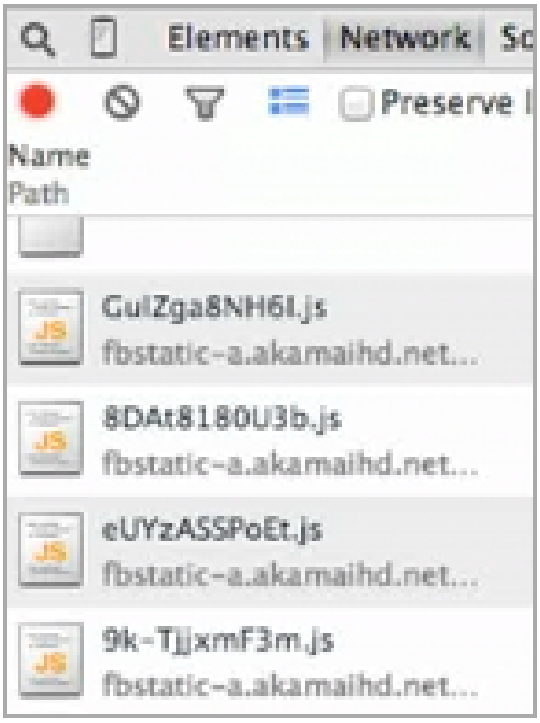
    # onblur
    # onchange
    # onclick
    # onfocus
    # onkeydown
    # onkeyup
    # onload
    # onmousedown
    # onmouseup
    # onmouseout
    # onmouseover
    # onmouseup
    # onresize
    # onselect

```
# onsubmit

# …
```

- If we look at the first `.js` file we request from visiting facebook.com, we can't really tell what it does, but see that there is a lot of code:



- In fact, we can see many more `.js` files in our **Networks** tab:



- Even though Facebook is written in a language based on PHP, there is a lot of JavaScript that controls, say, chatting and live updating of the News Feed.
- Let's look at today's source code[15], in particular `dom-0.html`[16]:

[15] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/js/
[16] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/js/dom-0.html

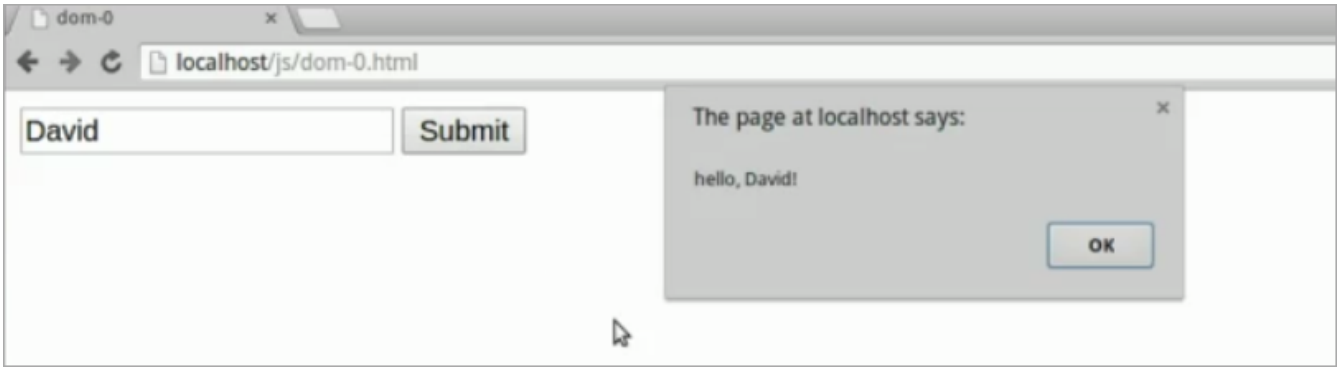
```
<!DOCTYPE html>

<html>
    <head>
        <script> ❶

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

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

In line 5, we have a new tag called `script` inside the `head` of the page, and inside it is a `function`, `greet()`. As an aside, JavaScript has nothing to do with Java.

In line 9, what our function does is `alert` the user, by means of a pop-up box, which you may have seen, and been annoyed by, before. Notice also that we used single quotes, though they are treated exactly the same as double quotes. `+` just concatenates two strings, which PHP has in the form of `.`. Then we access the `name` element (recall the Document Object Model (DOM) that we talked about before, with the tree of various pieces of an HTML page connected like a tree) with the `document.getElementById` syntax. `document` is a JavaScript global variable, and you can call one of its methods (like a function inside a variable) by using the `.` character. In this case, we're getting the `name` tag, and then its value.

Then on line 16 we create a `form` with an `id` of `demo` (which allows us to identify it with a unique name) and attach an event handler, `onsubmit`, to it, which will call the `greet` function and then `return false`, when the form is submitted.

- So the page ends up looking like this:



Remember that we named the field where we put our name `name`, so `greet` has access to it by way of `getElementById('name')`. And then we said `return false` after we called the `greet` function, which meant the page didn't refresh or go anywhere else.

- Let's look at `dom-1.html`[17]:

[17] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/js/dom-1.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>dom-1</title>
    </head>
    <body>
        <form id="demo">
            <input id="name" placeholder="Name" type="text"/>
            <input type="submit"/>
        </form>
        <script>

            document.getElementById('demo').onsubmit = function() { ❶
                alert('hello, ' + document.getElementById('name').value +
 '!');
                return false;
            };

        </script>
    </body>
</html>
```

Notice that we've moved the `script` from the `head` to after the `form`. Then we get the `demo` element, which is the form itself, and now we can set the `onsubmit` property (like a data field in a `struct` in C) to a function. So in JavaScript (and PHP, and technically C, even though you shouldn't) you can actually have anonymous (so-called lambda) functions that don't have names but can still be called. So now when the `submit` button is clicked, the browser will run that function, because it's attached to the `onsubmit` property for the form.

- We can make this look even fancier with a popular library called jQuery, in `dom-2.html`[18]:

[18] http://cdn.cs50.net/2014/fall/lectures/9/m/src9m/js/dom-2.html

```
<!DOCTYPE html>

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

            $(document).ready(function() {
                $('#demo').submit(function(event) {
                    alert('hello, ' + $('#name').val() + '!');
                    event.preventDefault();
                });
            });

        </script>
        <title>dom-2</title>
    </head>
    <body>
        <form id="demo">
            <input id="name" placeholder="Name" type="text"/>
            <input type="submit"/>
        </form>
  </body>
</html>
```

- On Wednesday we'll use this library to create web applications that update the page without refreshing and have other fancy features.