Computer Science 50 Week 8 Monday: October 26, 2009
Fall 2009 Andrew Sellergren
Scribe Notes

Contents

1 Announcements (0:00–1:00)

- Your (optional) challenge this week will be to top the Big Board! See if you can implement the fastest spellchecker in the class. Don't be shy, step right up! Drew Robb currently holds the first position, as he did last year, but see if you can beat him. Of course, realize it's completely optional, so you don't need to feel the need to butt heads with the rest of the class and the teaching staff. Purely for fun!
- Don't forget about office hours!

2 Web Programming (1:00–68:00)

- So far, we've worked almost exclusively in C, a so-called lower-level language which affords both great power and great danger thanks to its ability to manipulate memory directly. In the weeks to come, we'll be moving toward PHP and JavaScript, so-called higher-level languages which abstract away the details of memory manipulation and have a learning curve which is not so steep as that of C.
- As powerful as C is, it's certainly not the right language for web development. If you were to try to implement HarvardTweets or HarvardMaps using C, you would find yourself re-inventing the wheel many times over. Better to start with a language like PHP which has a lot of packages, frameworks, and toolsets already built in that will make your job that much easier.
- One of the downsides of C is that you'll effectively have to develop and compile your programs twice if you want them to run on both Windows and Macs. Not to mention, their output will not always be standard across both. Web development, on the other hand, is at least a little more standardized, although it is still subject to the shortcomings of the browser wars.[1]
- The language in which all web pages are written is HTML, or *hyper text markup language*. HTML is not a programming language, per se, since it can't implement any real logic. However, it specifies how the browser will render data which is passed to it. This rendering is controlled by tags such as `<b>`, which specifies bold text.
- Thanks to Wayback Machine, we can check out Frogman, one of David's early web endeavors. Besides the hideousness, you'll notice that the language hasn't changed much since 1998, when he wrote it.

[1] Firefox rules!

2.1 HTTP

- HTTP, or *hyper text transfer protocol*, is what dictates the communication between client and server. In other words, it's what determines how a request is sent by the browser and how a response is sent by the server.
- Every computer on the internet has a unique IP address,[2] which is a number of the form w.x.y.z, where w, x, y, and z are numbers between 0 and 255. If you do the math, that means each of the four numbers can be represented by 1 byte, so the full IP address can be represented by 4 bytes. Data on the internet is passed through routers, or gateways, whose sole purpose is to direct data from source to destination. Typically, data will pass through 30 or fewer stops along its way between point A and point B.
- Once a web page request reaches the server, the server returns the web page itself, which is really just a text file containing HTML. When the browser receives this file, it will parse it, which is to say that it will read through it and interpret the various tags within it in order to determine how to display it. If the web page specifies any other files as graphics or JavaScript or CSS, the browser will then make separate requests for each of those files to be downloaded.
- As an aside, try Googling recursion. Har har har.

2.2 XHTML

- XHTML, or *extensible* HTML, is the clean version of HTML. Over the years, HTML has gotten somewhat sloppy in terms of its standards—and browsers have enabled this by rendering sloppy HTML reasonably well. XHTML, for example, requires that every tag be explicitly opened and closed. Browsers, however, will still display HTML which has unclosed tags.
- Let's start writing some XHTML:

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

  If we save this file as `hello.html` and then double click it on our Desktop, it will be opened in our default web browser and display the text "hello,

[2](cough). Well, sort of.

world." Of course, it's not really a live web page, yet, because it's not actually on the web. Our friends and family won't be able to view our handiwork. What a tragedy!

- `cloud.cs50.net` is a web server which means it can host your website if you upload your files properly. If we SSH to the Cloud and create a directory called `public_html` and within it a file called `hello.html` that has the same code as above, we can then navigate to `http://cloud.cs50.net/~cs50/hello.html` (where `cs50` would be replaced with your username) to see our web page displayed. Somewhat confusingly the `~cs50` refers to the CS 50 user's home directory when we're using SSH but it refers to the CS 50 user's root web directory (`public_html`) when we're using a web browser.

- Actually, before the web page will properly display, we need to set the correct permissions lest we get a Forbidden error when we try to access it. To view the current permissions, we'll run `ls -l hello.html` and see the following output:

  ```
  -rw------- 1 cs50 cs50 103 Oct 26 13:33 hello.html
  ```

  This output tells us the file's last modified date and time as well as its owner, its user group, and its permissions. The first `rw` means that the owner, or the user `cs50`, has read and write permissions for the file. The hyphen afterward means that the owner does not have execute permissions. And the remaining six hyphens mean that no one else in the user or anyone else has read, write, or execute permissions for this file. This is why when we try to access the page from the web, we get a Forbidden error because we're accessing the file as "everyone else."

- To give "everyone else" permissions to read our web page, we run the following command:

  ```
  chmod 644 hello.html
  ```

  Now, when we execute `ls -l` again, we see that we've granted read permissions to everyone:

  ```
  -rw-r--r-- 1 cs50 cs50 103 Oct 26 13:33 hello.html
  ```

- Let's take a look at a slightly more complete version of `hello.html`:

  ```
  <!--

      /*****************************************************************
       * hello.html
       *
  ```

```
     * Computer Science 50
     * David J. Malan
     *
     * Says hello.
     *****************************************************************/

-->

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <head>
        <title>hello, world</title>
    </head>
    <body>
      hello, world
    </body>
</html>
```

Comments in HTML are demarcated by `<!--` and `-->`. The lines beginning with `<!DOCTYPE` are a *document type declaration*, or DTD. The World Wide Web Consortium (W3C) decreed that these lines (and slight variations thereof) will be used to designate the beginning of a web page. No need to memorize these lines—just remember to copy and paste them when you need them.

- One other difference between this and our previous version is the `xmlns` within the `html` tag. This is an *attribute* of the `html` tag. Suffice it to say that this needs to be included in order for our XHTML to be valid, but we won't discuss the reasons here and now since they're mostly uninteresting.[3]

- Let's start spicing up our website a bit:[4]

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <head>
        <title>hello, world</title>
    </head>
    <body>
      <b>hello</b>, world
      Welcome to my first website!
```

[3]Like conversations with David. Oh snap!
[4]Note from now on, we're omitting the comment and DTD for the sake of succinctness.

```
          I hope you enjoy your stay!
        </body>
    </html>
```

Also this does display all the text, it's not quite what we intended. What we really want is for each sentence to be on a separate line. To do that, though, we'll need to insert `<br />`, or line break, tags. While we're at it, let's center all the text using the somewhat arcane method of `center` tags:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        <center>
          <b>hello</b>, world
          <br />
          Welcome to my first website!
         <br />
          I hope you enjoy your stay!
        </center>
    </body>
</html>
```

Let's start using the web for what it was intended—to link information and pages to each other:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        <center>
          <b>hello</b>, world
          <br />
          Welcome to my first website!
         <br />
          I hope you enjoy your stay!
          <br />
           My favorite site is
           <a href="http://www.cs50.net/">www.cs50.net</a>
               and I can keep saying more stuff.
        </center>
    </body>
</html>
```

The `a` tag, with a URL specified as the `href` attribute, is used to make clickable links. Tags, or elements, can have zero or more attributes, as we've seen already. We might add a `bgcolor` attribute to the `body` tag and specify its value as red. Let's also add an image—heck, let's make it an animated GIF:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <head>
        <title>hello, world</title>
    </head>
    <body bgcolor="red">
        <center>
          <b>hello</b>, world
          <br />
          Welcome to my first website!
         <br />
          I hope you enjoy your stay!
          <br />
           My favorite site is
           <a href="http://www.cs50.net/">www.cs50.net</a>
               and I can keep saying more stuff.
           <br />
           <img src="http://www.animatedgif.net/computers/comp6_e0.gif" />
        </center>
    </body>
</html>
```

The `img` tag is an empty element, so we can open it and close it within the same tag, just like the `<br />` tag, which is shorthand for `<br></br>`. Here, we're pointing the `src` attribute to the absolute URL of the GIF, which is hosted elsewhere. We could also download the GIF, place it in our own web directory, and then specify the `src` using a relative URL like `computer.gif`. This is generally best practice since it's not exactly fair to the host of your external content to have to pay the price for bandwidth in transferring files to a site that's not his own. Of course, you should only be downloading images into your own web directory if they are not copyrighted, and you should always cite your source in an HTML comment. To actually upload the file to The Cloud, we can use an SFTP client like SecureFX or Cyberduck, as you probably did for Problem Set 5. By default, the SFTP client will upload the file with world-readable permissions, but in case it doesn't, just realize that you'll need to run `chmod` on it before it will properly display.

- What about the real world? If you want some of your work to live on beyond this course, you'll want to register your own domain name using a registrar like GoDaddy. In addition, you'll need to sign up for web hosting. Both are pretty cheap these days. For the final project, at least, if you'd like to have your own domain name, we can take care of the hosting. You'll simply need to tell GoDaddy, or whichever registrar you use, that the nameservers for your site will be hosted by `cloud.cs50.net`. More details on that are forthcoming.

- Fortunately, since XHTML can get pretty messy, there are tools which exist to make sure that your code is valid. You can pass your URL to the W3C Validator for example, and it will analyze your code for errors and make suggestions for how to fix it. We'll expect for Problem Sets 7 and 8 that your XHTML be valid.

2.3 CSS

- CSS, or *cascading stylesheets*, afford a more consistent and clean way of controlling the aesthetics of a web page. There are two ways to include CSS to be read by a browser: embedded within `style` tags directly in the page or included externally via `link` tags. For example, we might embed the following styles in `hello.html`:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
    <!--

body
{
    color: blue;
    font-size: 96pt;
}

.copyright
{
    font-size: 12pt;
}

    -->
    </style>
    <title>hello, world</title>
  </head>
  <body bgcolor="red">
    <center>
      <b>hello</b>, world
      <br />
      Welcome to my first website!
      <br />
      I hope you enjoy your stay!
      <br />
      My favorite site is
      <a href="http://www.cs50.net/">www.cs50.net</a>
          and I can keep saying more stuff.
      <br />
      <img src="computer.gif" />
      <br />
      <span class="copyright">Copyright CS50</span>
    </center>
  </body>
</html>
```

This has the effect of increasing the font size and turning the text of the `body` blue. We're also defining a CSS `class` to represent a copyright footer at the bottom. CSS has three ways of defining styles:

```
element
{
  property:  value;
  property:  value;
}
#id
{
  property:  value;
  property:  value;
}
.class
{
  property:  value;
  property:  value;
}
```

The *element* syntax allows control of the styles of tags like `body` or `html`. The *#id* syntax allows control of the styles of a tag whose `id` attribute is specified. For example, `#footer` would refer to a tag whose `id` was `footer`. Finally, the *.class* syntax defines a CSS class which controls the styles of any tags who have their `class` attribute so specified.

- One other property of CSS is that it's cascading, as is specified in the name. That means that properties can be overwritten when re-specified. For example, in the above, the `font-size` of the `copyright` class is set to be 12-pt. and, in fact, displays as 12-pt. because we've specified the `copyright` class below the `body` element. So even though the text initially inherits a `font-size` of 96-pt. because it is part of the `body` element, this value gets overwritten below it.

2.4 Forms

- Forms are what allow us to take user input and generate dynamic content—in fact, this is what all user-drive websites boil down to. Here are some of the types of forms we'll be implementing this semester:
  - Text Fields
    ```
    <input name="email" type="text" />
    ```
  - Password Fields
    ```
    <input name="password" type="password" />
    ```
  - Checkboxes
    ```
    <input name="save" type="checkbox" />
    ```
  - Radio Buttons
    ```
    <input name="gender" type="radio" value="F" />
    <input name="gender" type="radio" value="M" />
    ```
  - Drop-Down Menus
    ```
    <select name=state">
      <option value=""></option>
      <option value="Matthews"></option>
      <option value="Weld"></option>
    </select>
    ```
  - Text Areas
    ```
    <textarea name=comments"></textarea>
    ```
- Let's examine the source code for Google's home page. If we boil it down to just the form elements, we are left with the following:
  ```
  <form action="/search" name=f>
  <input name=hl type=hidden value=en>
  <input autocomplete="off" maxlength=2048
         name=q size=55 title="Google Search" value="">
  <br>
  <input name=btnG type=submit value="Google Search">
  <input name=btnI type=submit value="I'm Feeling Lucky">
  </form>
  ```

For all practical purposes, this is all it takes to implement Google search on the front end! Notice that this isn't valid XHTML because they don't have quotation marks surrounding a lot of their attribute tags. However, consider that those extra few quotation marks would probably cost them billions of bytes per day.

- The only input that's really worth mentioning is the text field that contains the user's query. The name of this input field is simply `q`.
- So let's re-implement Google on our own site. All we need are the following lines, as written in `fakegoogle.html`:

```
<!--

    /****************************************************************
     * fakegoogle.html
     *
     * Computer Science 50
     * David J. Malan
     *
     * Re-implements Google.
     ****************************************************************/

-->
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Fake Google</title>
  </head>
  <body>
    <center>
      <h1>Fake Google</h1>
      <form action="/search">
        <input size="96" type="text" name="q" />
        <br />
        <input type="submit" value="Fake Google Search" />
        <input type="submit" value="I'm Feeling Lucky" />
      </form>
    </center>
  </body>
</html>
```

Let's step back for a second. If we type "cs50" into Google's search box, we can see that it gets passed to Google's servers via a string `q=cs50` which is actually embedded in the URL itself. As it turns out, the `q` pertains to the `name` attribute of the `input` element.

- In our implementation of Fake Google, we get a series of error messages when we click Fake Google Search. We can see that our query was added to the URL just as it was when we searched Google. But nothing came of our query on our own website. We can change this by changing the `action` attribute of the form to point to http://www.google.com/search.

- Although we'll steer clear of the browser wars,[5] we encourage you to use Firefox for Problem Sets 7 and 8 because of the debugging tools it offers. If we open up a plugin called Live HTTP Headers, we can see what is actually be transmitted across the wire when we execute a Google search:

```
GET /search?q=cs50 HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1.3)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: SS=Q0=Y3M1MA; PREF=ID=611ad8917d3c725b:U=0bc3186823857b88:FF=4...

HTTP/1.x 200 OK
Cache-Control: private, max-age=0
Date: Thu, 29 Oct 2009 08:22:18 GMT
Expires: -1
Content-Type: text/html; charset=UTF-8
Set-Cookie: SS=Q0=Y3M1MA; path=/search
X-Content-Type-Options: nosniff
Server: gws
Transfer-Encoding: chunked
Content-Encoding: gzip
X-XSS-Protection: 0
```

  This first line which is displayed is a `GET` string and represents the actual HTTP request which is transmitted from our browser to Google's servers.

- Next week, we'll begin looking at implementing our own user-driven website, starting with an example which allows users to register for Frosh IMs. This was, in fact, David's first experience with programming, for when he was a freshman,[6] the system was still paper-based.

[5] Firefox rules!

[6] I would say that was in 1960, but frankly, I use that joke every year, and it makes me sad that I don't have new material.