Week 9, continued

This is CS50. Harvard University. Fall 2015.

Anna Whitney

Table of Contents

1. Introduction

- Today's lecture is being shot in 3D!
- David calls volunteer Ariana up to demonstrate watching a video trailer on the Oculus Rift.
- Donuts available today at 4 PM at the SEAS Advising Fair in Maxwell Dworkin.

2. Anonymous Functions and Callbacks

- An **anonymous function** is a function without a name. It can be bound to an event, or passed to another function.
- When we pass a function like this (anonymous or otherwise) to another function or to an event handler to be called when something happens or when some data is ready, it's called a **callback** (or callback function).

3. Ajax

- Ajax once upon a time stood for "Asynchronous JavaScript And XML", but now refers more generally to the technology whereby a browser can request more data from a server after a page has already loaded, without reloading the whole page.

- In Problem Set 7, you're not using Ajax - when the user fills out a form or clicks a button and you generate dynamic content, that content is being generated entirely on the server and the browser reloads the page or goes to a new page to show it to you.
- However, in Problem Set 8, you'll use JavaScript and Ajax to make dynamic changes a little more seamlessly.
- There was, once upon a time in the '90s, an HTML tag called `<blink>` that did just that - made text blink on the screen. Today's browsers don't support it, because it's heinous, but we can reimplement it in JavaScript, as in `blink.html`[1]:

[1] http://cdn.cs50.net/2015/fall/lectures/9/w/src9w/blink.html.src

```
<!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>
    <body>
        <div id="greeting">
            hello, world
        </div>
    </body>
</html>
```

    # We've given the `<div>` a unique identifier of `greeting`, so we can access it by `id` in either CSS or JavaScript.

    # We then use CSS in a `<style>` element to make the text big.

    # But we also have a `<script>` tag, in which we're actually going to make the text blink.

        # In the `blink` function, first we use `document.getElementById('greeting')` to get a reference to the node of the DOM tree corresponding to the text we want to blink.

        # We can use JavaScript to manipulate the CSS properties of HTML elements! We then check whether the text is currently hidden and make it visible if so, or vice versa. But this only changes the visibility of the text once, not repeatedly, so we're not quite blinking yet.

        # We use a function called `window.setInterval`, where `window` is a global object not unlike `document` that refers to the entire window rather than the contents of the page. `setInterval` lets you set a function to be called repeatedly at some specific interval, so we're calling `blink` every 500 ms.

- Now let's do something a little more interesting. In Problem Set 7, you've implemented a quote page, where the user can type in a stock symbol and get back the current price of the stock, by sending an HTTP request to a PHP page that gets that price from Yahoo.

    # Instead of reloading, going to another page, etc. when we make that HTTP request, we can use Ajax to give us the stock price without going anywhere else, as in `ajax-0.html`[2]:

[2] http://cdn.cs50.net/2015/fall/lectures/9/w/src9w/ajax-0.html.src

```
<!DOCTYPE html>

<html>
    <head>
        <script src="https://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" autofocus id="symbol" type="text"/>
             <br/><br/>
             <input type="submit" value="Get Quote"/>
         </form>
     </body>
 </html>
```

We're using the `onsubmit` event handler to get the quote, and then we `return false;` to prevent the page reloading.

The `quote` function calls `$.getJSON` with a URL based off `quote.php`, which David wrote earlier to format the stock information as JSON. We're using HTTP `GET` to pass the symbol to `quote.php`, getting the symbol the user entered from the element with id `symbol` (don't worry too much about the JQuery syntax here if unfamiliar - it's a useful library to get to know for a final project, though).

- Note that the `$` in `$.getJSON` is just an alias for `jQuery`, rather than a special character like in PHP.

The first argument to `getJSON` is the URL, and the second is a callback function!

- The purpose of a callback function is that the Internet can be slow, so if you go to get data from somewhere else, you don't want your entire webpage to hang until the data comes back - instead, you can tell the browser what to do when it gets the data back using a callback function, so everything else on the page still works while waiting for the data.

This is pretty ugly, though, since it pops up an alert with the price. We can do much better and embed the result in the actual HTML of the page, much more akin to how the websites you're familiar with - Facebook, Gmail, etc - actually go about using Ajax. See `ajax-2.html`[3], which is pretty similar except for the following changes:

In the HTML of the page, we've added a new `<span>` (like a `<div>`, but inline) called `price`:

```
...
<form onsubmit="quote(); return false;">

 Symbol: <input autocomplete="off" autofocus id="symbol" type="text"/
>
    <br/>
    Price: <span id="price">to be determined</span>
    <br/><br/>
    <input type="submit" value="Get Quote"/>
</form>
...
```

And in our script, we've changed our callback function a little:

[3] http://cdn.cs50.net/2015/fall/lectures/9/w/src9w/ajax-2.html.src

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

All we're doing here is getting the element with id `price` and updating its HTML contents.

Note that because we're changing the HTML directly, we have to be careful about sanitizing our inputs for security reasons.

4. Giza Project

- Professor Peter Manuelian speaks about the convergence of computer science and archaeology.
- At Giza, not far from modern Cairo, there are three pyramids, but many, many other tombs around them.
- We can think of Giza as having two major datasets - what's there and what's been taken to museums.

  # Both give us information about how the Egyptians lived and how they died.

- "Harvard Camp" was a collection of mud-brick huts not far from the pyramids, where the Harvard University-Museum of Fine Arts expedition was headquartered.

  # The HU-MFA expedition excavated at over 40 sites!

- What happens with the massive archaeological backlog that an expedition like this produces? Tons of manuscripts, glass plate negatives, etc. This is where the computers come in.
- We can apply a grid over the whole area, and we see that some parts of the grid were dug by the American (HU-MFA expedition) archaeologists, but there was also an Austrian expedition, and some Egyptian digs. We need to combine all this data.
- Each tomb or temple has many different types of data associated with it, which have been collected in a SQL database.

- This is complicated by damage to the sites, not just from the elements and the passage of time but also from intentional vandalism.
- Which is more likely to be readable many years from now - a hard drive, or a "really hard drive" (information carved in stone)? As information storage systems go, carvings in stone are particularly durable in the long term.
- From all the databases and information, the next step is visualization - letting people see the sites, using all kinds of 3D technology (including the Oculus Rift).
- We can go beyond visualization to recreation. Thanks to meticulous documentation, we can reconstruct the state of some of the furniture in the sites in 3D.
    # We can then use these virtual 3D reconstructions to make actual, physical 3D reconstructions so everyone can see what the furniture actually looks like.
    # This method is being used with Queen Hetepheres's chair, found in fragments at one site, which is currently being recreated and will be put on display at the Harvard Semitic Museum.
- How do we build computer models of *all* of Giza, not just individual tombs?
    # We can scan using aerial drones, particle detectors, radiography imaging, and more. The particle techniques let us see inside the pyramids, while the drones can put together complete 3D scans.
- Some of this information is online in slightly outdated form at gizapyramids.org[4]; it's being updated and could use your help.
- There's also a 3D site (unfortunately does not work on Macs) at giza.3ds.com[5].
    # You can explore many sites, including at different points in time. There are narrated tours, photos, and wireframes showing how the 3D reconstructions were made, as well as the actual 3D models you can explore.
- Massive amounts of data like this need to be organized well to make them accessible to non-Egyptologists.
- Some suggested CS50 Giza archaeology project topics:
    # Build a mobile app tour guide for the Hetepheres chair

[4] http://www.gizapyramids.org
[5] http://giza.3ds.com

    # Create a better thesaurus tool to add descriptive content to http://www.gizapyramids.org/search/advanced/Photos?t:state:flow=cc91c7d3-41ee-499c-b2a9-1b8a8e379598
    # Try a Google Cardboard app: Immersive stereo 3D content (we have sample bubble images)
    # Create a web tour of "Harvard Camp" dig house
    # Build a tomb in 3D Studio Max or Maya and animate it
    # Build a "my Giza" feature for the new website (saved collections)
    # Build a better way to handle our long list of PDF bibliography items[6]
    # Build a website guide to Giza or Egyptian content in the Harvard Semitic Museum
- There will be a Gen Ed course next semester that covers significant amounts of Egyptian history.
- There is still exploration ongoing at Giza, which we want to incorporate into a central location online.

[6] http://www.gizapyramids.org/static/html/authors_list.jsp