Week 9

This is CS50. Harvard University. Fall 2015.

Anna Whitney

Table of Contents

1. Announcements

- Today would have been Mr. Boole's (of Boolean logic) 200th birthday, so Google celebrates with a Boolean-themed Google Doodle.
- As usual, CS50 Lunch is Friday; sign up at cs50.harvard.edu/rsvp.
- If you're considering concentrating in Computer Science (or switching your existing concentration to CS), you should stop by the SEAS Advising Fair on Wednesday, 11/5, from 4-5:30 PM in Maxwell Dworkin (donuts will be available!).
- David shows us some embarrassing old websites he created.

2. JavaScript

- We'll give you the final piece of our web programming tools in this course now, which you may find particularly helpful for final projects.
- Recall that C is a compiled language, meaning that your code gets turned into 0s and 1s that a computer can understand before you run it.
- PHP, on the other hand, is an interpreted language, meaning that a program called an interpreter is figuring out what your code means line by line as you run it and directing the computer to do the correct actions.

- We also introduced HTML, which isn't a programming language at all - it's a markup language, so it doesn't have loops, conditions, etc, just formatting and ways of representing the content of a page.
- CSS is also not a programming language, and even more aesthetically oriented.
- SQL is in fact a programming language (albeit one specifically tailored to database manipulations), and you can write entire procedures (functions) in SQL, although in this course we stick to a small (but very useful) subset of what SQL can do.
- And now we'll formally introduce **JavaScript**, which, like PHP, is an interpreted language.
    # Unlike PHP, JavaScript can (and usually does) run **client-side** - that is, it's not being interpreted by your web server and served up to your site's users (server-side, like PHP), it's interpreted by the browser on your user's computer.
        # It is possible to write server-side JavaScript (and use it as the backend of any application you might otherwise write in PHP or some other language), using frameworks like Node.js, but in this course we're going to focus on client-side JavaScript.
    # This means anyone can see the JavaScript code on your site, because their browser needs to have that information so it can execute the JavaScript.

2.1. Syntax

- JavaScript is also quite syntactically similar to C and PHP.
- There's no main function - you just start writing your code.
- Conditions are syntactically identical to C and PHP:

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

Also exactly as in C and PHP, boolean conditions can be combined with `&&` or `||`.

- Switches are also syntactically equivalent to those in C and PHP:

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

    case j:
        // do that
        break;

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

- The loops we've seen before are structured the exact same way, too, such as the `for` loop:

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

- We have a new type of loop as well, though:

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

We'll refer to this as a **for-in** loop, sort of analogous to PHP foreach loop.

All this means is if we've got an object (which we'll describe more in a moment), we can iterate over the key-value pairs inside, with JavaScript grabbing a new key from the object each iteration. We can get at the value associated with that key by treating the object as an associative array and indexing into it by key as `object[key]`.

- Arrays are declared similarly to PHP:

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

Note that we don't prepend our variable names with dollar signs in JavaScript, though!

JavaScript, like PHP, is loosely typed, so we as the programmer don't need to specify our variable types.

- More generally, we can declare variables like so:

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

- And then we have **objects**, with keys and values, not unlike PHP's associative arrays (although with some differences):

```
var quote = {symbol: "FB", price: 101.53};
```

This object has two keys (`symbol` and `price`), each with a value.

Contrast this with the syntax for an associative array in PHP:

```
$quote = ["symbol" => "FB", "price" => "101.53"];
```

These are just syntactic differences - the key-value pairs work exactly the same way.

Just like in PHP, we could access the value associated with `symbol` via `$quote["symbol"]`, we can do the same in JavaScript with `quote["symbol"]`.

2.2. JSON

- Apart from these minor syntax issues, what can we actually do with JavaScript?
- Recall in Problem Set 7, we provide you with a configuration file called `config.json`. We could have used any of several different text formats to store the configuration information, but JSON has some advantages.

```
{
    "database": {
        "host": "localhost",
        "name": "pset7",
        "username": "TODO",
        "password":   "TODO"
    }
}
```

This is an object in JavaScript - there's one key-value pair, where the key is `database`, and the value is itself another object (with keys `host`, `name`, `username`, and `password`).

- JSON is a standard format for storing data - a little nitpicky (a JSON file won't be readable if we omit a comma or quotes, etc), but widely used and pretty flexible.

3. Document Object Model

- Recall that we can represent HTML as a tree diagram:

```
<!DOCTYPE html>                                    (document)
                                                       |
<html>                                               [html]
  <head>                                             /    \
    <title>hello, world</title>                   [head]  [body]
  </head>                                            |        |
  <body>                                          [title]  (hello, world)
    hello, world                                     |
  </body>                                        (hello, world)
</html>
```

- JavaScript will let us manipulate the DOM, this tree of relations between different HTML elements, directly.
  - # We don't have to code the tree data structure ourselves - the browser does it for us, and essentially hands us a pointer to the root of the tree.
- We can do all kinds of things with the elements of the DOM, like handling events.
  - # Recall from Week 0, in Scratch, we had an example where one sprite said "Marco!" and the other sprite listened for that and responded with "Polo!". This is an example of event handling.
  - # JavaScript lets us listen for all kinds of events related to DOM elements, such as `onclick`, `onsubmit`, `onmouseover`, and many, many more. An `onclick` listener waits for the user to click the element in question, and then does something when they do.
  - # For example, when you click and drag in Google Maps, the map moves around. They could implement this by listening for a `mousedown` event and a `drag` event, and implementing the appropriate behavior in response.
- Let's look at some actual code, in `dom-0.html`[1].
  - # When we type in the name "David" in the input box and click Submit or press enter, an alert window pops up saying "Hello, David!"
  - # If we type in a different name, the alert adjusts to show a different name.
  - # Alerts are pretty ugly and distracting, so for real sites we'll want a different way to say something to the user, but this shows one big difference between this method

[1] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/dom-0.html

and what we were doing in PHP - when we submit the form, we don't reload or go to another page!

If we check in Chrome's Network tab, we can see that there's no network traffic when we submit the form.

- Looking at the code itself, we can see that there's some HTML that we're mostly already familiar with, and a new `<script>` tag in the header:

```
<!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>
```

First looking just at the HTML, we can see that our `<form>` tag now has an attribute called `onsubmit` with the value `"greet(); return false;"`.

This quoted string is in JavaScript, calling a function called `greet`, which is defined up above.

`greet` calls the JavaScript function `alert`, which pops up that ugly alert window we saw.

We're using `+` to concatenate strings here, just like we used `.` in PHP (and again, MUCH easier here than in C!).

    # In between the string parts we can see, we have `document.getElementById('name').value`. Let's break this down some more.

        - Remember our DOM tree above? `document` here refers to the root node of the tree, which is a special global variable in JavaScript.
        - We have a function called `getElementById` that lets us traverse the DOM tree to find the element with the specified `id` attribute, here `name`.

            # Note that this function is inside the document object, which is why we call it with this dot notation (not unlike getting an element of a C struct).

            # This function hands back the node of the tree with the `id` we asked for, and then we can get various properties of that node - one of which is called `value`.

        - So at the end of all of this, we get back the string that the user typed in the box.

    # So why, after `greet();`, do we `return false;` in the `onsubmit` here?

        # We didn't specify an `action` for this form, but by default, that means the form submits to this same page (`dom-0.html`).

        # If we delete the `return false;`, reload the page, open up our Network tab and click "Preserve log", and then submit the form again, now we can see that the form is submitting back to this same URL and sending another HTTP request.

        # This is not what we want to do, so we `return false` to short-circuit the browser's default behavior and avoid reloading the page.

- Let's look at a marginally better version of this code, in `dom-1.html`[2]:

[2] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/dom-1.html.src

```
<!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() {
                alert('hello, ' + document.getElementById('name').value +
 '!');
                return false;
            };

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

This implementation behaves exactly the same way, but rather than having an `onsubmit` attribute that calls a function in the midst of our HTML, we have all of our JavaScript code inside our `<script>` element.

    # This is much more maintainable, because it's easier to find the code to change it if it's all in one place!

We're grabbing the form element using its `id`, once again with `document.getElementById`, and we're directly assigning its `onsubmit` attribute within our script, which we know is a valid event listener for a form element.

    # The value of the `onsubmit` property needs to be a function, but rather than name this function, write it up separately, and then call it, we're using a feature of JavaScript called an **anonymous function** here (PHP and C actually also have this feature - C via "function pointers" - but we never really went into them, and they're more commonly used in JavaScript).

- What more can we do to improve our code? Let's see in `dom-2.html`[3]:

```
<!DOCTYPE html>

<html>
    <head>
        <script src="https://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>
```

Now we're incorporating a very common JavaScript library called **jQuery**, which takes normally very verbose JavaScript code like `document.getElementById('demo')` and turns it into `$('#demo')`.

This is the equivalent way of doing the exact same thing as before in jQuery as opposed to native JavaScript.

This code roughly means "When the document is ready, set this anonymous function as the submit handler for the element called 'demo', which will pop up an alert and then return false."

[3] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/dom-2.html.src

    - Rather than actually `return false;`, here we have the line `event.preventDefault();`, which has the same effect - short-circuiting the browser's default behavior of submitting the form and reloading the page.

- Let's use this for something a little more useful in `form-0.html`[4]. We can set up a form, e.g. for the Frosh IMs website.

    - When we submit the form, whether we cooperate and type a real email address, matching passwords, and check the box, or not, this takes us to a page saying, "You are registered! (Well, not really.)".

    - We can certainly check for correct input in PHP, but then the request has to go all the way to the server just to tell the user they did something wrong and have to try again - it would be better if we could give the user more immediate feedback.

- In `form-1.html`[5], we use JavaScript to perform **client-side validation**, and actually check for correct input before the request can be sent to the server:

[4] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/form-0.html
[5] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/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');

            // 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 == '')
                {
                    alert('You must provide a password!');
                    return false;
                }

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

                // validate agreement
```

    # Now we detect when passwords do not match, or when the user did not agree to the terms and conditions without submitting the form.
        # This decreases server load, because it prevents erroneous input from wasting your server's time just to say that the input is wrong.
        # Especially on a mobile device, there can be significant latency to a call to the server, so it can be faster to validate input on the client side.
    # Inside our `<script>` tag, we're finding the form element using `document.getElementById`, and then registering an event listener with an anonymous function that checks various input.
        # Even if some of the syntax looks a little unfamiliar, the basic logic here is the same as what we might have used in C or PHP.
        # Note that the checkbox input type has a property called `checked` with a Boolean value indicating whether the box is checked or not.
    # If everything went well, we `return true;` and the form is submitted to the server; otherwise, if the user typed something wrong, we `return false;` and prevent submission of the form.

- In `form-2.html`[6], we have the jQuery version of the same code, which you can look at if you're interested in doing things that way for your final project.

4. Ajax and Asynchronicity

- If we go to Google Maps and scroll around the page really quickly, we can see that as we scroll to a new location, it's first full of blank squares before the actual map loads.
- This is an example of **Ajax**, or Asynchronous JavaScript And XML, which we can use JavaScript for going forward.
- On MapQuest, a 1990s precursor to Google Maps, if you wanted to scroll up on the map, you had to click an up arrow and the whole page would reload to show you the next square of the map upward.
- Ajax is a technique that uses JavaScript to send additional HTTP requests after the page has loaded, without reloading the whole page.

[6] http://cdn.cs50.net/2015/fall/lectures/9/m/src9m/form-2.html.src

    # For example, when you receive a new email in Gmail, the page doesn’t reload every time; the new email just appears.
    # Gmail traverses the DOM tree to find the inbox part of the page, and inserts a new node - a new HTML element - corresponding to the new email.
- If you go to Google (with Instant Search on) and start typing in a search query, a new request gets sent every time you add another character.
- To finish up today’s lecture, David calls up a volunteer to give a demo of a game on the Oculus Rift.